

MATHESIS

Elements of Learning and Intelligence

Volume I: Synthesis and Coherence

MATHESIS

(ΜΑΘΗΣΙΣ)

Elements of Learning and Intelligence

Volume I: Synthesis and Coherence

John A. Drakopoulos

(Ιωάννης Α. Δρακόπουλος)

Copyright© 2022 by John A. Drakopoulos

Printed in the United States of America

All rights reserved. No part of this publication may be reproduced, distributed, stored in a database or retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning, or otherwise, without the prior written consent of the Author.

Limit of Liability/Disclaimer of Warranty: While the publisher and author have used their best efforts in preparing this book, they make no representations or warranties with respect to the accuracy or completeness of the contents of this book and specifically disclaim any implied warranties of merchantability or fitness for a particular purpose. No warranty may be created or extended by sales representatives or written sales materials. The advice and strategies contained herein may not be suitable for your situation. You should consult with a professional where appropriate. Neither the publisher nor author shall be liable for any loss of profit or any other commercial damages, including but not limited to special, incidental, consequential, or other damages.

Library of Congress Cataloging-in-Publication Data

Drakopoulos, John (Ioannis) A. 2022 –

Mathesis. Elements of learning and intelligence. Volume I: synthesis and coherence.

Includes bibliography and index.

ISBN 9798802410592

1. Neural networks. 2. Machine learning.

I. Title. II. Series. Mathesis series; 3, etc.

To my family and dear friend and mentor,
Theodore Tomaras

Contents

Preface	v
Acknowledgments	ix
1 Introduction	3
1.1 A Progressive Approach	3
1.2 Notation	4
1.3 Directed Graphs	7
1.4 Defining Neural Networks	11
1.4.1 A Note on Terminology	16
1.4.2 The Polynomial Functions	17
1.4.3 Activation Functions	18
1.4.4 A Note on Unidimensionality	21
1.4.5 Input Types	21
1.5 Locality	22
1.5.1 Receptive Fields	24
1.5.2 Neural Locality Types	25
1.5.3 The Pale Fire of Convolutional Neural Networks	27
1.6 Historical and Bibliographical Remarks	33
1.6.1 Attribution	34
2 Learning	37
2.1 Change	37
2.2 Machine Learning	38
2.3 Learning Paradigms	40
2.4 Learning as Optimization	44
2.4.1 The Back-Propagation Method	48
2.4.2 Neural Stochastic Gradient Descent	49
2.5 Loss Functions	50
2.6 Computational Limitations	57
2.7 Historical and Bibliographical Remarks	59
3 Foundational Principles	65
3.1 The Imperative of Biology	65
3.2 Cell Theory	68

3.3	Neuron Theory and Elementary Neural Theory	69
3.4	Energy	72
3.5	Heredity	80
3.6	Homeostasis	81
3.7	Evolution	82
3.8	Conclusion	82
3.9	Historical and Bibliographical Remarks	83
3.9.1	Attribution	85
4	Synthesis	87
4.1	Structure	87
4.2	Synergy	89
4.3	Graph Cuts	91
4.4	Neural State and Synergy	93
4.5	A Neural Algebra	96
4.5.1	Neural Output	97
4.5.2	Neural Input	98
4.5.3	Neural Parallelism	98
4.5.4	Neural Repetitions	99
4.5.5	Neural Equivalence	99
4.5.6	Neural Synergy	100
4.5.6.a	Parallel Neural Synergy	100
4.5.6.b	Neural Rigidity	101
4.5.6.c	Sequential Neural Synergy	103
4.5.6.d	Neural State Selection	103
4.5.7	An Algebra for Synthesis	104
4.5.8	Is an Algebra Necessary?	106
4.6	Cooperative Learning	107
4.6.1	Homologous and Heterologous Learning	109
4.7	Cooperative Game Theory	112
4.8	Examples of Synergetic Learning	113
4.8.1	Ensembles	113
4.8.2	Generative Networks	116
4.9	Diversity	119
4.9.1	Voting and Condorcet's Theorem	122
4.9.2	A Theoretical Justification for Diversity	124
4.10	The Synergy Conjecture	127
4.11	Extrinsic Diversity	129
4.12	Definitions of Diversity	132
4.13	Diversity Algorithms	136
4.14	Conclusion	139
4.15	Historical and Bibliographical Remarks	139
5	Coherence	143
5.1	Split Notation	143

5.2	Concepts from Set and Measure Theory	148
5.2.1	Measure Theory	151
5.3	Estimators	152
5.4	Generalization	154
5.5	Is Generalization Impossible?	156
5.6	Regularization	161
5.6.1	Razors and Minimum Descriptions	161
5.6.2	Dropout	162
5.6.3	Network Pruning	164
5.6.4	Metric Regularization and Weight Decay	165
5.6.5	Batch Normalization	167
5.7	Overfitting and Generalization	168
5.7.1	Functional Distinction	170
5.7.2	Non-local and Locality-sensitive Approximation	174
5.7.3	Finite Differences	183
5.8	Coherence	184
5.9	Notes on Coherence	196
5.10	Locality as Coherence	197
5.11	Historical and Bibliographical Remarks	200
6	Operators for Coherence	207
6.1	Of Correlations and Convolutions	207
6.2	Finite Recursion	211
6.3	Recursion versus Correlation and Convolution	214
6.4	Correlation and Convolution Examples	221
6.5	Recursion and Coherence	228
6.6	Transformation Bias	231
6.7	Historical and Bibliographical Remarks	232
7	Coherent Learning	235
7.1	The Case for Recursion	235
7.1.1	μ -Recursive Functions	235
7.1.2	Recursion in Learning	238
7.2	Synaptic Coherence	240
7.3	Synaptic Gradient Coherence	247
7.3.1	Momentum Gradient Descent	247
7.3.2	Synaptic Cross-Gradient Coherence	250
7.4	The Syc Algorithm	252
7.5	Neural Coherence	254
7.6	Factorized Neural Coherence	255
7.7	The Sync Algorithm	256
7.8	A Research Note	262
7.9	Historical and Bibliographical Remarks	262
8	Preliminary Evaluation	265

8.1	Synaptic Design	265
8.2	Striders, Corollas, and Flowers	267
8.3	Growing Flowers	272
8.4	Data Superposition	275
8.5	MNIST	277
8.6	Implementation Woes	287
8.7	Mathesis for MNIST	289
8.7.1	SE Block Extension	289
8.7.2	Local Pooling	290
8.7.3	CSE Blocks with Dropout	291
8.7.4	Squeeze-and-Excitation Flower Nets	292
8.7.5	Diversity in the Ensemble	294
8.7.6	Data Coherence	299
8.7.7	Training and Testing the SEF Ensemble	302
8.8	Publication Bias	306
8.9	Flowers for ImageNet	307
8.10	The N Programming Language	307
8.11	Historical and Bibliographical Remarks	311
9	Science in Crisis	313
9.1	The Perils of Empiricism	313
9.2	Metascience	315
9.3	Replication Crisis	318
9.4	Ulterior Motives	321
9.5	Medical Dark Age	324
9.6	Preparing for the Future	330
9.7	Historical and Bibliographical Remarks	338
	List of Figures	343
	List of Tables	345
	Bibliography	347
	Alphabetical Index	379

Preface

Journeys and Destinations

Mathesis (Hellenic *Μάθησις*, “learning”) is an interdisciplinary theory for learning and intelligence that combines biology, neuroscience, computer science, engineering and various branches of mathematics. It is largely a formal theory that attempts to unify all learning under a common framework, deriving its axiomatic basis and approach from nature and biology. It is also a theory that demonstrates the strength of mathematics, the imperative of biology in learning, and the potency of interdisciplinarity over complexity. The overall effort, which has spanned more than three decades, arguably demonstrates the power of perseverance, too.

In science and education, we teach theories and scientific facts. It is an approach based on reduction and economy, focusing on the end results and ignoring the often long and arduous process that led to them. It allows education to cover more material within any given curriculum. Thus students learn about gravity, calculus, and evolution but they hear very little or nothing at all about the observations, the early ideas and motivations, the trials and failures, and the personal struggle of scientists such as Newton, Leibnitz, or Darwin in their effort to form and deliver their groundbreaking theories. We may metaphorically argue that education focuses on destinations rather than journeys.

Yet, our understanding of a subject is significantly enhanced when we travel the path that led to it. For those who aspire to make scientific contributions, the journey of discovery and invention, the trials, mistakes, and corrections are as important as the end result. They provide context and structure and they help us understand the magnitude of the contribution in a more complete way. They are instructive and guiding. They can help us continue on the journey, avoid pitfalls, and advance upon the endless landscape of science. Related biographies, such as those for Newton and Darwin [363, 37, 38], contain a great deal of historical information that is often neglected but is an essential part of the overall effort behind the end result.

In this book series, we attempt to combine the journey with the destination. We gradually present the theory and the process that led to it. We provide some theoretical results and we sometimes include an initial empirical evaluation. By necessity, most of the experiments are omitted. Some of them had a formative effect but primarily because they were missteps. The

whole journey and the theory consist of three volumes that are indicated with Roman numerals. The first volume represents the first leg of the journey with our initial seaworthy vessels: *synthesis* and *coherence*. Those represent a natural beginning. Synthesis is required in order to create complexity from simplicity and coherence is necessary to manage the complexity.

We will progressively build the theory and significantly augment it in each volume. Of course, we will not present all the meandering dead ends and failed attempts that led to the end result, but we will discuss or prove limitations and intuitions that eventually guided the overall structure and design. We will also include material and mathematical concepts and tools that we deem important for the extension and evolution of the theory. Some segments of the theory are still at an early phase and they will most likely change as we develop the theory. Hopefully, those will not create inconsistencies between the volumes. It is all part of the nature of research. Constructs that initially seem great may not necessarily survive in the end.

We will also construct the theory from scratch. This series is practically self-contained. Only some familiarity with calculus, linear algebra, and probability theory is required. Set theory, graph theory, and theoretical computer science (as well as game theory to some extent) provide important components but they are not prerequisites. We will introduce related concepts and mathematical tools and prove associated theorems as needed. The theory borrows heavily from computer science, biology, and neuroscience but those are not prerequisites either. There is at least one concept that derives directly from quantum mechanics but we will present it independently and generalize it as well.

Throughout this series, we use the term *learning* to refer to the fields of machine learning and neural networks.* Those are currently driven by empiricism. There have been a number of theoretical results but they are limited, disconnected, and insufficient as a unifying or guiding theory. Without such a theory, it is often hard to distinguish between important empirical evidence and incidental results or artifacts; vision and direction can be compromised and confused.

It is not uncommon practice for a field to start on the empirical side or gradually drift into that direction. Empirical evidence is essential for the development of effective theories. However, pure empiricism has clearly overstayed its welcome in learning. A theory has long been overdue.

Mathesis attempts to fill a current theoretical gap, redefine the field in a rigorous way, and make the imperative connection with biology. The latter is paramount not only because nature is currently the only source of intelligence on earth but also because it has accumulated an extraordinary amount of structure and neural sophistication into living organisms. Nervous tissue has evolved for about 550 to 600 million years. A time scale of such magnitude

*We will use the term *biological neural networks* to refer to neural networks of living organisms.

cannot be replaced by engineering practices and principles. The connection is indeed imperative. Nature and biology are invaluable empirical treasure chests. Without them this theory would have advanced at a slower pace and, most likely, would still be in one of its early and rather disoriented forms.

Principles, Implementation, and Plans

Volume I of Mathesis is devoted to the first two foundational and axiomatic principles of the theory: synthesis and coherence. They imply growth, synergy, and diversity.

Volumes II and III will introduce more principles and grow and improve the theory. They will be published on separate timelines.

The source code will be released within a reasonable time frame after the publication of the first volume. The implementation is currently in a partial state with complete and incomplete parts; and so is the evaluation. The third portion of the theory is also incomplete. We hope that the community will undertake some of the burden, evaluate various segments of the theory, its algorithms and structures, and amend and improve as appropriate.

Coherence seems to have implications for biology and even physics. A number of phenomena can be understood as transitions or transfers of coherence. It is not clear whether those are merely human interpretations or inherent properties. The subject requires further research; but it is beyond the scope of this publication and the proximate plans and interests of the author.

Acknowledgments

This book series and the underlying theory are the outcome of decades of research and experience with machine learning and neural networks as well as an accumulation of interdisciplinary knowledge. It is clearly an individual effort, but like many previous scientific endeavors, it affords a further view upon a field because it is an extension of other theories and contributions, the “shoulders of giants” per Isaac Newton.

This theory has been directly influenced by the collective work of David Hubel and Torsten Wiesel, David Rumelhart and James McClelland, Jürgen Schmidhuber, David G. Stork, the Tablet PC group of Microsoft, and more recently Peter Sterling and Simon Laughlin.

In approximate chronological order, the following people have also influenced the theory in a different and more indirect way through teachings and mentoring, discussions and advice, collaboration, support, and mere chance: Theodore Tomaras, Manolis Katevenis, John Hennessy, Nils Nilsson, Barbara Hayes-Roth, Marc Levoy, George John, Ahmad Abdulkader, Gordon Rios, Heather Alden, Japjit Tulsi, Sanjeev Katariya, and Balamurali Meduri.

Theodore Tomaras provided valuable advice and assistance through most of this long journey. He was a mentor, guiding light, and dear friend. Ahmad Abdulkader offered suggestions and intuition that helped stay on course or steer clear of various obstacles. Heather Alden pointed to what was hidden in plain view. Sanjeev Katariya was the first person to see a large part of this work and his enthusiastic reception of it became a singular factor in the decision to publish the theory. The overall effect of the above individuals on the formation of the theory and my career can only be underestimated.

Gordon Rios and Sanjeev Katariya reviewed the manuscript of the first volume. Heather Alden also reviewed the last chapter.

It is very sad to recall that D. Rumelhart, N. Nilsson, T. Tomaras, and A. Abdulkader passed away during the long development of this theory. The world is less without their kindness and wisdom.

Finally, many thanks to eBay for providing a great work environment as well as hardware and resources to implement and evaluate some parts of the theory. A completely new, redesigned, and more extensive implementation is currently developed on our own hardware using Pytorch. A limited version was implemented in TensorFlow and a number of experiments were performed on our own hardware as well as on Google Colaboratory. Gordon Rios ran some of the experiments on his own computer and Google Colaboratory.

The biggest innovations of the twenty-first century will be the intersection of biology and technology. A new era is beginning.

— Steve Jobs, 2011.

1.1	A Progressive Approach	3
1.2	Notation	4
1.3	Directed Graphs	7
1.4	Defining Neural Networks	11
	1.4.1 A Note on Terminology	16
	1.4.2 The Polynomial Functions	17
	1.4.3 Activation Functions	18
	1.4.4 A Note on Unidimensionality	21
	1.4.5 Input Types	21
1.5	Locality	22
	1.5.1 Receptive Fields	24
	1.5.2 Neural Locality Types	25
	1.5.3 The Pale Fire of Convolutional Neural Networks	27
1.6	Historical and Bibliographical Remarks	33
	1.6.1 Attribution	34

1

Introduction

1.1 A Progressive Approach

Current efforts to create intelligent systems can be classified into two broad categories or paradigms.

The first is dominant in machine learning and neural networks. It uses simple algorithms, optimization, and empirical evidence to create increasingly more complex systems. There is no guiding theory and the efforts are primarily driven by empiricism. Consequently, it is hard to distinguish between true evidence and artifacts, progress is limited and incremental, there is no overall sense of direction, and research sometimes drifts to simplistic or ephemeral methods and architectures (such as support vector machines in the 2000s or excessively deep neural networks more recently). Without a theory, empiricism is effectively random search and research is often devoid of vision. The overall approach is equivalent to guesswork, and in the long term, it can only become an impediment to progress.

The second category is more closely related to neuroscience. It attempts to understand and reverse engineer the human brain (or other brains or their parts) in order to create a technology that has similar capabilities. However, the extraordinary complexity of the brain has hindered progress and limited success. Despite great advances and a number of breakthroughs in neuroscience, the human brain largely remains a mystery. Detailed mathematical models of essential cognitive functions are highly incomplete, approximate, or not available at all.

In Mathesis we follow a different approach. We treat learning and intelligence – and all of current life and nature – as a tentative outcome of a long and indefinite evolutionary process. Instead of focusing on incidental empiricism or trying to replicate complex outcomes directly, we propose that we identify the principles and the forces behind the underlying processes, combine them in a unifying theory as axiomatic truths, and let the theory guide the design, structure, growth, and evolution of the systems. We furthermore propose that we start with an initial and approximate theory and that we augment

and amend it as we go along. This progressive approach allows us to initiate the process without requiring a comprehensive understanding of learning and intelligence in advance. (The opposite leads to a cycle, a chicken-or-the-egg problem for learning.)

The three volumes of this book series are such a progression. An initial theory is introduced in volume I. It is then augmented and improved in volumes II and III. The goal is a general theory of learning – and eventually intelligence. The process is not meant to terminate in volume III but rather to reach a landmark that may be otherwise hard or inaccessible. And the theory is not meant to replace evolution but rather to initiate, fuel, and drive it.

1.2 Notation

Neural networks admit an algebraic (or functional) definition as compositions of functions. They also admit a graphical definition as directed graphs. The algebraic definition is elegant and affords simple and concise forms but it becomes somewhat cumbersome later on with coherent and adaptive structures. The graphical approach is not as concise but it is more resilient.

In this book series, the two definitions are combined into one. The combination may seem elaborate and unnecessary at first but it will become increasingly more effective as we augment and evolve the theory with new and arguably necessary elements.

Let us now introduce some basic notation before we proceed to the definition.

$\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}, \mathbb{C}$ are the sets of natural numbers, integers, rational numbers, real numbers, and complex numbers, respectively.

$\mathbb{R}_+ = \{x \in \mathbb{R} / x > 0\}$ is the set of positive real numbers.

$\mathbb{N}_+ = \mathbb{N} - \{0\}$ is the sets of positive natural numbers.

$\mathbb{N}_d = \{n \in \mathbb{N} / n < d\}$ is the initial segment of $\mathbb{N}$ with size d .

$\lfloor x \rfloor, \lceil x \rceil, \lceil x \rceil$ are the floor, ceiling, and rounded value of $x \in \mathbb{R}$. Thus $\lfloor x \rfloor = \max\{n \in \mathbb{Z} / n \leq x\}$, $\lceil x \rceil = \min\{n \in \mathbb{Z} / n \geq x\}$, $\lceil x \rceil = \lfloor x + 0.5 \rfloor$.

iff is a shorthand for “if and only if”

$\mathcal{F} \circ \mathcal{G} = \{f \circ g / f \in \mathcal{F}, g \in \mathcal{G}\}$ represents the composition of two sets of functions $\mathcal{F}$ and $\mathcal{G}$. It is the set of the functions of the form $f \circ g$, where $f \in \mathcal{F}$ and $g \in \mathcal{G}$.

(a, b) denotes a *pair* of two elements a and b .*

*A pair (a, b) is defined as $\{\{a\}, \{a, b\}\}$ in set theory. The term *ordered pair* is sometimes used to distinguish from the *unordered pair*, which is defined as $(a, b) \cup (b, a) = \{\{a\}, \{b\}, \{a, b\}\}$.

$(x_0, \dots, x_{n-1})$ indicates a *tuple* of n elements, which is defined inductively using the definition of a pair: $(x_0, \dots, x_{n-1}) = (x_0, (x_1, \dots, x_{n-1}))$. Thus, $(a, b, c) = (a, (b, c)) \neq ((a, b), c)$.

$[x_0, \dots, x_{n-1}]$ indicates an array of n elements. Arrays are special cases of tuples. The elements of an array must have the same type, size, and dimensionality while the elements of a tuple can be arbitrary. In other words, $[x_0, \dots, x_{n-1}]$ is a shorthand for $(x_0, \dots, x_{n-1})$ and the condition that all $x_0, \dots, x_{n-1}$ have the same type, size, and dimensionality. Equivalently, for all $x_0, \dots, x_{n-1}$, either $[x_0, \dots, x_{n-1}] = (x_0, \dots, x_{n-1})$ or the array $[x_0, \dots, x_{n-1}]$ is not defined. As a consequence, a function that is defined over tuples of a set X is also defined over arrays of the same set. The opposite is true if and only if the elements of X have the same type, size, and dimensionality.

When the elements of an array are arrays too, the array is *multidimensional*. An array can thus have arbitrary dimensionality.

$\emptyset$ denotes the *empty set* $\{\}$. Depending on context, it also denotes the *empty array* $[]$ as well as the *empty tuple* $()$. (Of course, it is $[] = ()$.)

$X^{d_0, \dots, d_{n-1}}$ denotes the set of all n -dimensional arrays over the set X with size d_k in their k dimension, for $k < n$. In the special case where $n = 1$, the set consists of unidimensional arrays (or tuples) that contain d_0 elements of X . In other words, $X^d = X \times X^{d-1} = \times_{i < d} X$, where $\times$ denotes the Cartesian product of two sets.

$X^* = \bigcup_{n=0}^{\infty} X^n$. This is the *Kleene star* notation for numeric sets and it denotes the set of all tuples over X . X^0 only contains the empty tuple and X^k contains tuples with k elements from X , for $k > 0$. (Thus $X^1 = \{(x)/x \in X\} \neq X$.) When the elements of X have the same dimensionality, tuples over X are identical to arrays over X and thus X^* also denotes the set of all arrays over X .

A function $f(x_0, x_1, \dots)$ with arbitrary number of arguments from a set X can be formally defined as a function over X^* . Then $f(x_0, x_1, \dots)$ is defined as an alternative notation for $f((x_0, x_1, \dots))$. (Interchangeable usage of $f((x_0))$ and $f(x_0)$ can create ambiguity when x_0 is a tuple. In that case, our convention is to use the form $f((x_0))$. In all other cases, there is no ambiguity, either form can be used, but we will usually use $f(x_0)$.)

$\mathbb{N}_{d_0, \dots, d_{n-1}} = \mathbb{N}_{d_0} \times \dots \times \mathbb{N}_{d_{n-1}}$ is the initial segment of $\mathbb{N}^n$ with size d_i on dimension i for $i < n$.

$i = i_0, \dots, i_{n-1} \in \mathbb{N}_{d_0, \dots, d_{n-1}}$ indicates an index for an n -dimensional array $A \in X^{d_0, \dots, d_{n-1}}$. The term *multi-index* is sometimes used for such

indices. Optional parentheses can be placed around an index for clarity but they are almost always absent when the index is applied to an array.

A_* indicates the set of all *permissible indices* of array A that access a single element of A . Indices to sub-arrays are not included. As an example, if $A \in \mathbb{R}^{d_0, \dots, d_{n-1}}$ then $A_* = \mathbb{N}_{d_0, \dots, d_{n-1}}$.

A_{**} indicates the set of all permissible indices of array A including indices to sub-arrays. As an example, if $A \in \mathbb{R}^{d_0, \dots, d_{n-1}}$ then $A_{**} = \bigcup_{k=0}^{n-1} \mathbb{N}_{d_0, \dots, d_k}$.

A^T indicates the *transpose* of a two dimensional array A , that is $(A^T)_{j,i} = A_{i,j}$ for all $(i, j) \in A_*$. Similarly, if x is a vector, x^T indicates its transpose (which is a single row).

Lexicographic ordering is a linearly ordered relation of tuples (and thus indices) over a linearly ordered set. It can be defined inductively:

$$(a_0, a_1, \dots) < (b_0, b_1, \dots) \text{ iff } \left\{ \begin{array}{l} a_0 < b_0 \text{ or} \\ a_0 = b_0 \text{ and } (a_1, \dots) < (b_1, \dots) \end{array} \right\}$$

$[f(j)/j \in J]$ indicates an array $A \in X^*$ such that $(\forall j \in J) A_j = f(j)$, where f is a function from J to X . In other words, the square-bracket notation imitates the dimensionality that is induced by the form of the indices in J . As a result, the equation $[A_i/i \in A_*] = A$ is a tautology. (Note that the above square-bracket notation may seem as a generalization of the previous unidimensional form $[x_0, \dots, x_{n-1}]$. However, the two forms are equivalent, for we can write $A = [A_0, \dots, A_{d_0-1}]$ where A_j is the sub-array j of A along its first dimension.)

$(f(j)/j \in J)$ denotes a tuple containing the values $f(j)$ for $j \in J$ in lexicographic order of the index j . Note that this tuple form does not retain the structure of the indices like the corresponding array form above but rather ‘flattens’ out the elements and sorts them in lexicographic order of their indices.

$(f(j)/j \in J \text{ and } \phi(j))$ denotes a tuple containing the values $f(j)$ for $j \in \{k \in J/\phi(k)\}$ in lexicographic order of the index j . In other words, $(f(j)/j \in J \text{ and } \phi(j))$ is $(f(j)/j \in J)$ restricted to the indices that satisfy the condition ϕ .

$I()$ denotes the identity function over arrays.*

Two indices i and j are *equivalent* with respect to an array or indexed entity A (denoted as $i \equiv_A j$) when they refer to the same element of A . If the same is true for all applicable arrays or indexed entities, we write that $i \equiv j$.

*In set theory, every number, tuple, or array is represented as a set. The identify function can thus be defined to be $I(x) = x$, for every set x .

In general, indices are assumed to be associative. Consequently, it is $(i, (j, k)) \equiv ((i, j), k) \equiv (i, j, k)$ for all indices i, j, k . For example, if A is an n -dimensional array and $(i_0, \dots, i_{n-1})$ is an index for it, then $A_{i_0, \dots, i_{n-1}} = (A_{i_0})_{i_1, \dots, i_{n-1}}$. The same applies to any contiguous partitioning of the index $i_0, \dots, i_{n-1}$ as long as the order of the elements remains the same.

Finally, if A is an array and S is a subset of the indices of A (that is $S \subseteq A_*$) then A_S can indicate two different entities depending on the structure of S . If S is a Cartesian product $S_1 \times \dots \times S_m$ for some sets $S_1, \dots, S_m$ then A_S denotes the sub-array of A that contains the elements of A indexed by S . In other words $A_S = [A_i / i \in S]$. However, if S cannot be written as a Cartesian product, A_S is the tuple containing of the elements of A that are indexed by S in lexicographic order of the indices. Thus $A_S = (A_s / s \in S)$.

1.3 Directed Graphs

In order to define neural networks as directed graphs, we shall need some material from *graph theory*.

A *directed graph* or *digraph* is a pair $G = (V, E)$ where V is a finite set and E is a subset of V^2 . The set V is the set of *vertices* or *nodes* of the graph and it is often a finite subset of $\mathbb{N}$. The set E is the set of *edges* of the graph – also called *directed edges* or *arrows*.

Graphs are just visual representations of binary relations over finite sets. For any edge $(u, v) \in E$, the corresponding pair belongs in the relation represented by the graph. In the graph, the two vertices appear connected with an arrow and v is said to be a *direct successor* of u while u is a *direct predecessor* of v .

We write $G(u)$ to denote the set of direct successors of u in G . We also write $G^{-1}(u)$ to denote the set of direct predecessors of u in G .*

We can extend those definitions to sets of vertices. Thus, if $S \subseteq V$, we define $G(S) = \bigcup_{v \in S} G(v)$ and $G^{-1}(S) = \bigcup_{v \in S} G^{-1}(v)$. (As a result, $G(u)$ and $G(\{u\})$ are identical expressions for all $u \in V$; and the same applies to G^{-1} .)

We can define $G^-(u)$ to denote all predecessors of vertex $u \in G$; and $G^-(S)$ to denote all predecessors of all vertices in S . In other words, $G^-(S) = \bigcup_{k>0} G^{-k}(S)$, where $G^{-k} = G^{-1} \circ G^{-(k-1)}$ for $k > 1$.

Similarly, we can define $G^+(u)$ to denote all successors of vertex $u \in G$; and $G^+(S)$ to denote all successors of all the vertices in S . In other words, $G^+(S) = \bigcup_{k>0} G^k(S)$, where $G^k = G \circ G^{(k-1)}$ for $k > 1$.

A *path* in G is a sequence of vertices $v_0, \dots, v_n \in V$ such that $(\forall k \in \mathbb{N}_n) (v_k, v_{k+1}) \in E$. Thus $G^-(S)$ is the set of vertices u that connect to one or more vertices in S through a path.

*Formally, $G(u) = \{s \in V / (u, s) \in E\}$ is the image of the singleton set $\{u\}$ in the relation represented by G while $G^{-1}(u) = \{p \in V / (p, u) \in E\}$ is the image of $\{u\}$ in the converse relation – which is represented by the graph with its edges reversed.

A directed graph (V, E) contains a *cycle* if and only if there exist a path $(v_0, \dots, v_n, v_0)$ in it. In other words the vertices $v_0, \dots, v_n, v_0$ form a directed cycle. A *directed acyclic graph* is directed graph that contains no cycles.

Let us now prove a lemma about successors and predecessors in graphs.

Lemma 1.1. *Every directed acyclic graph $G = (V, E)$ with $V \neq \emptyset$ contains at least one vertex u that has no predecessors and at least one vertex v that has no successors (that is $(\exists u, v \in V) G^{-1}(u) = G(v) = \emptyset$).*

Proof. Assume on the contrary that $(\forall u \in V) G^{-1}(u) \neq \emptyset$. We can then pick u_0 to be any vertex in the graph and inductively pick u_k to be any element of $G^{-1}(u_{k-1})$ for $k > 0$. Given that the graph has no cycles, the vertices $u_0, u_1, \dots$ must be all different, which is a contradiction because the set of vertices is finite. Similarly, if $(\forall v \in V) G(v) \neq \emptyset$ we can construct an infinite path of distinct vertices by picking v_0 to be any vertex in the graph and v_k to be any element of $G(v_{k-1})$ for $k > 0$. $\square$

If we define $I_G = \{u \in V / G^{-1}(u) = \emptyset\}$ and $O_G = \{u \in V / G(u) = \emptyset\}$ then the above lemma is equivalent to $I_G \neq \emptyset$ and $O_G \neq \emptyset$. Intuitively, the sets I_G and O_G represent the ‘input’ and ‘output’ vertices of the graph; and thus the above lemma states that every non-empty directed acyclic graph has at least one input and one output.

The following theorem allows us to partition the vertices of a directed acyclic graph into layers.

Theorem 1.1 (Graph Layering). *The set of vertices of a directed acyclic graph $G = (V, E)$ can be partitioned into subsets $L_0, \dots, L_{m-1}$, for some $m \in \mathbb{N}$, such that*

$$(\forall k \in \mathbb{N}_m)(\forall u \in V)(\forall v \in L_k) (u, v) \in E \Rightarrow u \in \bigcup_{j < k} L_j. \quad (1.1)$$

Furthermore, if the union of the subsets $\bigcup_{i < k} L_i$ is maximal for each k , then the partition L is unique and it has minimum length m .

Proof. We can construct the partition $L_0, \dots, L_{m-1}$ as follows:

$$\begin{aligned} (V_0, E_0) &= (V, E) \\ \text{for } k \geq 0 : \quad G_k &= (V_k, E_k) \\ L_k &= \{v \in V_k / G_k^{-1}(v) = \emptyset\} \\ V_{k+1} &= V_k - L_k \\ E_{k+1} &= E_k - L_k \times G_k(L_k). \end{aligned} \quad (1.2)$$

We shall first prove that all $G_k = (V_k, E_k)$ are valid graphs; that is $E_k \subseteq V_k^2$ for all k . This is obvious since we start with the graph G and, at each step, we remove all vertices that have no predecessors as well as their edges to their immediate successors so that the result is indeed a valid graph.

(A formal proof is by induction on k . G_0 is a graph by definition. For the inductive step, we assume on the contrary that $E_k \subseteq V_k^2$ and $E_{k+1} \not\subseteq V_{k+1}^2$. Given that $E_{k+1} \subseteq E_k$ and $V_{k+1} = V_k - L_k$, our assumption implies that there exist $u, v \in V_k$ such that $(u, v) \in E_{k+1}$ and $(u \notin V_k - L_k \text{ or } v \notin V_k - L_k)$. The latter implies that $u \in L_k$ or $v \in L_k$. However, if $u \in L_k$ then $(u, v) \in L_k \times G_k(L_k)$ and thus $(u, v) \notin E_{k+1}$, which is a contradiction. Alternatively, if $v \in L_k$, then $u \in G_k^{-1}(v)$, which is a contradiction since the vertices of L_k have no predecessors in G_k .)

The construction of the intermediate graphs G_k only removes vertices and edges from the original graph G_0 and thus cannot introduce cycles in the intermediate graphs. Given that G_0 is acyclic, all intermediate graphs G_k must also be acyclic. (Formally, $V_{k+1} \subseteq V_k$ and $E_{k+1} \subseteq E_k$; consequently if a cycle appears in G_{k+1} it must be present in G_k and ultimately in G_0 .)

The acyclic property allows us to apply lemma 1.1 on G_k and establish that $V_k \neq \emptyset \Rightarrow L_k \neq \emptyset \Rightarrow V_k \supset V_{k+1}$. As a consequence, if $(\forall k \in \mathbb{N}) V_k \neq \emptyset$ the infinite sequence $V_0, V_1, V_2, \dots$ satisfies $V_0 \supset V_1 \supset V_2 \supset \dots$, which is a contradiction since V_0 is finite. This implies that there exists $m > 0$ such that $V_0 \supset V_1 \supset \dots \supset V_{m-1} \supset V_m = \emptyset$. ($m = 0$ iff $V_0 = \emptyset$, that is if the original graph is empty.)

By definition, it is $L_k \subseteq V_k$ and $V_{k+1} = V_k - L_k$. Those imply that $V_k = V_{k+1} \cup L_k$, and inductively that $V = V_0 = V_m \cup \bigcup_{k < m} L_k = \bigcup_{k < m} L_k$. Furthermore, the sets $L_0, L_1, \dots, L_{m-1}$ are mutually disjoint. (Indeed, if $j < k$ then $L_k \subseteq V_k \subseteq V_{j+1} = V_j - L_j$, which implies that $L_k \cap L_j = \emptyset$.) As a result, the sets $L_0, \dots, L_{m-1}$ form a partition of V .

Furthermore, if we assume that the condition (1.1) is not true, there must exist $k < m$ such that for some $u \in V$ and $v \in L_k$ it is $(u, v) \in E$ and $u \notin \bigcup_{j < k} L_j$. The latter condition implies that $u \in V_k$ (since $V_k = V - \bigcup_{j < k} L_j$). Hence $(u, v) \in E_k$. (Indeed, if $(u, v) \notin E_k$, the edge must have been removed at a previous step $j < k$, which implies that $(u, v) \in L_j \times G_j(L_j)$ and thus $u \in L_j$, which is a contradiction.) Furthermore, $v \in L_k$ implies that $v \in V_k$. Thus $u \in G_k^{-1}(v)$, which is a contradiction because $G_k^{-1}(L_k) = \emptyset$.

This completes the first part of the theorem. For the second part, we assume $A_0, \dots, A_{a-1}$ to be a partition of G that satisfies condition (1.1). We can prove that

$$\bigcup_{i < k} A_i \subseteq \bigcup_{i < k} L_i \quad (1.3)$$

by induction on k .

Indeed, for $k = 1$, condition (1.1) implies that $A_0 \subseteq I_G = L_0$.

For the inductive step, we assume that $\bigcup_{i < k} A_i \subseteq \bigcup_{i < k} L_i$. We would like to prove that $\bigcup_{i < k+1} A_i \subseteq \bigcup_{i < k+1} L_i$. However, if the latter condition were not true, there would exist $v \in A_0 \cup \dots \cup A_k$ such that $v \notin L_0 \cup \dots \cup L_k$. This implies that $v \in A_k$ (since $\bigcup_{i < k} A_i \subseteq \bigcup_{i < k} L_i$). Thus

$$(\exists v \in A_k) v \notin L_0 \cup \dots \cup L_k. \quad (1.4)$$

As a result, it must be

$$(\forall i \leq k+1) \quad v \in V_i. \quad (1.5)$$

Moreover, $v \in A_k$, condition (1.1), and the inductive hypothesis of (1.3) imply that

$$(\forall (u, v) \in E) \quad u \in \bigcup_{i < k} A_i \subseteq \bigcup_{i < k} L_i. \quad (1.6)$$

Finally, (1.5) and (1.6) imply that $v \in I_{G_k} = L_k$, which contradicts (1.4).

We have thus shown that partition L contains subsets with maximal partial unions (as defined in (1.3), analogous to the partial sums of a sequence of numbers). If A also contains maximal unions, it must be $A_0 = I_G = L_0$ and thus (by induction on (1.3)), it must be $A_k = L_k$ for all k . Thus $A = L$.

The last step is to show that L has minimal length, too. If we assume on the contrary that $a < m$, we get $V = \bigcup_{i < a} A_i \subseteq \bigcup_{i < a} L_i \subset \bigcup_{i < m} L_i = V$, which is a contradiction. $\square$

Intuitively, at each iteration of the construction (1.2), we remove vertices with no predecessors in the current graph; and we eventually remove all vertices. As a result, when the vertex v of an edge (u, v) is finally removed, its predecessor u must have been previously removed. Therefore, if v belongs in L_k for some k , u must belong to L_j for some $j < k$.

We call the sets L_k for $k < m$ the *forward layers* or simply *layers* of graph G . We also call L the *maximal forward partition* of G . (The graph layering theorem implies that all edges to a vertex of a layer come from vertices of previous layers; there are no ‘backward’ edges that come from vertices of subsequent layers.)

In general, the partition of a directed acyclic graph into layers is not unique. For example, we can split any layer into two or more layers as long as the layer has two or more vertices. Even if we keep the number of layers minimal, the partition is not necessarily unique. For example, the graph $G = (V, E)$, where $V = \{1, 2, 3, 4\}$ and $E = \{(1, 3), (2, 4), (3, 4)\}$, admits two partitions $(\{1, 2\}, \{3\}, \{4\})$ and $(\{1\}, \{2, 3\}, \{4\})$ with minimal length $m = 3$. However, only the first one has maximal partial unions.

In general, if a vertex u has no connections to the next layer, like vertex 2 in the first partition of the above example, it can be moved to the next layer without introducing backward edges or changing the length of the partition. However, the layers cannot have maximal partial unions after such a change has taken place. Equivalently, the maximal forward partition assigns each vertex to the earliest possible layer (without introducing backward edges).

More material from graph theory will be required in later chapters but it will be introduced locally in those chapters. At this point, we have enough to define neural networks.

1.4 Defining Neural Networks

In its most elementary form, a neural network is a composition of alternating polynomial and non-polynomial functions:

$$Y_n = (F_n \circ \dots \circ F_1)(X), \quad (1.7)$$

where X is the input to the net, Y_n is its output, $F_k = f_k \circ p_k$ for $k \in \{1, \dots, n\}$, and p_k and f_k are the polynomial and non-polynomial functions, respectively. (f_n can also be the identity function so that polynomial functions can be represented as a neural network.) Those functions usually contain parameters that are randomly initialized and then ‘learned’ from data.

Equivalently, we can define the network in a recursive fashion:

$$\begin{aligned} Y_0 &= X \text{ and} \\ Y_{k+1} &= F_{k+1}(Y_k), \text{ for } k < n. \end{aligned} \quad (1.8)$$

Such a composition is often visualized as a sequence of layers, each layer representing a pair of polynomial and non-polynomial functions, as in figure 1.1. The first layer Y_0 denotes the input to the net, which is usually a vector or a multidimensional array. The last layer Y_n denotes the output, which is usually a vector or a single number. The rest of the layers are indicated as *hidden*. The *depth* of the network is defined to be the number of hidden layers plus 1, which is n for the network of figure 1.1.

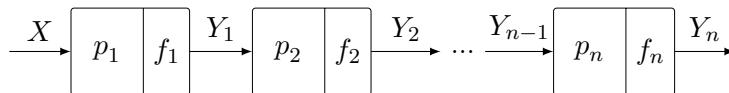


Figure 1.1: Generic definition of neural networks.

In some texts, the definition of depth includes the input and output layers. Thus it is $n + 1$ for the above network. However, we opted for counting only the hidden layers plus 1, which is equivalent to counting the polynomial or non-polynomial functions, so that when two or more networks are connected in cascade (that is one after another) the total depth is equal to the sum of the depths of the cascaded networks.

Note that definition (1.7) is too general. It implies that every function is a neural network! (Indeed, if we set $n = 1$ and pick p_1 to be the identity function, which is a polynomial, we obtain that any non-polynomial function f_1 is a neural network. Similarly, for polynomial functions, we can set $n = 1$ and pick f_n to be the identity function.)

The main issue with the definition is the complexity of its constituent functions. We need to constrain those functions. Thus we will assume that a neural network is defined over relatively small and predefined sets $\mathcal{P}$ and $\mathcal{F}$ of polynomial and non-polynomial functions, respectively. The non-polynomial functions would usually be simple and low dimensional; and the polynomial

ones would usually have a low degree and a concise form but many dimensions. We will elaborate upon this issue later in this chapter as well as in volume II.

Given that the composition of two polynomials is always a polynomial, it seems intuitive that our definition should require alternating polynomial and non-polynomial functions. However, such mathematical equivalences usually fail when the underlying domain is hard and functions are not designed but generated from data. (Computationally incomplete processes cannot guarantee that a solution will be found or that two solutions will be equivalent.)

Furthermore, composition affects the underlying parameter space. For example, consider two polynomials p and q of degree n and m , respectively. Then p has $n + 1$ parameters (its coefficients) and q has $m + 1$ parameters. However, the composition $p \circ q$ is a polynomial of degree nm but it has $n + m + 2$ instead of $nm + 1$ parameters.

Hence, we have opted to remove the constraint of alternating between polynomial and non-polynomial functions from our definition.

Equation (1.8) can induce a definition of the neural network as a directed graph if, for all applicable k and i , we consider the index (k, i) of $Y_{k,i}$ to be a unique vertex in the graph. (The vertices of the graph are indices.) More precisely, we can define

$$V = \{(k, i) \in \mathbb{N}^* / k \leq n, i \in (Y_k)_*\} \text{ and} \\ E = \{((k, i), (k + 1, j)) \in V^2 / k < n \text{ and } Y_{k+1,j} \text{ depends on } Y_{k,i}\}.$$

Then we can generate forward layers $L_0, \dots, L_n$ for the directed acyclic graph $G = (V, E)$ using the construction (1.2). As a result, it will be $L_k = (Y_k)_*$ for all $k \leq n$. We leave the proof as an exercise. (Hint: compare $[Y_{k,i}/i \in L_k]$ and $[Y_{k,i}/i \in (Y_k)_*]$.)

We shall refer to the vertices in V as *units* because they correspond to units in neural networks.* The value $Y_{k,i}$ is simply the output of the unit (k, i) . The edges are often called *connections* for similar reasons.

Note that the above graph only contains connections from each layer L_k to the next layer L_{k+1} . We call those *successive connections*. In general, we would like to allow connections from any layer L_j to layer L_k as long as $j < k$. We will refer to connections between layers L_j and L_k , where $j < k - 1$, as *cross-connections*.

Cross-connections emerge naturally in representations of functions since different parts of a formula can have different ‘depths’. For example, a network that computes $F(x) = (e^x - x)$ uses two expressions, e^x and x that have different depths (one belongs to L_0 and the other to L_1). Thus a cross-connection is used as shown in figure 1.2.

Mathematically, cross-connections do not have to be addressed explicitly because they can be ‘carried over’ by modified versions of the functions F_k

*We will eventually use the term *neurons* for those vertices but only after we have significantly increased their complexity.

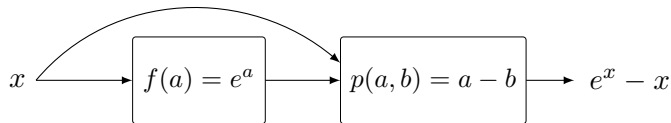


Figure 1.2: Cross-connections for expressions of varying depths.

and learned too. We call such cross-connections *implicit* and we shall examine them in more detail in volume II. In this section, we focus on the former type only.

In recent years, the term *residual connections* has also been used for the special case where the cross-connections between layers L_j and L_k cover the entire layer L_j , for some $j < k - 1$. However, we consider the term to be limiting because it focuses on specific aspects of the functionality behind such connections.

Residual connections are usually added or appended to the inputs of their target units. When added, such connections do not increase the input dimensionality of their target units and they do not introduce any new parameters (the units will only have parameters for their successive connections) but they create a specific non-locality* with a fixed identity mapping. An identity mapping may be good as a starting point but it will most likely prove suboptimal in most situations. On the other hand, when residual connections are appended to the inputs of their target units, they maximally increase dimensionality and the number of parameters. It is unlikely that either of those two extremes would be optimal in practice.

In general, cross-connections should have their own parameters; and they should have a limited (and adaptive) effect on dimensionality depending on the relative importance of the information they carry.

Cross-connections can play an additional and perhaps more important role. They can connect distinct parts of large neural systems. For example, in the human brain, white matter tracts are bundles of axons that connect different brain regions. The largest is the *corpus callosum* and it connects the right and left hemispheres.†

Given that the information that cross-connections carry can be diverse and qualitatively different, we have opted to treat them separately from successive connections and allow for special functions that aggregate such information.

Let us now try to put everything together. We can define I_u to be the set of units that provide input to unit u and I_k to set of units that provide input

*We will discuss locality in section 1.5.

†The *cerebrum* is the uppermost part of the human brain and it represents about two thirds of the total weight of the brain. It consists of two hemispheres and it contains three main types of white matter tracts. *Projection tracts* extend vertically and connect higher regions to lower ones and spinal cord centers, effectively connecting the cerebrum and the rest of the body. *Commissural tracts* connect the left and right hemispheres of the cerebrum while *association tracts* connect different regions within the same hemisphere.

to layer L_k . ($I_0 = \emptyset$ given that L_0 is a special layer that represents the input to the network and has no incoming connections.) Hence, $I_u = G^{-1}(u)$ and $I_k = G^{-1}(L_k) = \bigcup_{u \in L_k} I_u$. Of course, when $u \in L_k$ and $k > 0$, the units in $C_u = I_u - L_{k-1}$ and $S_u = I_u \cap L_{k-1}$ correspond to (explicit) cross-connections and successive connections into unit u , respectively. Similarly, the units in $C_k = I_k - L_{k-1}$ and $S_k = I_k \cap L_{k-1}$ correspond to cross-connections and successive connections into layer L_k .

We will interchangeably use the notation Y_{I_k} for the pair (Y_{S_k}, Y_{C_k}) and vice versa depending on whether we need to distinguish between successive and cross-connections or not. Thus, the output for layer L_k would be $Y_k = F_k(Y_{I_k}) = F_k(Y_{S_k}, Y_{C_k})$.

Element-wise summation or multiplication, concatenation (along an existing dimension), or stacking on a new dimension are common forms of combining the arrays Y_{S_k} and Y_{C_k} . Those are all polynomial operations and they can be represented by a polynomial layer.

However, we would like to allow for explicit functional forms in the combination of successive connections and cross-connections but also and more importantly create an entry point for additional and external aggregation – which we shall need later in this theory. This will be our first mathematical mechanism to introduce interactions among neural components.

Definition 1.1. A function $a: X^* \rightarrow X$ is an *aggregating function* iff

- a. $(\forall x \in X) \ a((x)) = x$
- b. $(\forall z \in X^*) \ a(z) = a((z_i/i \in z_* \text{ and } z_i \neq \emptyset))$

Intuitively, an aggregating function operates on a tuple that contains an arbitrary number elements from a set X . It ignores the elements that are empty tuples; it has no effect if there is at most one non-empty element; but when there are multiple non-empty elements, it maps them back to X .

If the empty tuple $\emptyset$ is not in X , the second condition in definition 1.1 has no effect. Although we do not explicitly require it in the definition, our intention is that X will be a set of tuples that include $\emptyset$. In that case, a fixed point at $\emptyset$ is also implied. ($\emptyset = a((\emptyset)) = a(()) = a(\emptyset)$.)

In other words, the minimal requirements for aggregation is that elements must always be mapped back to the original set X but empty parts should have no effect on the mapping.

Returning to our neural network, we can use an aggregating function a_k to combine the arrays Y_{S_k} and Y_{C_k} . When the layer L_k has no cross-connections, it will be $C_k = Y_{C_k} = \emptyset$ and the aggregating function will have no effect (it can be omitted altogether).

Aggregation was the last prerequisite. We can now provide our first formal definition of a neural network.

Definition 1.2. Assume that $\mathcal{F}$, $\mathcal{P}$, and $\mathcal{A}$ are sets of non-polynomial, polynomial, and aggregating functions, respectively. Furthermore assume the

identity function is contained in both sets $\mathcal{P}$ and $\mathcal{F}$ (as the only polynomial member of the latter). Then a *feed-forward neural network* over $(\mathcal{F}, \mathcal{P}, \mathcal{A})$ is a tuple (G, F, Y) that satisfies the following conditions:

- A. $G = (V, E)$ is a directed acyclic graph with a maximal forward partition $L_0, \dots, L_n$ for some $n \in \mathbb{N}$.
- B. $F = (F_1, \dots, F_n)$ and, for all $k \in \{1, \dots, n\}$, $F_k = f_k \circ p_k \circ a_k$, $f_k \in \mathcal{F}$, $p_k \in \mathcal{P}$, and $a_k \in \mathcal{A}$.
- C. Each vertex $u \in V$ represents a computing element that produces an output. $Y = (Y_0, \dots, Y_n)$ is a tuple of arrays such that Y_k stores the output of layer L_k for $k = 0, \dots, n$. Given that units are uniquely allocated to layers, we allow Y to be indexed by units directly. Thus we write, Y_u to indicate the output of unit u (instead of $Y_{k,j}$ when $u \in L_k$ and j is the index of u in L_k .) Similarly, we write Y_B to indicate the outputs of a set of units $B \subseteq V$. The dimensionality of Y_B is then implicitly derived from the units in B . (Thus Y_k is the same as Y_{L_k} .)
- D. $Y_0 = Y_{I_G}$ is the input the network and $Y_n = Y_{O_G}$ is its output.
- E. For each layer L_k , the set $I_k = G^{-1}(L_k)$ is the set of units that provide input to the layer. When $k > 0$, $S_k = I_k \cap L_{k-1}$ and $C_k = I_k - L_{k-1}$ are the sets of units that provide input to the layer through successive or cross-connections, respectively. Similarly, for each unit $u \in L_k$, $I_u = G^{-1}(u)$ is the set of units that provide input to u . When $k > 0$, $S_u = I_u \cap L_{k-1}$ and $C_u = I_u - L_{k-1}$ are the sets of units that provide input to u through successive or cross-connections, respectively. (Obviously, $I_0 = S_0 = C_0 = \emptyset$ and $(\forall u \in L_0) I_u = S_u = C_u = \emptyset$.)
- F. The computations at each layer satisfy the property

$$(\forall k \in \{1, \dots, n\}) \quad Y_k = F_k(Y_{S_k}, Y_{C_k}) = F_k(Y_{I_k}).$$

We will sometimes write (V, E, F, Y) instead of (G, F, Y) to denote the network, where $G = (V, E)$.

Intuitively, the above definition divides the network in three parts: the computing elements V , their structural interactions E , and their functionality F . Partitioning is implicitly defined by the interactions.

The latter is a tentative simplification. Neural partitioning is essential in complex systems and not merely a derivative of connectivity. Layers represent a coarse partition but it is the starting point in this theory. We will progressively refine and improve it.

Finally, we should point out that our definition creates a many-to-many relationship between graphs and functionality, allowing multiple graphs to be consistent with a given functionality and multiple functions to be consistent with the same graph. The layers are uniquely defined by the graph but they are not uniquely defined by the functionality F .

For example, let us define a network (G, F, Y) such that

$$F = ([f(x + y, x - y), f(x - y, x + y)]),$$

where $x, y \in \mathbb{R}$ are the inputs to the network and $f(a, b) = ae^b$. The network is shown in figure 1.3. The polynomial functions compute $(x + y, x - y)$ and they can be assumed to be a separate layer or part of the layer that also contains the instances of the function f . The network can thus have depth 2 or 1. In the latter case, the connections between the polynomial and the non-polynomial functions are ‘inside’ the layer and they are called *inner connections* (depicted with dotted arrows in figure 1.3).*

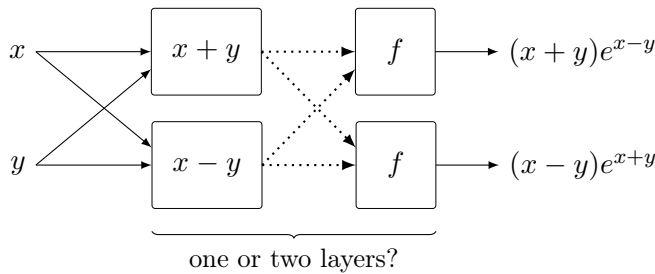


Figure 1.3: Layers with alternative binding.

The allocation of functionality to layers is a process we call *binding* and it is part of *neural partitioning*. The latter is the partitioning of a neural network into parts with specific structure and computational goals. With our current definition, binding and partitioning may seem as technicalities or conventions but they are important because they affect learning. We will examine them in more detail later in the theory.

1.4.1 A Note on Terminology

Tensors are algebraic objects that represent linear mappings.

In recent years, the Y_k arrays of the previous definition as well as the input, output, and all other intermediate results of neural networks have been called tensors. This is an unfortunate shift in terminology because it implies linearity. Linearity does not apply to the non-polynomial parts of a network (except in piecewise form) and it only applies when the polynomial transformations are linear, which is an unnecessary and suboptimal restriction. As a result, we do not use the term tensor in this theory and we discourage its use in neural networks altogether.

In general, each Y_k is an array. It can be unidimensional or multidimensional. There is no necessity or practical value for any additional terminology.

*Inner connections can affect layer connectivity. For example, consider the function $F = ([((x + y)e^x, (x + y)e^y])$. It can be implemented as an indirect fully connected network of depth 1 with no cross-connections; it can also be implemented as a partially connected network with depth 2 and cross-connections such as $F = ([x + y], [f, f])$.

The term “array” is well-understood and unambiguous, it contains no unnecessary or misleading conditions, and it has been used in a number of fields. It is used in this theory, too.

However, as we develop this theory further and create more sophisticated neural architectures, we will realize that the layers (or any other neural partitions that apply) are not actually arrays but tuples with varying and adaptable shapes and sizes across each dimension. Arrays require that all elements across each dimension have the same shape and size. Although this constraint is a reasonable simplification for initial approaches, it is ultimately untenable for neural networks.

On a broader level, we also contest the term “artificial intelligence” as a buzzword, an anthropocentric oxymoron, and the outcome of confusion. The term was coined in the 1950s by J. McCarthy with little understanding of the mathematical foundations of intelligence and what intelligence is.

We will discuss intelligence in more detail in volume III. At this point, we can briefly say that a learning system either does have the capacity for a given level of intelligence or it does not. If it does, then it has the ability to build itself up from a seed state, given a proper environment and sufficient resources, until it reaches a state that admits measurements of intelligence at the given level. If it does not have such capacity, it can be artificial or not, but it cannot be artificial and intelligent at the same time.

In other words, intelligence is a property of growth, not human design. In the end of our venture with neural networks, we will be the gardeners of their intelligence but not their makers.

Yet, the term artificial intelligence seems to have captured the imagination of the public and attracted the interest of corporations and governments as well as venture capitalists. Despite the fact that the field of artificial intelligence was primarily hype composed of vague or naive abstractions and proved to be a broad financial and research failure in the 1970s and 1980s, the term reemerged after it was adopted by the neural network community.*

In the end, buzzwords and oxymora can only bring damage and disrepute to a field. We can hope that the neural network community will be wiser but we understand that fallacies can persist. In the words of Max Planck: “A scientific truth does not triumph by convincing its opponents and making them see the light, but rather because its opponents eventually die and a new generation grows up that is familiar with it.”

1.4.2 The Polynomial Functions

Polynomial functions appear in all or most layers of a neural network. They are commonly used to create mappings across multiple dimensions and they often affect dimensionality. They usually contain the largest number of parameters within each layer.

*See *Neural Age* [92, chapter 3] for a more complete history.

In their most elementary form, they are affine transformations

$$p(x) = Wx + b,$$

where W and b are called the *weights* and the *bias* of the transformation and the associated layer.

A *fully connected layer* is a layer L_k such that $k > 0$ and $L_{k-1} \times L_k \subseteq E$. In other words, L_k is fully connected iff it is not the input layer and every unit $u \in L_{k-1}$ connects to every unit $v \in L_k$. Note that this does not necessarily imply that the polynomial component p_k of the layer is fully connected when $F_k = f_k \circ p_k$ and elements of f_k can combine multiple elements of p_k . (We shall see examples in volume II.) Thus we will distinguish the case of *indirect full connectivity* as described above and *direct full connectivity* where each element of the intermediate polynomial result $p_k(x)$ of a layer L_k depends on all elements of the input x to the layer.

Direct full-connectivity is contraindicated by this theory. It is virtually impossible in large biological nervous systems. Even if it were possible it would be an extraordinary waste of energy and resources and it would only amount to an evolutionary disadvantage.

A more general concept is the *sparsity* of a layer, which is the fraction of the number of incoming connections into the layer versus the total number of all possible connections from the input to the layer. When the polynomial component p is affine, the sparsity of the layer is the sparsity of the array W .

Large neural systems are dramatically sparse. For example, the human brain contains approximately 86 billion neurons, layers can easily contain a hundred million neurons or more, but neurons only have about ten thousand connections on average.*

Structure and sparsity in the polynomial components can result in reduced or minimal connectivity and generate effective neural forms. It is one of the most important factors of any neural architecture. We will return to it when we discuss receptive fields and when we examine polynomial transformations in greater detail.

More sophisticated and novel forms for polynomial transformations will be presented in volume II.

1.4.3 Activation Functions

Multivariate functions can account for interactions of their input variables or not. The latter class contains a special set of functions that have the same number of inputs as outputs and their i -th output depends only on the i -th input. Those functions are called *element-wise*.

Formally, for any given number $n \in \mathbb{N} - \{0\}$, tuple $d \in \mathbb{N}^n$, and sets A and B , a function f from A^d to B^d is element-wise iff there exist functions

*Perkinje cells are among the largest with up to 200,000 parallel fibers.

$f_i : A \rightarrow B$, for $i \in \mathbb{N}_d$, such that

$$(\forall x \in A^d)(\forall i \in \mathbb{N}_d) f(x)_i = f_i(x_i).$$

The non-polynomial functions in neural networks are primarily element-wise functions generated by applications of unidimensional functions. Such unidimensional functions are called *activation functions* or *unit activation functions* because they are individually associated with each unit in the network. The same activation function is often repeated across multiple units and layers and often across the entire network.

Monotonicity and continuous differentiation are desirable properties of activation functions but a limited number of local minima or discontinuities are not uncommon and they have representational and learning advantages (and disadvantages, too). Table 1.1 shows some commonly used unidimensional and multidimensional activation functions.

Sigmoids are bounded and monotonic real functions with a center of symmetry. The latter implies that their derivative is symmetric around the vertical line that passes through their center of symmetry,* as shown in figure 1.4.

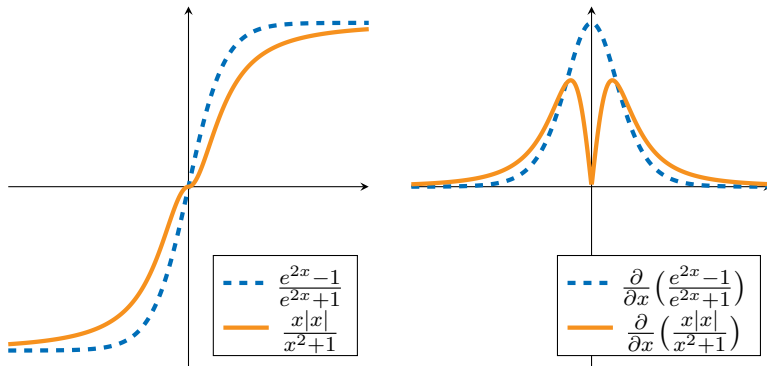


Figure 1.4: Examples of sigmoids and their derivatives.

Note that the derivative of a sigmoid can be a bell-shaped function (since it is symmetric around a vertical line) but this is not always the case. A counterexample is also shown in figure 1.4. If we wish to make sigmoids look like the letter ‘S’, as their name implies, the derivative must not change monotonicity on either side of the center of symmetry and thus the sigmoid should be a convex function on one side and a concave function on the other. However, we have not added such a condition in the definition, allowing more functions, like the second function in figure 1.4, to be sigmoids.

Monotonicity is important as a constraint for the overall shape of sigmoids. If we remove the constraint, various non-monotonic functions and even periodic

*A real function f is symmetric around a point $p \in \mathbb{R}$ iff $\forall x \in \mathbb{R}$ it is $f(p+x) - f(p) = f(p) - f(p-x)$, which is equivalent to $\frac{df}{dx}(p+x) = \frac{df}{dx}(p-x)$, assuming that the derivative $\frac{df}{dx}$ exists.

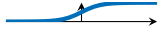
Sigmoids			Multidimensional
Step	$\begin{cases} 1, & x \geq 0 \\ 0, & \text{else} \end{cases}$		Softmin
Logistic	$\frac{1}{1+e^{-x}}$		$e^{-x_i} / \sum_k e^{-x_k}$
Relu6	$\begin{cases} \min(x, 6), & x \geq 0 \\ 0, & \text{else} \end{cases}$		Softmax
Tanh	$\frac{e^{2x}-1}{e^{2x}+1}$		
Arctan	$\tan^{-1}(x)$		
Softsign	$\frac{x}{1+ x }$		
HardTanh	$\begin{cases} x, & x \leq 1 \\ x/ x , & \text{else} \end{cases}$		$e^{x_i} / \sum_k e^{x_k}$
Partly Linear			
Relu	$\begin{cases} x, & x \geq 0 \\ 0, & \text{else} \end{cases}$		
Elu	$\begin{cases} x, & x \geq 0 \\ \alpha(e^x - 1), & \text{else} \end{cases}$		
Leaky Relu	$\begin{cases} x, & x \geq 0 \\ \alpha x, & \text{else} \end{cases}$		
Prelu	$\begin{cases} x, & x \geq 0 \\ wx, & \text{else} \end{cases}$ (w is learned)		
Selu	$\begin{cases} \lambda x, & x \geq 0 \\ \lambda \alpha(e^x - 1), & \text{else} \end{cases}$ ($\alpha \simeq 1.6733, \lambda \simeq 1.0507$)		
Celu	$\begin{cases} x, & x \geq 0 \\ \alpha(e^{x/\alpha} - 1), & \text{else} \end{cases}$		
Hardshrink	$\begin{cases} x, & x > \lambda \\ 0, & \text{else} \end{cases}$		
Swish	$\frac{x}{1+e^{-bx}}$		

Table 1.1: Typical activation functions.

functions like the sinusoid would be sigmoids. Given that sigmoids are also bounded, their monotonicity implies that their derivative converges to zero at infinity. In other words, the derivative of sigmoids is zero or near zero everywhere except in a relatively small interval around their center of symmetry.

On the other hand, partly linear units have one or more linear or affine segments that are non-constant and usually span half or more of the real line. Thus they have a constant non-zero derivative in a large part of the real line.

The linearity of such functions is often diagonal and thus the derivative is equal to 1 in those segments. The swish function is an approximate case in the sense that the segments are approximately linear and the derivative is asymptotically constant. A non-trivial derivative for a large part of the real line is a crucial difference between such functions and sigmoids for optimization purposes.

1.4.4 A Note on Unidimensionality

Unidimensionality and element-wise functions have primarily been motivated by simplicity and related observations in biological neural systems as well as theoretical results such as the *Kolmogorov superposition theorem*:

Theorem 1.2. *Assume $\mathbb{R}$ to be the set of real numbers and n to be a positive integer. Any multivariate continuous function $f : [0, 1]^n \rightarrow \mathbb{R}$ can be written in the form*

$$f(x_1, \dots, x_n) = \sum_{q=0}^{2n} \Phi_q \left(\sum_{p=1}^n \Psi_{q,p}(x_p) \right)$$

with continuous one-dimensional inner and outer real functions Φ_q and $\Psi_{q,p}$. The inner functions $\Psi_{q,p}$ are independent of the function f .

In the 1960s, the theorem was extended by G. Lorenz and then by D. A. Sprecher so that it eventually employed only a single Φ function and a single Ψ function but with affine transformations in the inner sum.

Those results have been construed into an argument that the only true multivariate function is the summation and that every other function can be composed by univariate functions and summations. Thus they seem to imply that the polynomial functions of neural networks should be affine and that the non-polynomial ones should be element-wise and unidimensional on each unit.

However, we understand the first argument to be more qualitative than precise or mathematical. For example, and given that any summation $\sum_i x_i$ can be written as $\ln(\prod_i e^{x_i})$, products are also ‘true’ multivariate functions; and likewise for polynomials, which are combinations of products and summations.

More importantly, we consider the above evidence to be insufficient if not misleading about its potential consequences to neural networks. We will present contrasting biological evidence and mathematical arguments – that derive from the limitations of learning – against element-wise unidimensionality as well as amendments of the functions later in volumes II and III. At the time of this writing, a partial departure from unidimensionality seems justifiable.

1.4.5 Input Types

In general, the input to a neural network can be any numeric representation. In real applications, the input usually takes various forms.

For example, when the domain is images, a common representation is to use the red-green-blue (RGB) values of each pixel. Given that each pixel also has x, y coordinates in the image, the image is represented as a five dimensional array (using three dimensions for color and two dimensions for position). When the domain is video, the input can be a sequence of images.

Another common example is text. Each word or character is usually represented as a vector. A sentence or a paragraph is then a sequence of such vectors. Neural networks often operate on a fixed window that is shifted across the input sequence and they either generate an output sequence from the input sequence or an aggregate representation of the input sequence. The former is used in applications such as machine translation while the latter is commonly used in text classification and other natural language processing tasks.

Representations based on the Fourier transform of the input signal are sometimes used. For example spectrograms and cepstral coefficients are used in speech recognition.

The purpose of this discussion is not to cover all common representations but rather to demonstrate that the input types to a neural network can be diverse and multidimensional. The inherent dimensionality of the input, and especially spatial or temporal dimensions, often affect the dimensionality of the hidden layers and the structure of the network.

1.5 Locality

In physics, the *principle of locality* states that an object is directly influenced only by its immediate surroundings; and a theory that is consistent with that principle is a *local theory*.

With a measure of proximity or association, we can extend the above definitions to abstract mathematical models and their elements (that do not necessarily correspond to physical entities or the physical reality). Given that any relation among such elements can produce an association, locality effectively translates to the property that the elements of mathematical models tend to be directly affected by a small subset of all the elements, which thus constitute a locality.

The above definition is different from the homonymous principle in computer science, a.k.a. *locality of reference*, which states that a processor tends to access data in proximate storage locations (*spatial locality*) and repetitively over relatively short periods of time (*temporal locality*). Locality of reference is probabilistic, and thus non-deterministic, because the associated access patterns may occur most of the time but not necessarily all the time. *Sequential locality* is a special case of deterministic spatial locality where data are arranged and accessed linearly (e.g. in an array).

Locality is also present in universal computing models such as *Turing*

machines.^{*} It emerges from the locality of the transition function δ (or transition relation Δ for nondeterministic machines) which usually associates each state p with a small subset of the set of all the possible states K . It also emerges from the fact that each transition in a Turing machine depends on the current *scanned symbol* and not the rest of the tape. As A. Turing stated in 1948:

“ At any moment there is one symbol in the machine; it is called the scanned symbol. The machine can alter the scanned symbol, and its behavior is in part determined by that symbol, but the symbols on the tape elsewhere do not affect the behavior of the machine. ”

Although this kind of locality will tend to increase locality of reference it is primarily an instance of the principle of locality in mathematical models.

In neural networks, sequential locality of reference is most common. Each unit, as a computing element, accesses only its own parameters and often in a linear fashion. There is no uncertainty about which locations or parameters will be accessed next as in typical programs for von Neumann architectures.[†]

Such deterministic locality of reference has important implications for the implementation of neural architectures and related hardware solutions. For example, linear access affects the efficiency and viability of common hardware elements, such as caches, RAM, or any off-chip memory as well as processing dichotomies such as between central processing units (CPUs) and graphics processing units (GPUs) or between those processing units and memory.

However, from a mathematical viewpoint, it is rather the principle of locality, as defined in physics, that matters most for neural networks. It is a special case of a more foundational concept (coherence) that will be introduced in chapter 5. Locality of reference is only enhanced by it. Our theory and our models will increasingly become more local and more dependent on coherence.

^{*}A *deterministic Turing machine* consists of an infinite tape that can store symbols, a head that can read or write symbols on the tape, and a transition function that specifies the actions of the machine.

More formally, a deterministic Turing machine is a tuple of the form (K, Σ, δ) , where K is a finite set of states that contains the *initial state* s and a *final state* f , Σ is a finite set of symbols (that includes the blank symbol $\sqcup$), and $\delta: K \times \Sigma \rightarrow K \times \Sigma \times \{\leftarrow, \rightarrow\}$ is the transition function so that, if $\delta(p, \sigma) = (q, \tau, d)$ and the machine is currently at state p with scanned symbol σ , then the machine would move to state q , replace the current scanned symbol with τ , and move the head either left (if $d = \leftarrow$) or right (if $d = \rightarrow$).

[†]Von Neumann architectures separate processing units from memory, thus creating a material and computational dichotomy, and allow the processing units to access memory in arbitrary ways. This introduces uncertainty in the memory access patterns. Intractability of the underlying computation, when the processing units are equivalent to Turing machines, implies that the uncertainty cannot be resolved except with statistical methods that are not optimal and only capture a fraction of the patterns and their locality.

1.5.1 Receptive Fields

The *receptive field* $R_N(v)$ of a unit v in a neural network $N = (G, F, Y)$ is the set of the network's inputs that affect Y_v . Assuming that $G = (V, E)$, it can inductively be defined as follows:

$$R_N(v) = \left\{ \begin{array}{ll} \{v\} & \text{if } v \in I_G \\ \bigcup_{u \in G^{-1}(v)} R_N(u) & \text{otherwise} \end{array} \right\}. \quad (1.9)$$

In other words, the receptive field of a unit v is the set of input units of N that connect to v through a path in G . Thus $R_N(v) = G^-(v) \cap I_G$. However, the inductive definition demonstrates more clearly that receptive fields tend to increase for units that are further from the input.

We can extend the definition to sets of units. Thus the receptive field $R_N(S)$ of a set of units $S \subseteq V$ is the set of the network's inputs that affect Y_S . Mathematically, $R_N(S) = \bigcup_{u \in S} R_N(u)$.

Receptive fields are important in neuroscience and in studies of neural systems because they can be used to locate specific neurons or regions that respond to certain stimuli.

We can generalize the definition to target input from a specific layer. Thus if L_k is the layer k of the above network, we can define the receptive field $R_N(v, k)$ to be the set of units of L_k that affect Y_v . Thus

$$R_N(v, k) = G^-(v) \cap L_k = \left\{ \begin{array}{ll} \emptyset & \text{if } v \in L_j, j < k \\ \{v\} & \text{if } v \in L_k \\ \bigcup_{u \in G^{-1}(v)} R_N(u, k) & \text{otherwise} \end{array} \right\}. \quad (1.10)$$

In other words, we define receptive fields to be relative to a target layer; and when that layer is L_0 , the target is the input to the network. (Obviously, $R_N(v) = R_N(v, 0)$.)

An important measure of locality in the network N is the average portion of L_{k-1} that is the receptive field of units in L_k . Thus, assuming that $|S|$ indicates the size (number of elements) of a finite set S and that A/B indicates $|A|/|B|$ for finite sets A and B , the *locality ratio* of layer $k \in \{1, \dots, n\}$ is defined as

$$l_k = E_{v \in L_k} \left(\frac{R_N(v, k-1)}{L_{k-1}} \right) = \sum_{v \in L_k} \frac{R_N(v, k-1)}{L_{k-1} \times L_k}. \quad (1.11)$$

(Locality for L_0 is, of course, meaningless in this context.)

In general, units in higher layers that represent complex patterns and functionality of the input usually depend on a larger number of inputs. Instead of making the inputs connect directly to those units or their immediate or near predecessors, we can create the necessary dependencies using multiple

highly local layers (i.e. layers with small or trivial locality ratio). Receptive fields can grow exponentially with the number of layers. Thus even minimal connectivity can quickly grow the receptive fields to cover the entire input.

There is often overlap between receptive fields of ‘successive’ units and it reduces the growth of the fields with depth. The worst case is linear growth when the input to successive neurons only changes by at most one element in each dimension of their input array. When the change is greater than one in one or more dimensions, the growth is exponential. A combination of linear and exponential growth is often used in highly local networks.

Locality can be perceived as a divide-and-conquer strategy where a complex problem is divided into multiple and easier (and local) sub-problems. More complex forms and solutions can then be built from simpler ones. Of course, this is a simplification of the role of locality but it can help us understand the types of interactions that can emerge from it.

Biological evidence in support of locality is abundant. For example, the vertebrate retina has 10 distinct layers and they are primarily local. Simple and complex cells with relatively small and local receptive fields have been discovered in the visual cortex. Although the classification is not precise, it is based on the complexity of the patterns that can be detected by the cells in their receptive fields. Thalamocortical models assert the creation of both simple and complex cells from thalamocortical pathways. The models are hierarchical or parallel and they postulate the synthesis of the cells – from simple to more complex – in a hierarchical or parallel fashion, respectively.

The hierarchical model is more consistent with the gradual creation of complexity over multiple local layers and the parallel with the representation capabilities of shallow networks of depth 1 or 2.

Recurrent and state-dependent models have also been proposed. They focus more on networks of cortical connections instead of single cells and they are supported by the structure and the connectivity of the cortical network and in particular by the predominance of intracortical over thalamocortical connections.

Overall, locality does not necessarily imply the hierarchical or the parallel model. In its stateless and unit-oriented form and in combination with theoretical representation results – such as Kolmogorov’s superposition theorem – and the computational limitations of learning, it may suggest a mixture of sequential and parallel composition and thus a combination of the two models. This can also explain supportive evidence for both models. The recurrent version shifts the attention altogether and away from single units.

We will further examine locality in chapter 5 and consider recurrent models of locality in volume III.

1.5.2 Neural Locality Types

Neural networks can be classified in three broad categories based on the connectivity of their units.

Fully connected neural networks contain only fully connected layers; thus $l_k = 1$ for all $k > 0$. *Local neural networks* contain one or more locally connected layer, or equivalently, there is at least one k such that $l_k < 1$. Those two categories are mutually exclusive.

The third category is independent of locality and derives from cyclicity in the connections: *recurrent neural networks* contain at least one connection from a unit in a layer to a unit in the same or a previous layer.* This implies that their graph contains one or more cycles. Consequently, neither construction (1.2) nor definition 1.2 can apply to them. In such networks, neural partitioning cannot be derived from connectivity only.

Although recurrence and its substitutes are crucial in the development of complex neural systems and universal computing paradigms, we will defer their treatment until volumes II and III.

Fully connected networks are impractical and wasteful. They are invariably a suboptimal architecture and a poor choice. They should only be used as a baseline for comparisons with local networks, if no other networks are available, and they should never be used to compare neural networks with other paradigms.

The same largely applies to fully connected layers, too. Potential exceptions are in the last one or two layers of deep neural networks when those layers are relatively small. Another exception can be found in (some of) the layers of *encoder-decoder networks*, which are networks that learn the identity function but compress their input to a smaller array in their hidden layer(s) before they attempt to decompress it back to the original, thus learning a compression and decompression mapping. However, even in those cases, a local layer would usually perform better when its connectivity is learned through plasticity or even simple pruning.

Drawing an analogy with programming, we can say that locality is effectively iteration. In a local layer, the polynomial function p_k is not a function whole that operates on all its inputs at the same time but rather it consists of smaller parts that operate on smaller subsets of the input the same way that a programming loop operates on the element i of an array, or few elements around it, at its i -th iteration. From an architectural or learning viewpoint, it is not the computational details at each iteration that matter most but rather the fact that we use the iteration mechanism and thus locality.

In local layers, each unit usually connects to a relatively small number of other units and each connection has its own parameter(s). Although this seems to be the case in biological neural networks, it is not a common current practice in neural networks. Instead, connection parameters are usually shared across multiple local connections and locality is used in spatial or temporal dimensions but not across functionality. We consider this configuration to be suboptimal for learning.

Let us examine the issue in more detail.

*Formally, there exist layers L_j and L_k such that $j \leq k$ and $I_{L_j} \cap L_k \neq \emptyset$.

1.5.3 The Pale Fire of Convolutional Neural Networks

Cross-connections do not require special treatment in this section. They only add an additional source of input and they are subject to similar locality constraints on the local networks we consider. Thus, and in order to simplify the discussion, we will assume that the network contains no cross-connections.

Now let $v \in L_{k+1}$ be a local unit that operates on a proper subset of $I_{L_{k+1}}$. Given that the network has no cross-connections, $I_{L_{k+1}} \subseteq L_k$ and $I_v \subset L_k$. We will furthermore assume that $F_{k+1} = f_{k+1} \circ p_{k+1}$, p_{k+1} is affine, and f_{k+1} is element-wise with unidimensional elements. As a result, the output of the unit v is $Y_v = f_v(p_v(Y_{I_v}))$, where I_v is a neighborhood around v and p_v and f_v are the respective elements of p_{k+1} and f_{k+1} that correspond to v . Given that p_{k+1} is affine, the above can be written as

$$Y_v = f_v(W_v Y_{I_v} + b_v) \quad (1.12)$$

for some parameter arrays W_v and b_v of the unit v .

We can assume that Y_k has m_k dimensions and size $d_{k,i} > 0$ along dimension i . In other words, $(Y_k)_* = \mathbb{N}_{d_k} = \mathbb{N}_{d_{k,0}, \dots, d_{k,m_k-1}}$. Similarly we can assume that $(Y_v)_* = \mathbb{N}_{d_v} = \mathbb{N}_{d_{v,0}, \dots, d_{v,m_k-1}}$.

Given that v is local, one would expect that $d_{v,i} < d_{k,i}$ (and significantly so) for all i . Furthermore, if u and v are two distinct units, one would expect that their parameters will be distinct, too; that is W_u and b_u to be distinct from W_v and b_v . Both of those conditions are violated by a rather popular type of local networks that are called *convolutional neural networks* (CNN).

In each local layer L_{k+1} of those networks, one among the m_k dimensions is reserved and is treated in a fully connected fashion. We can examine the networks in the context of computer vision so that we can better understand the conventions and limitations of their design.

The input to such networks are images. Each image is a two dimensional array of pixels and each pixel has three values that represent the red, green, and blue components of its color. Thus an image with width w_0 and height h_0 pixels can be represented by an array $X \in \mathbb{R}^{3, h_0, w_0}$. (Or any permutation of w_0 , h_0 , and 3.) Intuitively, the last two indexes of X specify the location of the pixel in the image and the first specifies a color component.

X is the input Y_0 and it is transformed by a CNN to an array $Y_k \in \mathbb{R}^{\phi_k, h_k, w_k}$ at each local layer L_k , for $k > 0$. Those are called *convolution* or *convolutional layers*. They perform a 2D cross-correlation of their input with their parameters. This operation can also be defined as a convolution if we invert the indices of the weight arrays.* Thus the name of the layers. The last two dimensions in the Y_k arrays are spatial and they correspond to regions of the input image. The first dimension is functional and it represents features extracted from the corresponding region of the image. The arrays $Y_{k,\beta}$ for $\beta < \phi_k$ are two dimensional arrays and they are called *planes* or *filters*. Each

*We will examine cross-correlation and convolution and their differences in chapter 6.

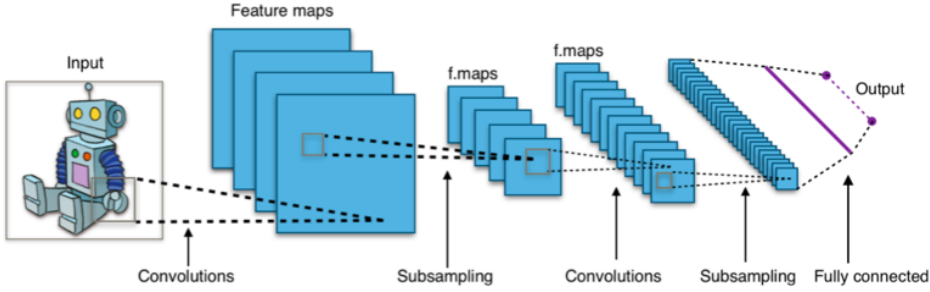


Figure 1.5: A convolutional neural network for vision.

is a representation of the input image, or rather a feature mapped across the image.

Formally, each unit $u \in L_k$ can be represented as a tuple $u = (k, \beta, y, x) \in \mathbb{N}_{n+1} \times \mathbb{N}_{\phi_k} \times \mathbb{N}_{h_k} \times \mathbb{N}_{w_k}$, where $n+1$ is the number of layers in the net. For $k=0$, it is $\phi_k=3$, of course.

The correlation (or convolution) operation that is applied to layer L_k is assumed to have width c_k and height r_k (with $c_k < w_k$ and $r_k < h_k$ when L_k is spatially local). Those specify the *kernel size* of the operation. The weight of a unit $v = (k+1, \gamma, y, x)$ is an array in $\mathbb{R}^{\phi_{k+1}, c_k, r_k}$ and its bias is a real number. Thus the weight W_{k+1} and the bias b_{k+1} for the entire p_k transformation are arrays in $\mathbb{R}^{\phi_{k+1}, \phi_k, c_k, r_k}$ and $\mathbb{R}^{\phi_{k+1}}$, respectively. If we assume that $\star$ denotes cross-correlation, the exact formula of the output Y_v of v is then

$$\begin{aligned} (Y_{k+1})_{\gamma, y, x} &= f_v \left((b_{k+1})_{\gamma} + \sum_{\beta < \phi_k} ((W_{k+1})_{\gamma, \beta} \star (Y_k)_{\beta})_{y, x} \right) \\ &= f_v \left((b_{k+1})_{\gamma} + \sum_{\beta < \phi_k} \sum_{j < r_k} \sum_{i < c_k} (W_{k+1})_{\gamma, \beta, j, i} (Y_k)_{\beta, y+j, x+i} \right). \end{aligned} \quad (1.13)$$

As it can be seen from the above equation, the network is local in its spatial dimension x and y but it is not local across its filters. Each filter γ of Y_{k+1} depends on all filters β of Y_k . This situation is depicted in fig 1.5. Each stack of filters corresponds to a layer and all filters in a stack depend on all filters of the previous stack. Such full connectivity is often wasteful and detrimental. It is a constraint that can adversely affect the ability of the network to scale or learn and generalize.

Empirical evidence has led to a modified version of the operation to partition the ϕ_k filters of L_k in g_k equal groups (where g_k must divide ϕ_k) and then have full-connectivity within each group only. This operation is called *grouped convolution*. However, partitioning, and especially partitioning in equal parts, does not solve the problem. It is an improvement but not a solution.

Intuitively, filters are feature maps over the input image. Successive filters

build upon previous ones to create more complex features. For example lower lever filters may detect simple features such as edges of various orientations. Filters on subsequent layers can combine the previous features to create more complex curves. Then higher layers can create features that detect crests, beaks, gills, or fins in images of animals. And top layers may create features that detect birds or fish in images. It is highly unlikely that *exactly* the same lower level features will be used in all the higher level features. There will be a distribution of differences and omissions that render full-connectivity an excess; and there will be features in common that make partitioning (and especially partitioning in equal parts) a source of unnecessary redundancy and unnecessary constraints. All of those factors can only affect learning in an adverse way.

In general, we would like to extend locality to the filters as well. Locality is not an exclusive property of spatial or temporal dimensions. It is a fundamental property of all computing. It should appear in every dimension. The only potential difference across dimensions is their optimal level of locality.

Furthermore, it is trivial to do add locality to the filters. For example, in equation (1.13), we can limit the first summation so that such $\gamma \leq \beta \leq \gamma + \epsilon_k$, where ϵ_k is the kernel size of the correlation over filters. This is a simple and probably suboptimal form of locality.* Yet, it allows the number of filters to scale up without quadratic growth of the underlying parameters, provides for specific features to be included in different parts of higher layers, and permits common features to be employed across those parts. The locality in the filters will furthermore tend to cluster units that compute related features near each other – which is a desirable property of the network as we shall see in volume II of this theory.

The advantages of locality are obvious and significant. This leads to a question. Why were CNNs not defined to have local filters in the first place and why, after decades of usage, do they still remain fully connected in most cases?

The case of CNNs demonstrates the perils of pure empiricism. When fields are driven exclusively by empirical results without a guiding theory, mistakes and misconceptions are practically inevitable and common.

Although CNNs were inspired by the seminal work of D. H. Hubel and T. N. Wiesel on receptive fields in 1962, they are primarily a misinterpretation. The importance of such networks is not in sharing parameters or in the correlation or convolution or any such supported operation but rather in the fact that they implement (spatial) locality. Their *pale fire* is a reflection of the ‘light’ of locality.† It is the true source of their significant advantages over fully connected or largely non-local networks.

Yet, and despite the many decades since Hubel and Wiesel’s work, this

*We will examine more sophisticated forms in chapter 8 and in volume II as well.

†See the last paragraph in section 1.6 for an explanation of the origin and meaning of the term “pale fire”.

fact has not been sufficiently emphasized. Perhaps the realization never took place. The focus has been on alternative factors such as the operation (convolution or correlation), weight sharing, or associated properties such as translation invariance. In reality, locality overrides all other factors. Without a theory that can identify or postulate important properties and components of a system or architecture, empiricism can indeed lead to oversights.

Now, let us consider the second issue with CNNs.

CNNs share weights across spatial dimensions. Each filter applies the same parameters (weights and biases) to the entire input. It has been argued that this has three advantages.

Given that parameters are learned from data and all units in a filter update the same parameters, shared parameters can be learned faster, with fewer data, and with less risk to overfit the data.

Second, sharing the parameters of a filter is equivalent to scanning the image with the filter and reporting its value at each region that has the shape and the size of the filter. Obviously, this provides translation invariance. If a pattern is detected by the filter, it will be detected irrespectively of its position in the image. Hence, a filter that can detect an apple, it will detect it anywhere in the image.

Third, shared parameters make the network smaller, requiring less memory to be stored and possibly less energy to be maintained. (The later is questionable since shared parameters must be communicated and communication has a cost, but the memory is indeed less.)

Let us now contest those ‘advantages’ and demonstrate that they are actually disadvantages.

Sharing filter parameters is a violation of locality. The shared parameters cannot be locally associated with any region of the visual field. Instead, they are associated with the entire field and they are learned rather indiscriminantly from it. This has many adverse implications.

For one, it makes *plasticity*, which is the ability of the network to grow or prune local connections, unnecessarily hard. Plasticity is considerably more important than convolution. Increasing its complexity in order to support convolution (or any such operation) is simply poor design.

Sharing also makes implementation harder. In the end, we will have to implement the mathematical models behind neural networks on some computing platform. If we implement the units as individual computing elements in hardware, like neurons in biological systems, which is a better approach than current array-based approaches, then the units will have an additional communication overhead to propagate updates of the parameters.

Translation invariance is a desirable property but it is not the only one. Rotation, scale, shearing or color invariance are equally important. For example, a car should be detected in an image irrespectively of its size, color, or orientation. We cannot possibly try to hard-code all those invariants in the architecture and the structure of the computation. Even if we succeed some-

how, the result would be overly complex, inelegant, and hard to understand and evolve. In general, a learning architecture should not explicitly provide for input invariants but rather let constituent learning elements, such as the filters in this case, acquire the invariants as needed.

Even if we make the generous assumption that achieving translation invariance through the architecture is somehow a good idea, the approach creates more problems than it solves. It requires that local statistics be stationary across the input. This is obviously not the case in reality. Different parts of an image have different statistics – with predictable differences if the input data can be aligned. This was first realized in face recognition applications that used aligned images of faces. Non-stationarity of local statistics led to local networks instead of CNNs. In general, stationarity is a significant limitation.

Furthermore, translation invariance through convolution requires that the resolution across the image be constant. If it is not, only resolution-invariant filters will also be translation invariant. The latter is a severe limitation.

Consequently, CNNs can effectively process images only under stationarity and with a fixed resolution across the image. Fixed resolution is currently the standard in computer vision but it represents an oversimplification.

Fixed resolution is an unfortunate idea, too. Most of the time, the center of an image contains more useful information than the corners. When organisms process visual information, they invariably have a center of attention, which is the point of focus of their eyes, and they use much larger resolution around that center.

There are two types of primary photoreceptor cells in the retina: *cones* and *rods*. Cones are responsible for the perception of colour and high precision vision that is used in tasks such as reading in well-lit conditions. Rods provide black-and-white vision and function mainly in dim light. As shown in figure 1.6, the distribution of cones and rods in the retina is clearly non-uniform.

The concentration of cones near the focus of attention is more than an order of magnitude greater than the distribution of cones away from the center. Rods have a less extreme distribution but they are also non-uniform.

In short, biological vision systems do not process images uniformly. They use a higher resolution at the center or the point of focus for tasks that require high visual precision. This allows such systems to detect fine details in regions where it matters most without wasting neural resources to detect similar details or confirm their absence in regions where they do not matter as much. Consequently, neural networks that are architecturally restricted to operate on fixed resolution have an evolutionary disadvantage. If they ever appear on earth, they would most likely be driven to extinction by natural selection.

The third argument that a CNN is actually smaller is invalidated by the same reasoning. If a CNN must support variable resolution or non-stationarity, it will probably be as large or larger than a local network with

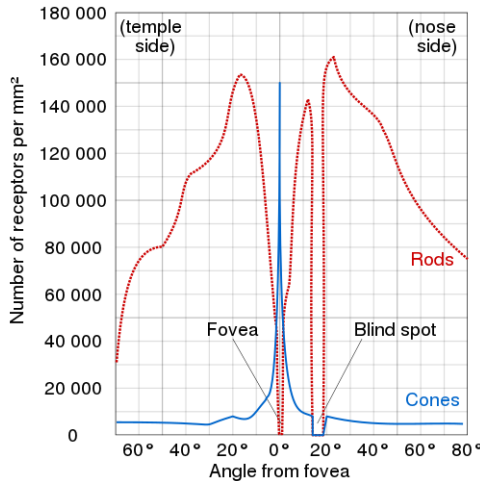


Figure 1.6: Distribution of cones and rods along a line passing through the fovea and the blind spot of a human eye.

no shared weights. In other words, CNNs can be smaller only if we simplify the underlying problem to admit a largely stationary representation under constant resolution. For most applications, this is merely oversimplification.

In the author’s opinion, one of the least appealing features of CNNs is their complexity and inelegance. For example, we can compare the rather complex equation (1.13) with equation (1.12). In the latter case, we can furthermore define I_v to be any neighborhood around unit v and let its shape and size be learned through growth and plasticity. Non-stationarity and variable resolution are no longer an issue. The formulas remain as obvious and simple. In many ways, convolutional neural networks are unnecessarily convoluted (pun intended).

In the end, initial conventions and fallacies can only expire. Local networks are slowly emerging in the field. More adaptive local shapes have also appeared, albeit in simple forms such as *deformable convolutions*. Filter locality finally emerged as a consideration in 2018 (but it has been part of this theory for nearly two decades). It is only a matter of time before the emphasis shifts to essential properties and components irrespectively of our theory.

On the other hand, CNNs have played an important role in the field. They were more useful in previous decades when computing was more limited and only small data sets were available. They largely replaced full connectivity, which is a constraining and unsound architectural element. Their parameter sharing is not necessarily a bad idea when data sets are small or diminished and the visual field is treated uniformly.

In that regard, convolutional neural networks were not necessarily a misstep but rather an initial step in an evolutionary process that leads to locality, non-stationarity, and non-uniformity. The next step in this process is long overdue. We will introduce new local structures in chapter 8 and more

advanced topologies in subsequent parts of the theory. Fully local networks are the immediate next step and they are as essential as they are inevitable.

1.6 Historical and Bibliographical Remarks

Kolmogorov’s superposition theorem was stated in 1956 by A. N. Kolmogorov (albeit without a proof) [192]. An extension of the theorem that substituted the outer functions Φ_q with a single function Φ was proved by G. Lorenz in 1962 [222]. A further extension that replaced the inner functions $\Psi_{q,p}$ with a single function Ψ and affine transformations of its input and output was proved D. A. Sprecher in 1965 [323].

The word sigmoid derives from the word *sigma*, which is the name of the 18th letter of the Hellenic alphabet. It means curved like the letter C or S. Sigma once had a form like the Roman letter C, but sigma is equivalent to the English letter S. The second meaning is more common today. According to the Merriam-Webster dictionary, the first known use of the word (as C-shaped) goes back to 1670 [82].

Sigmoids have a long history in mathematics. The notation $\tan^{-1}$ for the inverse tangent function first appeared in 1813 in a paper by J. F. W. Herschel [157]. The logistic function was introduced in 1838 in a paper on population growth by P. F. Verhulst and was given its name in another paper by the same author in 1845 [351, 352]. A definition and a study of sigmoids as symmetric functions appeared in the author’s PhD thesis in 1995 [91, chapter 2].

The origins of the erroneous use of the term tensor to describe multidimensional arrays is not clear. In the context of neural networks, tensors have appeared at least as early as 2002 in the framework *Torch* to represent arrays and the two terms were used interchangeably and without sufficient attention to their differences [61]. Such usage also appeared in the *Theano* Python library – which is ironic given the stated purpose of the library to “define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays” [240]. The erroneous usage was subsequently adopted by other frameworks such as *TensorFlow* and *Pytorch* [170, 264] and most of the neural network community.

Turing machines were first defined in [344]. A. Turing’s quotation in section 1.5 is from [346].

Implicit cross-connections were introduced in *highway networks* by a team led by J. Schmidhuber [324]. Residual connections were introduced in *residual networks* [152].

An educational introduction to the structure of the cerebrum and white matter tracts appear in [205].

The term receptive field was introduced by C. S. Sherrington in 1906 to describe skin areas that could generate a scratch reflex in a dog [314]. H. K. Hartline applied the term to sensory neurons in 1938 [150].

D. H. Hubel and T. N. Wiesel were the first to identify local receptive

fields and differentiation between simple and complex cells in the visual system of the cat in 1962 [164]. They hypothesized a hierarchical model where simple and complex cells represent different stages of a hierarchical process, simple cells being created from thalamic inputs with visual receptive fields at the first stage and complex cells being generated from simple cells to detect orientation. They received the Nobel Prize in Physiology or Medicine in 1981 for their discoveries about the visual system and its processing.

The discovery that some complex cells receive mono-synaptic input from the thalamus was the first evidence against the hierarchical model and led to the introduction of the parallel model by K. P. Hoffmann and J. Stone in 1971 [160]. The recurrent model was introduced by R. J. Douglas, K. A. C. Martin, and D. Whitteridge in 1989 [89].

A general introduction to receptive fields appears in Scholarpedia [6]. An overview of complex receptive fields and the associated theories (hierarchical, parallel, recurrent) can be found in [226].

CNNs were inspired by the work of D. H. Hubel and T. N. Wiesel and were introduced in the *neocognitron* in 1980 [120]. They were later used in *time delay neural networks* with an emphasis on their translation invariance [356]. Despite their flawed design, CNNs have been used in computer vision, speech recognition, and natural language processing often with state-of-the-art results [398, 206, 207, 56, 57, 196, 62, 179]. (The power and importance of locality is indeed so great that it can overcome significant flaws.) An overview of computer vision contests won by deep CNNs appears in [302].

Local layers as well as a brief discussion about local statistics – which violate spatial stationarity and thus limit or invalidate the concept of convolution – appeared as early as 2013 in the domain of face recognition [334]. *Deformable convolutional networks* were introduced in 2017 [68] and have been applied to image classification and video restoration [403, 359]. Filter locality emerged in 2018 [402].

Pale Fire is a novel by V. Nabokov published in 1962 [246]. It's title is from Shakespeare's *Timon of Athens*, Act IV, scene 3:

The moon's an arrant thief,
And her pale fire she snatches from the sun

The above verse is often interpreted as a metaphor about novelty, creativity, and inspiration. (The title also constitutes a secondary Shakespearean reference to *Hamlet*, Act I, scene 5, where the Ghost notes that “The glowworm shows the matin to be near And 'gins to pale his uneffectual fire.”)

1.6.1 Attribution

The following are copyright attributions.

The image in figure 1.5 is from Wikipedia, BY-SA 4.0, created 15 December 2015. The image in figure 1.6 is by Cmglee - Own work, CC BY-SA 3.0,

<https://commons.wikimedia.org/w/index.php?curid=29924570>, uploaded on Wikipedia on 9 July 2019.

Πάντα ῥεῖ (everything flows).

— Heraclitus, c. 500 BCE.

2.1	Change	37
2.2	Machine Learning	38
2.3	Learning Paradigms	40
2.4	Learning as Optimization	44
	2.4.1 The Back-Propagation Method	48
	2.4.2 Neural Stochastic Gradient Descent	49
2.5	Loss Functions	50
2.6	Computational Limitations	57
2.7	Historical and Bibliographical Remarks	59

2

Learning

2.1 Change

Change as a universal property is briefly discussed in *Neural Age* [92, chapter 4]:

“Heraclitus (c. 535 – c.475 BCE) was a Hellenic philosopher who propounded that ever-present change is the essence of the universe. His phrase “πάντα ῥεῖ” (panta rhei) “everything flows” concisely describes his philosophy. Another similar quotation from Heraclitus appears in Plato’s dialogue *Cratylus*: “everything changes and nothing remains still.” It is striking that such a realization of the nature of the universe appeared so early.

More than 2000 years later, the German mathematician and philosopher Gottfried Leibniz (1646 – 1716) considered the question of a universal formal language that could unequivocally describe everything in the universe. Like Heraclitus, Leibniz realized the essential role of change and thus he set out to produce a formalism that could describe change in a quantitative way. The result was the discovery of differential and integral calculus, which is one of the most influential mathematical theories of all time. There is hardly any paradigm in physics that does not use differential equations. The theory set the stage for modern science and technology.”

Change is indeed fundamental. Measuring it is an important component of the sciences. However, there is a clear difference between measuring change and affecting or generating specific changes in a system, organism, or environment. When change is persistently made so that one or more functions are optimized and certain states can be achieved, then the change is *learning*.[†]

[†]This is not meant to be a definition of learning. It is rather a qualitative oversimplification. We will provide an expanded and formal definition in volume III.

There are various forms of learning from simple and usually local structural changes – that are described as *mutation* or *adaptation* in biology – to cognitive processes and knowledge acquisition as well as changes in the overall state of an environment or ecosystem. Learning thus spans a great range from elementary systems or organisms (such as bacteria) to entire ecosystems.

Learning is also a process for creating complexity. It is a prerequisite for life and intelligence. Entities that cannot adapt to changes in their environment cannot survive. They cannot be treated as intelligent no matter how extensive their knowledge is. An encyclopedia, for example, is not intelligent.

Overall, learning is an essential and foundational process for life and intelligence. It is also the main topic of this theory.

2.2 Machine Learning

Machine learning is a branch of computer science concerned with computing models or systems that can learn from data and interactions with their environment or *indirectly through their own internal processes*. (The latter clause derives from this theory and will be examined in more detail in volume III.)

The term machine learning was coined in 1959 by A. Samuel. The homonymous field grew substantially in the subsequent decades and particularly in conjunction with pattern recognition and statistics. However, the field still remains in a basic and preliminary state. Learning is still defined in qualitative, abstract, or extrinsic terms that provide no further insight. For example, in 1997, the following definition appeared:

“ A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T , as measured by P , improves with experience E . ”

The above definition is not comprehensive by any means. It is limited and extrinsic. It does not reveal anything about learning except one of its effects. Let us examine those issues in more detail.

Structural processes and changes (such as plasticity) often attempt to reduce resources or allocate them more efficiently but not necessarily with respect to specific improvements in predefined tasks T or any given experience E . And they may actually reduce performance through forgetting or interference. The same applies to various forms of learning over heterogeneous data sets where different data classes can interfere with each other or compete for resources, leading to a phenomenon that is known as *catastrophic interference* when no special provisions are in place.

The definition also assumes a persistent form of a measure P over experiences E for tasks T . This can easily apply when we train a system iterating over a predefined data set. However, it is not consistent with learning in

biological systems where experiences are fleeting rather than persistent. The same applies to robotics and every form of learning where the training data set is not actually stored in an explicit form.

Furthermore, the definition does not take cross-interactions of experiences and tasks into account. An experience E can affect performance on a task T that is largely unrelated. Even if we ignore memory and subjectivity, the associations are still inexact and can be very broad.

Those are significant limitations. At the very least, if we were to amend the above definition, we ought to remove the association between tasks T , experiences E , and performance P ; and we ought to add state and transitions as well as processes that reach certain states.

More importantly, learning can also be driven by generic factors that depend on the learning bias of the environment instead of specific or predefined tasks or experiences. A fundamental premise of this theory is that those factors are essential for learning processes.

Of equal importance are the facts that learning is principled (driven by a set of principles) and inherently recursive. When applied to complex tasks or systems, learning requires structure – which is also learned through evolutionary or other structural learning processes – and it operates recursively upon such structure and its elements. The premise that learning is merely some form of optimization upon an array of units or neurons is insufficient and untenable.

Overall, learning is a complex process with a long and elaborate formal definition that cannot possibly be reduced to a short sentence or paragraph without a significant reduction of its true nature and meaning. The definition of neural networks, as it appeared in the previous chapter (definition 1.2), spans almost a full page and requires a longer and detailed explanation to be understood. It will become considerably more complicated as we unfold this theory. Learning and intelligence are similar in that respect. The three concepts are intrinsically interwoven.

In current machine learning practice, learning is essentially optimization combined with few simple bias methods that are generically described as *regularization*. In artificial neural networks, the main structure is a set of layers; and each layer is a homogeneous or unstructured array of units. Structure is straightforward and certainly non-recursive. Binding and partitioning are similarly straightforward. Recursion is usually recurrence over the input and it is applied mainly to connectivity. Learning is reduced to the level of optimization. There are no explicit memory models or organization of memory. There is effectively no memory except for whatever is stored in the parameters of connections. Research on interference and retainment of knowledge is limited and at a basic level. (Most networks attempt to solve only one problem.) There is no dendritic or neural micro-structure. There is virtually no plasticity or structural learning. In most architectures, all structural elements are predefined and static. In other words, machine

learning currently covers only simple or rudimentary forms of learning.

Furthermore, the problems and the tasks are often reduced so that they can be addressed by such simplified forms. For example, computer vision is often reduced to a classification task over a static and predefined set of classes that cannot change dynamically during training. This is in direct contrast with vision in biological system where no classes are predefined, everything is essentially dynamic, and the system learns not only the classes but also probabilistic relations between them (e.g. a roof is usually *on the top* of a house and a horse is usually *pulling* a cart).

Of course, the field of machine learning had to start somewhere. Such simplifications in the methods, definitions, architectures, and even tasks are a natural and incremental way to proceed. However, they can also be quite misleading when we lose sight of the true goals and focus on ephemeral simplifications and customized solutions that can not apply or generalize to reality.

In this theory, we use the term machine learning with decreasing frequency. Learning is a unique and universal mathematical concept. The only argument in support of having two terms is ephemeral: machine learning is currently in a preliminary state of reduced complexity and sophistication and is not comparable with learning in nature and biology. However, this theory attempts to close or at least reduce the gap; the two terms are bound to merge in the end.

2.3 Learning Paradigms

Learning can be broadly classified into two categories. *Perceptive* or *extrinsic learning* is triggered or driven by external stimuli. *Cognitive* or *intrinsic learning* is primarily internal in the learning system and does not necessarily require external inputs.

The first category is more closely related to perception while the second is related to cognition. Information flows into the system in perceptive learning while it flows out of the system or circulates in it during cognitive learning.

Of course, those interpretations and relationships are approximate. Perception can contain various stages and it may involve both types of learning. The same applies to cognition. Most cognitive tasks usually involve one or more perceptive stages.

In this volume, we focus on perceptive learning. We will examine cognitive learning in more detail in volume III. Current cognitive methods in machine learning are primarily structural or generative and they are partly perceptive too.

Another significant limitation in current machine learning systems is that learning only takes place during an initial training stage. Processing following the training stage is usually called *inference* and it does not affect the learner. Thus, inference can be seen as perception that is not learning.

The above distinction is an artificial dichotomy if not an oxymoron. Perception is a significant and integral part of learning. We will combine the two in volume III.

Perceptive learning can be subdivided into three basic paradigms: supervised, unsupervised, and reinforcement learning.

Supervised learning requires the training data or stimuli S to be pairs (x, y) where x is the input to the learner and y is its desired output. The learner then develops a mapping f from x to y that usually minimizes an *error* or *loss function* ℓ which measures how closely $f(x)$ approximates y over S . Table 2.1 shows some common loss functions. $|S|$ indicates the number of pairs in S .

Loss name	$\ell(S)$	Comments/Conditions
Mean p -norm error	$\frac{1}{ S } \sum_{(x,y) \in S} \ f(x) - y\ _p^p$	$\ z\ _p = \left(\sum_{i \in z_*} z_i ^p \right)^{1/p}$ $p = 1$: Mean absolute error $p = 2$: Mean squared error
Huber δ loss	$\frac{1}{ S } \sum_{(x,y) \in S} \ell_\delta(f(x) - y)$	$\ell_\delta(z) = \sum_{i \in z_*} \begin{cases} z_i^2/2 & \text{if } z_i \leq \delta \\ \delta z_i - \delta^2/2 & \text{else} \end{cases}$
Cross entropy	$\frac{-1}{ S } \sum_{(x,y) \in S} y \cdot \ln(f(x))$	$0 \leq y, \quad 0 < f(x)$ $\ y\ _1 = \ f(x)\ _1 = 1$
Hinge loss	$\frac{1}{ S } \sum_{(x,y) \in S} H(f(x), y)$	$H(s, y) = \sum_{i \in y_*} \sum_{j \neq y_i} h(s_{y_i}, s_j)$ $h(t, v) = \max(0, 1 - t + v)$ $(\exists k \in \mathbb{N}) (\forall i \in y_*) y_i \in \mathbb{N}_k$

Table 2.1: Common loss functions.

In general, any differentiable metric* between $f(x)$ and y or any differentiable function $d(f(x), y)$ that is minimized when $f(x) = y$ can be used to generate a loss function. We will discuss loss functions more formally in the section 2.5.

The main disadvantage of supervised learning is that it requires the target value y to be provided for each data element x ; and thus it requires a teacher or supervisor to provide those targets. Consequently, supervised data sets tend to be smaller or limited and prone to human errors – when targets are provided by humans.

*A *metric* (or *distance*) on a set S is a function $m: S \times S \rightarrow \mathbb{R}$ that has the following properties:

- $(\forall x, y \in S) m(x, y) = 0 \Leftrightarrow x = y$ (*identity of indiscernibles*)
- $(\forall x, y \in S) m(x, y) = m(y, x)$ (*symmetry*)
- $(\forall x, y, z \in S) m(x, z) \leq m(x, y) + m(y, z)$ (*triangle inequality*)

On the other hand, *unsupervised learning* is a type of learning where only the data x are present in the data set S . The learner then learns a mapping f that has certain properties. The most common forms are the following:

Partitions. In partition methods, $f(x)$ defines a partitioning of the data x in S (that is often interpretable). *Clustering* is such a form of learning where the data are partitioned into a fixed or dynamically determined set of clusters using a variety of criteria including similarity or proximity, probability distributions, volume density, and graph connectivity.*

Representations. Unsupervised representation methods derive different representations of the input that usually satisfy certain constraints. There are three main approaches.

- $f(x)$ is an alternative representation of x and it is often a compressed form that retains useful information but discards noise and redundant or irrelevant information. This type of learning is used in *auto-encoders*, where a composite automorphism or identity mapping $g(f(x)) = x$ is learned and then the mapping f defines the compressed or alternative representation.
- $f(x)$ is a representation of x that retains underlying isomorphisms or similarities. For example, *word embeddings* are representations of words that retain semantic similarities. They derive from distributions of words in large corpora and the underlying *distributional hypothesis* that linguistic items with similar distributions have similar meanings and vice versa. (As J. Firth famously stated in the 1950s: “You shall know a word by the company it keeps.”)
- $f(x)$ is a representation of x that has associative or topological properties and is learned through iterative neural methods as in *Hebbian* or *generalized Hebbian learning* and *self-organized maps*.

Adversarial approaches.[†] Adversarial methods train two (or more) networks with contrasting objectives. The first network represents a model for the environment or specific entities in it. It is often a generator of such entities and it operates on latent (and often random) input. The second network represents the environment or a predictor of it. It operates on input that is either output from the generator or data from the environment, which is often a data set. The generator learns by attempting to minimize the predictability of its generated data or their properties. The predictor learns to predict generated data or their properties and often only to distinguish generated from actual data. The two networks may use two different loss functions or share a

*There are also non-partitioning methods, such as *soft clustering*, where the clusters can overlap.

[†]Not to be confused with *adversarial learning* where the goal is to force models to make errors or fail through malicious input.

common loss function that is minimized by one of them and maximized by the other.

The approach first appeared in 1991 as *adversarial curiosity* and *predictability minimization*. It was rediscovered in the 2010s and was popularized as *generative adversarial networks*.

Adversarial approaches blur the line between perceptive and cognitive learning as well as between unsupervised and reinforcement learning, which we describe next. The term *self-supervised* is sometimes used to describe such approaches.

*Reinforcement learning** assumes that the learner and the environment contain state, that the learner can apply actions that probabilistically change the environment's or the learner's state, that each state is associated with a reward (which can be negative and thus be punishment), and that the learner collects the reward at each state and attempts to find a sequence of actions that would maximize the expected total reward.

Reinforcement learning is quite generic and it appears not only in machine learning but also in genetic algorithms, operations research, control theory, game theory, information theory, and statistics. It is useful in situations where an evident final reward, such as when a player wins a game or a customer purchases a product after a discussion with a chatbot, should be regressed through a set of actions that have led to the final result and its reward.

Supervised learning is arguably the simplest and perhaps least significant of the above three basic paradigms. Its requirement of a teacher effectively implies that neither life nor intelligence can emerge on their own using only supervised approaches.

However, supervised learning is powerful and efficient. It permits simpler and effective forms that can be treated independently. It can transfer knowledge from data or the environment to active neural states with relatively low effort. As a result, it is the most common paradigm used in machine learning research and applications.

Supervised learning is also the focus of volume I of this book series. Presentation order is one of the main factors but not the only one. We start the theory from a mathematical and computational viewpoint and gradually approach biology; and we can advance it considerably in the supervised paradigm and remain strictly within its bounds.

Of course, true learning is not supervised, unsupervised, or reinforced. It is neither perceptive nor cognitive. It is rather a hybrid, a combination of all those forms. This is evident in complex biological organisms; and it is gradually becoming a reality in machine learning. For example, neural natural language processing systems adopt supervised approaches to solve specific problems but they also employ word or character embeddings that are

*Also known as *neuro-dynamic programming* or *approximate dynamic programming* in control theory and operations research.

learned from corpora using unsupervised methods. Reinforcement learning is used in conversational applications and chatbots and in combination with supervised methods and embeddings.

We will return to these forms of learning and their hybrid combinations in volume II and more in volume III. Ultimately, we would like to create a learning paradigm that does not require supervised data but can use them with great efficacy whenever they are available.

2.4 Learning as Optimization

In the following, we assume a neural network (G, F, Y) over $(\mathcal{F}, \mathcal{P}, \mathcal{A})$ with $n + 1$ layers as in definition 1.2. We furthermore assume a supervised learning setup with a loss function ℓ that is a function of the outputs of the network and the supervised targets. If y is the desired output of the network upon input x , then it will be $Y_0 = x$ and $\ell = \ell(Y, y)$.

Of course, if the loss function only depends on units in L_n , as is often the case, we will only need Y_n to calculate the loss. In other words, it will be $\ell = \ell(Y_n, y)$.

However, this is not necessarily true with networks of this theory. The loss function may depend on intermediate outputs of the network Y_k for any $k < n$. We indicate such dependence as *remote loss*. For example, remote losses will appear when we use different loss functions for different parts of a larger network and we aggregate the losses. We will elaborate on this in chapter 4.

In order to handle remote loss, we need to distinguish between two different forms of partial derivatives. We define $\frac{\partial f(x, y)}{\partial x}$ to be the partial derivative of $f(x, y)$ with respect to x . We also define $\nabla_x f(a, b)$ to be the partial derivative of f with respect to x evaluated at (a, b) . In other words,

$$\nabla_x f(a, b) = \frac{\partial f}{\partial x}(a, b) = \left. \frac{\partial f(x, y)}{\partial x} \right|_{x=a, y=b}.$$

Of course, the two definitions are identical when a and b are independent variables. However, they can be different when the variables are dependent. For example, if $f(x, y) = x^2 + y^2$ then $\frac{\partial f(x, x)}{\partial x} = 4x$ but $\nabla_x f = 2x$ and thus $\nabla_x f(x, x) = 2x$.

We will also use the notation $f'(x)$ to indicate $\frac{\partial f(x)}{\partial x}$ that is when the derivative is with respect to all input variables of the function f .

In general, layers contain *learnable* parameters that are learned (adjusted) during training. If the goal is to adjust those parameters in order to minimize a given loss function, a straightforward iterative method is *gradient descent*: at each step the parameters are decreased by an amount that is proportional to the gradient of the loss function with respect to the parameters.

Therefore, if ℓ is the loss function, w_t are the learnable parameters, and

Δw_t is the adjustment of w_t at time or iteration t , gradient descent specifies that

$$\Delta w_t = w_{t+1} - w_t = -\lambda_t \frac{\partial \ell}{\partial w_t}, \quad (2.1)$$

where λ_t is the *learning rate* (or *step size*) that affects how large the adjustment would be (and thus indicates how much we ‘trust’ the gradient to maintain its sign).

Computing the overall loss function or the exact gradient is often computationally expensive or even impossible. In that case, an estimate of the gradient (or its sign) can be used and the method is called *stochastic gradient descent*.

In neural networks, the loss function ℓ is approximated by the loss ℓ_t which is over one or more samples used at training iteration t . Thus the parameter update becomes

$$\Delta w_t = -\lambda_t \frac{\partial \ell_t}{\partial w_t}. \quad (2.2)$$

In order to simplify notation, we drop the time subscript t and thus write $\Delta w = -\lambda \frac{\partial \ell}{\partial w}$. We also assume the notation $[\phi]$ to denote the numerical value of condition ϕ (0 for false and 1 for true).

We furthermore define ψ as an operator that accepts a function as input and returns the learnable parameters of the function. Hence, for $k \leq n$, ψf_k , ψp_k , and ψa_k are the learnable parameters of the non-polynomial, polynomial, and aggregating functions of layer L_k , respectively. (We call this the *ψ -notation*.*) Finally, we define the derivative of a function $g(x)$ with respect to its learnable parameters ψg as $\nabla_{\psi} g(x) = \frac{\partial g(x)}{\partial \psi g}$.

We can now prove the following theorem:

Theorem 2.1 (Back-propagation). *Assume a neural network (G, F, Y) over $(\mathcal{F}, \mathcal{P}, \mathcal{A})$. As in definition 1.2, assume that the network has $n + 1$ layers $L_0, \dots, L_n$ and that the layers do not share learning parameters. Furthermore assume that ℓ is a loss function for the network. Then the partial derivatives of ℓ with respect to the parameters of each layer L_k are given by the following equation:*

$$\frac{\partial \ell}{\partial \psi_k} = \frac{\partial \ell}{\partial Y_k} F'_k(Y_{I_k}) \nabla_{\psi} F_k(Y_{I_k}), \quad (2.3)$$

where $\psi_k = \psi F_k$. The partial derivative of ℓ with respect to Y_k can be calculated recursively (in reverse order from $k = n$ to 0):

$$\frac{\partial \ell}{\partial Y_k} = \nabla_{Y_k} \ell + \sum_{\substack{r > k \\ I_r \cap L_k \neq \emptyset}} \frac{\partial \ell}{\partial Y_r} F'_r(Y_{I_r}) \nabla_{Y_k} Y_{I_r}. \quad (2.4)$$

* ψ is the first letter of the word $\psi\epsilon\upsilon\delta\acute{\epsilon}\varsigma$ in Hellenic, which means “false” or “lying”. The parameters of a learner can be seen as being in a false or incorrect state initially but they are likely to transition to a true or more correct state through learning.

Proof. Given that $Y_k = F_k(Y_{I_k})$ the derivative of the loss with respect to the learnable parameters of each layer (equation (2.3)) can be computed with a straightforward application of the chain rule of calculus:

$$\frac{\partial \ell}{\partial \psi_k} = \frac{\partial \ell}{\partial Y_k} \frac{\partial Y_k}{\partial \psi_k} = \frac{\partial \ell}{\partial Y_k} \nabla_{\psi} F_k(Y_{I_k}).$$

Furthermore, the output Y_k of layer L_k can affect the loss directly through regular or remote loss but it can also affect it indirectly through any other layer L_r (for $r > k$) and successive or cross-connections to that layer. This situation is depicted in figure 2.1. Dashed arrows indicate (remote) loss dependencies.

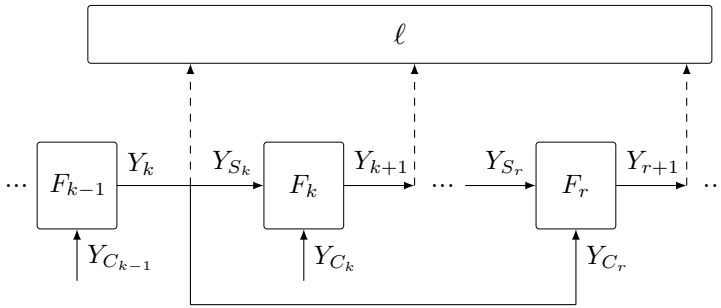


Figure 2.1: Potential dependency paths between the loss ℓ and outputs Y_k .

Taking those dependencies into account, we can apply the chain rule once more to calculate the derivative of the loss with respect to layer outputs

$$\frac{\partial \ell}{\partial Y_k} = \nabla_{Y_k} \ell + \sum_{\substack{r > k \\ I_r \cap L_k \neq \emptyset}} \frac{\partial \ell}{\partial Y_r} \frac{\partial Y_r}{\partial Y_{I_r}} \nabla_{Y_k} Y_{I_r} = \nabla_{Y_k} \ell + \sum_{\substack{r > k \\ I_r \cap L_k \neq \emptyset}} \frac{\partial \ell}{\partial Y_r} F'_r(Y_{I_r}) \nabla_{Y_k} Y_{I_r}.$$

□

As shown in the proof, equation (2.3) is an immediate consequence of the chain rule. Equation (2.4) can be intuitively understood as a separation of dependency paths where the first term captures direct influence of Y_k to the loss ℓ (as remote loss if $k < n$), and the rest of the terms derive from successive or cross-connections, if any, from Y_k to Y_r for $r > k$ (see figure 2.1).

Corollary 2.1. *Assuming that $F_k = f_k \circ p_k \circ a_k$, $f_k \in \mathcal{F}$, $p_k \in \mathcal{P}$, and $a_k \in \mathcal{A}$,*

for $k = 1, 2, \dots, n$, the back-propagation theorem is equivalent to:

$$\begin{aligned}\frac{\partial \ell}{\partial \psi f_k} &= \frac{\partial \ell}{\partial Y_k} \nabla_{\psi} f_k(P_k) \\ \frac{\partial \ell}{\partial \psi p_k} &= \frac{\partial \ell}{\partial Y_k} f'_k(P_k) \nabla_{\psi} p_k(A_k) \\ \frac{\partial \ell}{\partial \psi a_k} &= \frac{\partial \ell}{\partial Y_k} f'_k(P_k) p'_k(A_k) \nabla_{\psi} a_k(Y_{I_k})\end{aligned}\tag{2.5}$$

and

$$\begin{aligned}\frac{\partial \ell}{\partial Y_k} &= \nabla_{Y_k} \ell + \frac{\partial \ell}{\partial Y_r} f'_r(P_r) p'_r(A_r) \frac{\partial a_r(Y_{I_r})}{\partial Y_{S_r}} \nabla_{Y_k} Y_{S_r} \Big|_{r=k+1} \\ &\quad + \sum_{\substack{r > k+1 \\ C_r \cap L_k \neq \emptyset}} \frac{\partial \ell}{\partial Y_r} f'_r(P_r) p'_r(A_r) \frac{\partial a_r(Y_{I_r})}{\partial Y_{C_r}} \nabla_{Y_k} Y_{C_r},\end{aligned}\tag{2.6}$$

where $A_k = a_k(Y_{I_k})$ and $P_k = p_k(A_k)$ for $k = 1, \dots, n$.

Proof. Equations (2.5) are direct applications of the chain rule as in the proof of the back-propagation theorem.

Moreover, for $r > k$, it is

$$\frac{\partial Y_r}{\partial Y_k} = \frac{\partial Y_r}{\partial P_r} \frac{\partial P_r}{\partial A_r} \frac{\partial A_r}{\partial Y_k} = f'_r(P_r) p'_r(A_r) \left\{ \begin{array}{ll} \frac{\partial A_r}{\partial Y_{S_r}} \nabla_{Y_k} Y_{S_r} & \text{if } r = k+1 \\ \frac{\partial A_r}{\partial Y_{C_r}} \nabla_{Y_k} Y_{C_r} & \text{if } r > k+1 \end{array} \right\}.$$

The last equation derives from the fact that Y_k can affect Y_{S_r} but not Y_{C_r} when $r = k+1$ ($I_{k+1} \cap L_k = S_{k+1}$, by definition); and the fact that Y_k can directly affect Y_{C_r} but not Y_{S_r} when $r > k+1$ ($I_r \cap L_k = (S_r \cup C_r) \cap L_k = C_r \cap L_k$, since S_r only contains units of L_{r-1} and thus cannot contain units of L_k). Finally, equation (2.6) derives from the combination of the above equation and the following application of the chain rule:

$$\frac{\partial \ell}{\partial Y_k} = \nabla_{Y_k} \ell + \sum_{\substack{r > k \\ I_r \cap L_k \neq \emptyset}} \frac{\partial \ell}{\partial Y_r} \frac{\partial Y_r}{\partial Y_k}.$$

□

In most cases, it would be $S_{k+1} = L_k$ and thus $Y_{S_{k+1}} = Y_k$. In that case, the Jacobian matrix $\nabla_{Y_k} Y_{S_{k+1}}$ will be the identity matrix and it can be omitted from the calculations in equation (2.6). On the other hand, when $S_{k+1} \subset L_k$, the Jacobian matrix will contain ones in the positions that correspond to the included units and zeroes everywhere else. More precisely, it will be $(\nabla_{Y_k} Y_{S_{k+1}})_{i,j} = [(S_{k+1}, i) \equiv_Y (L_k, j)]$.

The same applies to the Jacobian $\nabla_{Y_k} Y_{C_r}$ (from equation (2.6)) with the exception that cross-connections cover entire layers L_k less frequently.

We should note that we assumed that the layers do not share learning

parameters with each other. This is usually the case. However, if sharing is used, the chain rule must take it into account. Therefore, if layers $k_1, \dots, k_m$ have some common learning parameters w , then it is

$$\frac{\partial \ell}{\partial w} = \sum_{j=1}^m \frac{\partial \ell}{\partial Y_{k_j}} F'_{k_j}(Y_{I_{k_j}}) \frac{\partial F_{k_j}(Y_{I_{k_j}})}{\partial w}. \quad (2.7)$$

We can better understand theorem 2.1 and how it incorporates remote losses with an example.

Example 2.1. We assume neural networks $M = (V_1, E_1, F, Y)$ and $N = (V_2, E_2, H, Z)$ over $(\mathcal{F}, \mathcal{P}, \mathcal{A})$. We assume that both networks have $n + 1$ layers. We assume that Y_{n+1} and Z_{n+1} have the same dimensionality; and we assume the same for Y_0 and Z_0 . We can then create an aggregate network that contains those two networks and averages their outputs. Thus the aggregate has an extra layer L_{n+2} such that $Y_{n+2} = (Y_{n+1} + Z_{n+1})/2$. If we assume mean squared error as loss, we can define the loss function of the aggregate network to be $\ell = \|Y_{n+1} - y\|^2 + \|Z_{n+1} - y\|^2 + \|Y_{n+2} - y\|^2$, where y is the desired output of the network upon input x and $Y_0 = Z_0 = x$.

Obviously, it is

$$\begin{aligned} \partial \ell / \partial Y_{n+2} &= 2(Y_{n+2} - y) \\ \partial \ell / \partial Y_{n+1} &= 2(Y_{n+1} - y) + (Y_{n+2} - y) \\ \partial \ell / \partial Z_{n+1} &= 2(Z_{n+1} - y) + (Y_{n+2} - y) \\ \partial \ell / \partial Y_n &= (\partial \ell / \partial Y_{n+1})(\partial Y_{n+1} / \partial Y_n) \\ \partial \ell / \partial Z_n &= (\partial \ell / \partial Z_{n+1})(\partial Z_{n+1} / \partial Z_n) \end{aligned}$$

and so on.

Given that the two sub-networks M and N have no interconnected parts (other than their outputs), the rest of the derivatives are restricted within each network and they are exactly as described in the back-propagation theorem.

2.4.1 The Back-Propagation Method

There is a long history behind the calculations of the back-propagation theorem. Various simpler versions appeared in the 1960s, 1970s, and 1980s. The first application in neural networks appeared in 1982. The method was rediscovered and popularized as back-propagation in 1986. More historical details can be found in section 2.7.

In its original form (1986), back-propagation assumes unidimensional element-wise activation functions, affine polynomial transformations, mean squared error as the loss function, and no remote loss.*

*For each unit u , back-propagation defines

$$\delta_u = \partial \ell / \partial Y_u = \left\{ \begin{array}{ll} 2(Y_u - t_u) & \text{if } u \text{ is an output unit} \\ \sum_{(u,v) \in E} \delta_v F'_v(Y_{I_v}) W_{u,v} & \text{if } u \text{ is a non-output unit} \end{array} \right\},$$

Of course, there is no reason to adopt such limitations. The back-propagation theorem and its proof are straightforward in their most generic form.

2.4.2 Neural Stochastic Gradient Descent

The back-propagation theorem (or its corollary 2.1) can be used to modify the parameters of a network using gradient descent over its loss function. The algorithm is straightforward. It is shown in figure 2.2 and it is known as *neural network stochastic gradient descent* (often abbreviated as SGD). We use the term *neural gradient descent* for it.

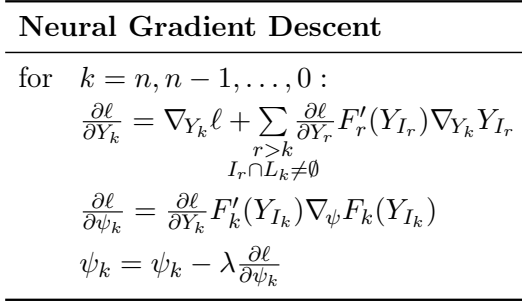


Figure 2.2: Neural network stochastic gradient descent.

Neural gradient descent operates in a reverse order from the last to the first layer. This is a direct consequence of the chain rule. Indeed, if $y = f(x)$ and $z = g(y)$ then $\partial z / \partial x = (\partial z / \partial y)(\partial y / \partial x)$. Thus, in order to compute $\partial z / \partial x$ we need to compute $\partial z / \partial y$ first.

Of course, it is not memory efficient to compute and store all the above derivatives if they are used only once. For example, when there are no cross-connections we can process one layer at a time and only store the derivatives for the current layer L_k . The derivatives for the layers L_r , for $r > k$, can then be discarded.

On the other hand, when there are cross-connections into layer L_k , we must keep its derivatives until we process all layers L_j that provide cross-connections into L_k . If we define π_k to be the index of the first layer that provides input to layer L_k (the word “first” starts with the later π in Hellenic) and π_j^{-1} to be the inverse relation and thus the set of layers that have L_j as their first input layer then it will be

$$\begin{aligned} \pi_k &= \min\{j \in \mathbb{N}_k / L_j \cap I_k \neq \emptyset\}, & \text{for } 0 < k \leq n, \\ \pi_j^{-1} &= \{k \in \{j+1, \dots, n\} / \pi_k = j\}, & \text{for } 0 \leq j < n. \end{aligned} \quad (2.8)$$

where t_u is the target value of output unit u and $W_{u,v}$ is the weight of the connection from unit u to unit v . And, of course, $\partial \ell / \partial W_{u,v} = \delta_v F'_v(Y_{I_v}) Y_u$ and $\partial \ell / \partial b_v = \delta_v F'_v(Y_{I_v})$, where b_v is the bias of the unit v .

Given the obvious tautology that $k \in \pi_{\pi_k}^{-1}$, if we assume that l_u returns the index of the layer that contains unit u (that is $u \in L_{l_u}$), we can efficiently construct the π^{-1} sets as shown in figure 2.3. The time complexity of the algorithm is $O(|E|)$, if we assume that l_u is stored or can be calculated in $O(1)$ time.

π-Set Construction	
Initialization:	for $k = 1, \dots, n$: $\pi_k = n + 1$; $\pi_{k-1}^{-1} = \emptyset$
First layers:	for $(u, v) \in E$: $\pi_{l_v} = \min(\pi_{l_v}, l_u)$
π -sets:	for $j = 1, \dots, n$: $\pi_{\pi_j}^{-1} = \pi_{\pi_j}^{-1} \cup \{j\}$

Figure 2.3: π -set construction.

Given the π -sets, we can easily modify neural gradient descent to be memory efficient as shown in figure 2.4. This is a very simple memory

Memory Efficient Neural Gradient Descent	
for $k = n, n - 1, \dots, 0$:	
	$\frac{\partial \ell}{\partial Y_k} = \nabla_{Y_k} \ell + \sum_{\substack{r > k \\ I_r \cap L_k \neq \emptyset}} \frac{\partial \ell}{\partial Y_r} F'_r(Y_{I_r}) \nabla_{Y_k} Y_{I_r}$
	$\frac{\partial \ell}{\partial \psi_k} = \frac{\partial \ell}{\partial Y_k} F'_k(Y_{I_k}) \nabla_{\psi} F_k(Y_{I_k})$
	$\psi_k = \psi_k - \lambda \frac{\partial \ell}{\partial \psi_k}$
	for $r \in \pi_k^{-1}$: Release memory of $\frac{\partial \ell}{\partial Y_r}$

Figure 2.4: Memory efficient neural gradient descent.

management scheme. We will elaborate on memory issues in volumes II and III.

In current neural network frameworks and libraries, there is no built-in mechanism for cross-connections. Users have to manage them manually. Although this is a reasonable compromise for current networks, it will become a significant impediment as the architectures become more complex.

2.5 Loss Functions

Loss functions represent one or more mathematical objectives. For supervised learning, the most common objective is that a learned function f approximates an unknown function that generated the training data. If learning requires derivatives of the loss function ℓ then, of course, ℓ must be differentiable almost everywhere. Continuous differentiation is a preferable condition but even discontinuities in ℓ can be handled as long as they are sparse (finite or countable).

As noted in the section 2.3, any differentiable metric or function $d(z, y)$ that is minimized when $z = y$ can be used to measure the error or the difference between $f(x)$ and the target value y for each pair (x, y) in the training set S . The loss function can then be the maximum, sum, mean or some other aggregate of the errors. For example, the mean is used in the loss functions of table 2.1.

In statistics, it is known that the mean absolute error is an unbiased estimator with respect to the median of the data samples (or geometric median in the case of multiple dimensions) while the mean squared error is an unbiased estimator with respect to the arithmetic mean of the samples.

However, the arithmetic mean is sensitive to outliers. The same applies to the mean squared error. The square in the exponent tends to amplify large errors and disproportionally increase their effect. For example, when the difference $\|Y_u - y_u\|$ is large for an output unit u and larger than the differences of the other output units, that output of u may dominate the loss function and its derivatives. Similarly, assuming that Y_n is the output layer, when $\|Y_n - y\|_2$ is large for a data sample (x, y) the sample may dominate the rest of the samples. And the sensitivity is even higher for p -norm errors with $p \geq 2$.

On the other hand, the Huber loss (see table 2.1) is quadratic for small differences but linear for large ones. This property allows it to combine the sensitivity of a mean-unbiased estimator (for small differences) with the robustness of a median-unbiased estimator (for large differences).

The *pseudo-Huber loss function* is a smooth approximation of Huber loss. It is defined as follows

$$\ell_\delta(z) = \delta(\sqrt{z^2 + \delta^2} - \delta). \quad (2.9)$$

When z is considerably smaller than δ , the above function is approximately quadratic: $\ell_\delta(z) = \delta z^2 / (\sqrt{z^2 + \delta^2} + \delta) \approx z^2/2$. However, when z is sufficiently larger than δ , the function is approximately linear: $\ell_\delta(z) \approx \delta\sqrt{z^2} = \delta z$. As a result, pseudo-Huber loss shares the same sensitivity and robustness properties with Huber-loss while it is clearly a smooth function.

In machine learning and statistics, *classification functions* are functions with a discrete and finite range. They are used to assign each element x of their domain into one or more classes. The former case is called *single-label classification* while the later is called *multi-label classification*. (In the second case, the range of the classification function is the power set* of the discrete and finite set that represents the classes; and, of course, the power set is discrete and finite, too.) The overall process is called *classification*. For example, recognizing a handwritten digit is a classification problem into ten classes, one for each decimal digit.

Cross entropy is a common and relatively effective loss for classification

*The *power set* of a set S is the set of all subsets of S , denoted as 2^S .

problems (and especially single-label ones). It derives from the concept of *cross entropy* between two *probability distributions* defined over the same sample and event spaces. If P and Q are two such distributions, the cross entropy of Q relative to P is $H(P, Q) = -E_P(\ln Q)$. For discrete distributions it is thus $H(P, Q) = -\sum_x P(x) \ln(Q(x))$; while for continuous distributions with densities p and q that are absolutely continuous with respect to a (Lebesgue) measure μ , it is $H(P, Q) = -\int p(x) \ln(q(x)) d\mu(x)$.

The strength of cross entropy loss derives from its relation to probability distributions and the use of softmax to enhance the difference of the maximum output from the rest. Thus small differences can result in significant reductions of the loss.

In probability theory, *probability distributions* are associated with random variables and they are often represented by their *probability density function* (pdf). We would usually use capital letters like P and Q to indicate probability distributions and the corresponding lower case letter (e.g. p and q) to indicate their density functions.

When a random variable X is continuous with probability distribution P , the density function p is the derivative of P ; that is $p(x) = \frac{\partial P(x)}{\partial x}$. When P has discontinuities (or is not differentiable for any other reason), the derivative is not defined in the standard sense. At each discontinuity, the density p contains an *impulse* or *Dirac delta function* δ with amplitude equal to the jump of the discontinuity.* Similarly when X is discrete, taking values $x_0, x_1, \dots$ with probabilities $r_0, r_1, \dots$ then the density is $p(x) = \sum_i r_i \delta(x - x_i)$.

The result is that a probability distribution P and density p of a random variable X always satisfy the equation $P(x) = \int_{-\infty}^x p(t) dt$. (They generally satisfy the equation $\text{Prob}(X \in A) = \int_A p(x) dx$ for every set A in the probability space of X .) Consequently, the introduction of the *delta* function unifies the notation and allows us to treat the probability distributions of both continuous and discrete variables as integrals instead of using summations in the latter case.

Probability distributions are relevant to loss functions because the output Y_n of a neural network or its target value y can define a distribution. (Y_n can always be transformed into a distribution through operations such as softmax or softmin. For example, softmax maps $Y_{n,i}$ to $e^{Y_{n,i}} / \sum_{j \in Y_{n*}} e^{Y_{n,j}}$, the latter quantities are positive and sum up to 1.)

*The Dirac delta function is not a function with the conventional definition. It is rather a *distribution* or *generalized function*. Those are mathematical entities that generalize the concept of a function. More precisely, distributions are linear forms that map functions to real numbers. The Dirac delta function is simply the distribution that maps a smooth function σ to $\sigma(0)$. This is often written as $\langle \delta, \sigma \rangle = \sigma(0)$. The delta function can be interpreted as the probability density of a single point source. As a result, the Dirac delta function allows us to differentiate discontinuous functions, which are not differentiable in the conventional sense. The delta function itself is the density function of the step probability distribution $S(x) = [x \geq 0]$; and it has the property that $\int_{\mathbb{R}} \sigma(x) \delta(x) dx = \sigma(0)$, if σ is a locally integrable function at 0. The Dirac delta function is sometimes defined as such albeit that is a cyclical definition.

Consequently, we can define a loss function to be an estimate of the differences or discrepancies of two probability distributions. Let us consider the following definition.

Definition 2.1. A *divergence function* (a.k.a. *contrast function*) is a real-valued function D on pairs of probability distributions P and Q over the same space Ω that has the following two properties:

- a. $(\forall P, Q) D(P, Q) \geq 0$
- b. $(\forall P, Q) D(P, Q) = 0 \Leftrightarrow P = Q$

In that case, we say that $D(P, Q)$ is the divergence of P from Q . (Note that divergence is not a metric. It is not necessarily symmetric and it may not satisfy the triangle inequality.)

Divergences are minimally constrained. There is an infinite and diverse multitude of such functions. For example, we can use convex functions to generate various divergences using the following lemma.

Lemma 2.1 (*f*-divergence). *If P and Q are probability distributions over the same space Ω , Q is absolutely continuous with respect to P (that is $dP(x) = 0 \Rightarrow dQ(x) = 0$), and $f: \mathbb{R}_+ \rightarrow \mathbb{R}$ is a non-affine convex function such that $f(1) = 0$ then*

$$D_f(P, Q) = E_P(f \circ \frac{dQ}{dP}) = \int_{\Omega} f\left(\frac{dQ}{dP}\right) dP \quad (2.10)$$

is a divergence function of P from Q .

Proof. The first condition of divergences is implied by the convexity of f and Jensen's inequality:*

$$D_f(P, Q) = E_P(f(\frac{dQ}{dP})) \geq f(E_P(\frac{dQ}{dP})) = f(\int dQ) = f(1) = 0.$$

For the second condition, we can observe that $D_f(P, Q) = E_P(f(1)) = 0$, when $P = Q$. Conversely, when $D_f(P, Q) = 0$, equality must hold in the above application of Jensen's inequality. Thus, either $f(dQ/dP)$ is affine or P is a step function. The former implies that $dP = dQ$, given that f is not affine.[†] When P is a step function, dP must be zero everywhere except at a single point x ; and because Q is absolutely continuous with respect to P , dQ must be likewise zero. As a result, $dP(t) = dQ(t) = \delta(t - x)$. In either case, it is $dP = dQ$ and thus $P = Q$. $\square$

*Jensen's inequality states that $f(E_P) \leq E_P(f)$ when f is a convex function and P is a probability distribution with expectation E_P ; equality holds iff f is affine or the underlying random variable is constant, which means that P is a step function.

[†]This helps us understand the constraint better. If we allowed f to be affine, that is if $f(z) = az + b$, then $D_f(P, Q) = E_P(a\frac{dQ}{dP} + b) = a + b = f(1) = 0$ and the 'divergence' would be trivially zero.

A divergence generated by a convex function f as in lemma 2.1 is called *f-divergence*. Table 2.2 shows some common cases of such divergence functions. (For simplicity, we use the density functions p and q , assuming that both distributions are absolutely continuous with respect to a measure μ , and we drop the argument x from the densities and the differential $d\mu(x)$ from the integral. Thus we write $\int p \ln \frac{q}{p}$ instead of $\int p(x) \ln \frac{q(x)}{p(x)} d\mu(x)$.)

Name	$\mathbf{f}(\mathbf{t})$	$\mathbf{f}$ -divergence $\mathbf{D}_{\mathbf{f}}(\mathbf{P}, \mathbf{Q})$
α -divergence	$\frac{t^\alpha - 1}{\alpha(\alpha - 1)}$	$D_\alpha(P, Q) = \frac{(\int p^{1-\alpha} q^\alpha) - 1}{\alpha(\alpha - 1)}$
χ^2 measure	$\frac{(1-t)^2}{1+t}$	$\chi^2(P, Q) = \int \frac{(p-q)^2}{p+q}$
Jeffrey's divergence	$(t-1) \ln(t)$	$D_J(P, Q) = \int (p-q) \ln \frac{p}{q}$
Kagan's divergence	$(t-1)^2$	$D_{\chi^2}(P, Q) = \int \frac{(p-q)^2}{p}$
Relative entropy or Kullback–Leibler divergence	$-\ln(t)$	$D_{\text{KL}}(P, Q) = \int p \ln \frac{p}{q}$
Squared Hellinger distance	$\frac{1}{2}(\sqrt{t} - 1)^2$	$H^2(P, Q) = \frac{1}{2} \int (\sqrt{p} - \sqrt{q})^2$
Total variation norm	$ t - 1 $	$D(P, Q) = \int p - q $

Table 2.2: Common f -divergence functions.

An alternative way to create divergence functions from convex functions is the following lemma.

Lemma 2.2 (Bregman divergence). *If n is a positive integer and $f: \mathbb{R}^n \rightarrow \mathbb{R}$ is a differentiable and strictly convex function then the*

$$B_f(x, y) = f(x) - f(y) - f'(y)(x - y) \quad (2.11)$$

is a divergence function of x from y .

Proof. Strict convexity of f implies that $f(x) - f(y) - f'(y)(x - y) > 0$, for $x \neq y$. Thus $B_f(x, y) > 0$, if $x \neq y$; and $B_f(x, y) = 0$ iff $x = y$. $\square$

A divergence generated by a convex function f as in lemma 2.2 is called *Bregman divergence*. Table 2.3 shows some common cases of such divergence functions.

As noted, divergence functions are not necessarily metrics. Symmetry is not hard to achieve. For example, if $D(P, Q)$ is a divergence function, $D(P, Q) + D(Q, P)$ is also a divergence and it is symmetric. However, the triangle inequality requires more effort and it is not always possible to incorporate.

We can always generate a metric from a strictly convex function f and its Bregman divergence using the following lemma.

Name	$\mathbf{f}(\mathbf{x})$	Bregman div. $\mathbf{B}_{\mathbf{f}}(\mathbf{x}, \mathbf{y})$
Squared Euclidean distance	x^2	$(x - y)^2$
Mahalanobis distance	$\frac{1}{2}x^T Q x$	$\frac{1}{2}(x - y)^T Q (x - y)$
Relative entropy	$\sum_i x_i \ln x_i$	$\sum_i x_i \ln \frac{x_i}{y_i} - \sum_i x_i + \sum_i y_i$
Itakura–Saito distance	$-\sum_i \ln x_i$	$\sum_i \left(\frac{x_i}{y_i} - \ln \frac{x_i}{y_i} - 1 \right)$

Table 2.3: Common Bregman divergence functions.

Lemma 2.3. *If n is a positive integer, $f: \mathbb{R}^n \rightarrow \mathbb{R}$ is a differentiable and strictly convex function, B_f is its Bregman divergence, A and B are $n \times n$ positive definite matrices* and I is the $n \times n$ identity matrix, then generalized symmetrical Bregman divergence (GSB) is defined as*

$$G_f(x, y) = B_f(x, y) + B_f(y, x) + \frac{1}{2}(x - y)^T A(x - y) + \frac{1}{2}(f'(x) - f'(y))^T B(f'(x) - f'(y))$$

and it is a metric iff $AB - I$ is positive semi-definite.

Proof is omitted. A reference that contains the proof can be found in section 2.7. $\square$

Of course, convexity and lemmas 2.1, 2.2, and 2.3 are not the only way to generate divergence functions. For example, the *Jensen–Shannon divergence* is defined as

$$D_{\text{JS}}(P, Q) = \frac{D_{\text{KL}}(P, M) + D_{\text{KL}}(Q, M)}{2},$$

where $M = \frac{P+Q}{2}$.

When P and Q are discrete distributions, it is $\sum_i p_i = \sum_i q_i = 1$. Consequently, the vectors $\sqrt{P} = [\sqrt{p_0}, \sqrt{p_1}, \dots]^T$ and $\sqrt{Q} = [\sqrt{q_0}, \sqrt{q_1}, \dots]^T$ are unit vectors in Euclidean norm. We can then define functions that measure how different those vectors are. For example, the Hellinger distance is the Euclidean distance of the two unit vectors. The *Bhattacharyya coefficient* is their dot product and thus the cosine of the *angle* between $\sqrt{P}$ and $\sqrt{Q}$

$$\text{BC}(P, Q) = \sqrt{P}\sqrt{Q} = \sum_i \sqrt{p_i q_i} = \cos(\angle(\sqrt{P}, \sqrt{Q})), \quad (2.12)$$

where the angle is defined such that $\angle(x, y) \in [-\pi, \pi]$, for all x, y .

*An $n \times n$ symmetric real matrix A is *positive semi-definite* iff $x^T A x \geq 0$ for all $x \in \mathbb{R}^n$. Similarly, an $n \times n$ symmetric real matrix A is *positive definite* iff $x^T A x > 0$ for all $x \neq 0, x \in \mathbb{R}^n$.

Similarly, for continuous probability distributions P and Q with densities p and q , respectively, the Bhattacharyya coefficient can be defined as $\text{BC}(P, Q) = \int \sqrt{p(x)q(x)} dx$. As a result, we can write

$$\text{BC}(P, Q) = \int \sqrt{\frac{dQ}{dP}} dP = \cos(\angle(\sqrt{P}, \sqrt{Q})) \quad (2.13)$$

for both cases.

Note that the Bhattacharyya coefficient has the same mathematical form as an f -divergence but it is not a divergence. When $P = Q$, $\text{BC}(P, Q) = 1$. However, we can easily generate divergences from it. The *Bhattacharyya distance* and the *statistical angle* (a.k.a. *Bhattacharyya angle*) are divergences (the latter is a metric too) derived from the Bhattacharyya coefficient

$$\begin{aligned} D_B(P, Q) &= -\ln \text{BC}(P, Q) \quad \text{and} \\ \Delta(P, Q) &= \cos^{-1} \text{BC}(P, Q) = |\angle(\sqrt{P}, \sqrt{Q})|, \end{aligned} \quad (2.14)$$

where $\cos^{-1}$ is the inverse cosine function with range $[0, \pi]$.

Any divergence or metric can be used as a loss function. However, custom loss functions are also used sometimes. This is common in applications such as image processing where perceived image quality is not measured well with standard losses; or applications such as fraud detection where false negatives are considerably more detrimental than false positives. Appropriate loss functions can vary significantly per application or domain and they often violate desirable properties such continuity, symmetry, or smoothness.

Neural network loss functions should eventually represent multiple learning objectives and be consistent with the fundamental principles of learning. Current loss functions are not derived from a formal definition of learning and they are not as advanced. Instead they treat learning as optimization and only measure how closely a neural model fits a set of data points or how dissimilar two or more entities are. They do not account for the structure of the network either. Most neural networks currently have simple and largely repetitive structure. However, as the networks become more complex, incorporating diverse structures and conflicting dynamics, their loss functions must likewise evolve.

In short, current neural network loss functions are generic statistical tools that inevitably miss essential aspects and principles of neural organization and learning as well significant perceptual and cognitive properties. Yet, they are important building blocks for more sophisticated objectives, constituents of a higher level of complexity that is necessary for learning and intelligence. We will gradually incorporate such complexity into our loss functions and conjunctively with the structural sophistication of our networks. Those two elements should evolve in tandem in neural networks; one should not outgrow or outpace the other.

2.6 Computational Limitations

In section 2.4, we regarded learning merely as optimization and used gradient descent to learn the neural network parameters. The loss functions and the data usually induce a highly non-convex optimization problem. On the other hand, gradient descent can converge to a local minimum (if the learning rate is appropriately small) but it is not guaranteed to find a global minimum, if the optimization problem is not convex. For real applications with large data sets and large networks, the number of local minima is often combinatorial and the probability that gradient descent would end up in a global minimum (by mere chance perhaps) is negligible. Furthermore, it can be shown that training a neural network is an NP-complete problem.*

More specifically, the underlying decision problem is called *neural network loading* (NNL) and it asks whether there exists a set of parameters for a neural network N so that N produces the correct output for all samples in a data set S . NNL is known to be NP-complete. This remains true even if we reduce the requirement that N produces the correct answer for two thirds of the training samples. NNL is also NP-complete for trivial networks that have only two hidden units and one output unit, the hidden units compute linear threshold functions of their input, the output unit is linear, and the underlying problem is *binary classification*, that is to map the input to two classes (usually denoted as 0 and 1 or -1 and +1). The only constraint is insignificant for it requires that the data set contains at least n data points, where n is the dimensionality of the input. (In general, data sets have many more samples than dimensions.)

Such limitations have sometimes been used to criticize neural networks or to declare them intractable. Yet, they are not relevant. All learning systems with universal approximation capabilities (such as multi-layer neural networks) contain underlying optimization problems that are hard (NP-complete or harder, which is defined as *NP-hard*). This is not an intrinsic issue or limitation of neural networks but rather a fundamental property of computing. As stated in *Neural Age* [92, chapter 3],

“All universal systems have similar constraints. Any system that is complete within a computation class must satisfy the complexity

*A *decision problem* is a problem that has two outcomes: positive or negative (i.e. any logical statement where true and false indicate positive and negative outcomes, respectively).

NP or *non-deterministic polynomial time* is a complexity class that represents the set of all decision problems that admit proofs for their positive instances and the proofs are verifiable (by a deterministic Turing machine) in computational time that is at most polynomial with respect to the size of the input of the instances. This is equivalent to the set of problems that can be solved in (at most) polynomial time by non-deterministic Turing machines.

NP-complete is subset of NP and contains all decision problems $p \in \text{NP}$ that have the property that every problem in NP can be reduced into p (by a deterministic Turing machine) and in time that is at most polynomial with respect to the size of the reduced problem. Intuitively, NP-complete problems are the hardest problems in NP.

characteristics of the class. There is nothing new or different about neural networks. If we reject them because of such limitations, we may as well reject all of computing. ”

Furthermore,

- “ There is a fundamental trade-off between complexity and provability. Each system has an underlying *hypothesis space*, which is defined as the set of relations the system can potentially represent. As the complexity of the system grows, its hypothesis space grows and provability (the ability to prove theorems about the system) becomes harder or impossible.

Kurt Gödel’s *incompleteness theorem* was published in 1931 and it was the first mathematical result in that direction. It demonstrated that any enumerable and consistent system of axioms cannot prove all true statements of basic arithmetic: there will always be true statements about the natural numbers that are not provable within the system.

In a letter to John von Neumann in 1931, Gödel furthermore described the *undefinability theorem*, which states that sufficiently expressive languages cannot represent their own semantics; in other words, any language capable of expressing the semantics of some other language L must have strictly greater expressive power than L . (The above result is often attributed to Alfred Tarski, who published a proof in 1936; Gödel never published a proof.) A direct consequence of the latter theorem is that the *Church-Turing thesis* implies that every Turing-complete computational framework has undecidable semantics. ”

In the end, we have to make a choice. On one hand, we can choose a simple and computationally constrained paradigm where complexity is significantly limited but all processes are efficiently verifiable or provable. On the other hand, we can choose a universal paradigm where complexity is essentially unbounded but provability is hard or intractable. The latter is the case with the software industry, and virtually all industry, human activity, nature, and life on earth. Quoting from *Neural Age* again:

- “ There is no complex software product that is provable. There are hardly any large software components that are logically verifiable or certainly free of glitches. We can never be certain about those things because software is a universal computation framework. There is a fundamental and critical factor behind those facts. Both nature and humanity have opted for complexity, not provability. The latter is a desirable property but it is often a luxury with a prohibitively high price. ”

Furthermore, and partly as a consequence of the above limitations, learning is not about finding some elusive or computationally prohibitive global optimum or intractable solution but rather about processing localities. We treat it as a strictly local process throughout this theory and we will eventually define it as such when we return to this topic in volume III.

2.7 Historical and Bibliographical Remarks

Plato's dialogue *Cratylus* was published around 360 BCE [273] and it contains the quotation from Heraclitus that "everything changes and nothing remains still." Leibnitz's seminal text on differential calculus was published in 1673 [211]. The term *mathesis universalis* was used by Leibnitz and Descartes for their hypothetical universal science [366].

Diophantus introduced the concept of *παρισότης* (*parisotes*) or *approximate equality* in the 3rd century CE [86]. P. Fermat translated it as *adequality* and devised a related method to calculate extrema of functions [76]. He was probably the first mathematician to introduce characteristics of algebraic expressions near their extrema as a method to find the extrema. The underlying theorem is known today as *Fermat's theorem for extrema* or simply as *Fermat's theorem*.

Gradient descent is a generalization of the above approach proposed by A. Cauchy in 1847 [45]. Stochastic gradient descent appeared much later. The first (and somewhat elaborate) versions were published in the 1950s as *stochastic approximation* and *estimation* [284, 185].

The term machine learning was coined by A. Samuel in 1959 while he was at IBM. It first appeared in a paper about computers learning to play checkers [295].

Simple approaches that treat learning as optimization appeared in the 20th century and they precede most of computer science. For example, *Fisher's linear discriminant* was published in 1936 [110], the *perceptron algorithm* was invented in 1957 [287], and the *delta rule* (a.k.a. *least mean squares method*) appeared in 1960 [365].

Early approaches to learning are described by N. Nilsson in [252] and as pattern recognition in a classic textbook by R. Duda and P. Hart [98].

Gradient descent methods to optimize complex control systems appeared in the 1960s [181, 39]. In 1962, S. Dreyfus used the chain rule and dynamic programming to derive the gradient formulas for numerical solutions to variational problems [93].

Yet, in 1969, it was claimed that such a method could not exist for multi-layer perceptrons. In a deleterious and propagandist book [236], M. Minsky and S. Papert first presented a trivial mathematical fact (that a monotonic function like a perceptron could not model a non-monotonic one like the xor-function) as a discovery. Then they offered an elaborate proof of it – even though there is an obvious and trivial proof that is about one line long

[92, chapter 3]. Without any proof or even supporting evidence, the authors furthermore claimed that multi-layer perceptrons could not be effectively trained. They also published various other reports critical of perceptrons in the following years, sometimes contradicting themselves. Their attacks were part of an orchestrated campaign to discredit the field and divert funding into artificial intelligence.

Those efforts, combined with hype and aided by computing and hardware limitations at the time as well as the premature death of F. Rosenblatt in a boating accident in 1971, resulted in a successful campaign. By the mid 1970s, neural network research funding had practically vanished from computer science while funding for artificial intelligence increased dramatically. Those conditions persisted for nearly two decades. The inevitable conclusion was a pseudo-scientific wasteland full of vague formalisms, failed ventures and start-ups, distrust with everything related to intelligence, and diminished research funding. More details about this nefarious affair can be found in [92, chapters 1, 3].

Despite the adverse conditions, neural network research continued. Early gradient descent methods were primarily for optimization. They did not use or reference neural networks. In his master's thesis, S. Linnainmaa applied the chain rule to discrete networks with differentiable nodes [218]. Dreyfus used his method to minimize cost functions with continuous parameters [94]. P. Werbos considered an application to neural networks in his Ph.D. thesis in 1974 [362]. He subsequently created the first generic application of gradient descent to neural networks in 1982 [361].

The method was independently rediscovered and popularized as back-propagation by D. Rumelhart et al. in 1985 [288, 290]. A more detailed version appeared in [289].

A good overview of the history of gradient descent and back-propagation for neural networks appears in [300, section backpropagation] and [301, section 5.5]. Despite its simplicity, and the fact that it is a straightforward application of the chain rule, back-propagation took a long time to become an established method in computer science. It was claimed to be impossible in 1969 and it has generated controversy about who deserves credit for its discovery [303].

Overall, the renaissance of neural networks started in the 1980s with the work for D. Rumelhard and J. McClelland [291, 292]. However, applications remained limited for a number of years. C. Bishop published an informative and thorough textbook on neural networks in 1995 [28]. In 2001, R. Duda, P. Hart, and D. Stork published a second edition of [98] that included neural networks as well as other machine learning paradigms and theoretical results [99]. At the time of its publication, it was probably the most comprehensive textbook for both pattern recognition and machine learning.

Distributional semantics originated in the work of J. Firth in 1930s. Firth drew attention to the context-dependent nature of meaning with his notion of

the “context of the situation”. As he stated in 1935, “the complete meaning of a word is always contextual, and no study of meaning apart from context can be taken seriously” [109, page 37]. In the 1950s, context led to an emphasis in collocational meaning, the explicit introduction of distributional semantics, and the distributional hypothesis – which states that words that occur in the same contexts tend to convey similar meanings [108, 147]. As J. Firth stated at the time, “You shall know a word by the company it keeps.”

In 1991, distributional semantics were handled with computational methods and linear algebra in high dimensional spaces [281]. In 2003, unsupervised learning and the hidden state of a neural network were used to derive word embeddings, which are dense representations of distributional semantics for words [23]. The approach was popularized in 2013 as *word2vec* [235].

Divergence as a formulation of statistical distance appeared at least as early as 1943 [24]. The Bhattacharyya distance appeared in the same publication. The statistical or Bhattacharyya angle was defined in 1946 [25]. The relative entropy or Kullback–Leibler divergence was introduced in 1951 [199]. Huber loss was defined in 1964 [165]. Pseudo-huber loss (with $\delta = 1$) appeared in 1997 [47]. Its most general form appeared in 2003 [149, page 619]. The Bregman divergence was introduced in 1967 [33]. Generalized symmetrical Bregman divergences and lemma 2.3 appeared in 2013 [3].

The definition of learning that is quoted in section 2.2 was given by T. Mitchell in his book *Machine Learning* [238].

The neural network loading problem and the proof of its NP-completeness appeared in 1987 in [178]. The result was extended to trivial networks with only two hidden units in 1992 [30].

Kurt Gödel’s incompleteness theorems were published in 1931 [140] while Tarski’s proof of undefinability was published in 1936 [335].

Modern game theory effectively started as a field in 1928 with the publication of a related paper by J. von Neumann [354] (English translation in [355]) and it reached a foundational landmark with the publication of a book by O. Morgenstern and J. von Neumann in 1944 [243]. Non-cooperative games appeared early in game theory as *zero-sum games*, which are games where one player’s payoff loss is the gain of another.

Adversarial approaches for learning were introduced by J. Schmidhuber in 1991 as predictability minimization (PM) and adversarial curiosity (AC) [298, 299]. However, the terminology was elaborate and sometimes vague, both PM and AC remained largely unnoticed for more than two decades.

Schmidhuber’s approach was rediscovered in the 2010s with some modifications and simplifications and it was popularized as *generative adversarial networks* (GAN) in 2014 [130].

The main difference between PM and GAN is that the former uses two loss functions that are designed to make the networks compete while the latter uses a single loss function and minimax optimization.

In our view, the main idea behind the adversarial approach is the adver-

sarial or contrastive nature of the two learners. The choice of one or two loss functions is a detail of secondary importance. It can potentially make a difference in applications but it does not affect the overall idea and the underlying concepts in any other way.

The main contribution of [130] is not conceptual or architectural but rather practical. It simplified and described adversarial approaches in more standard and constrained terms; it made the material more accessible and current; and it shifted the emphasis to generative applications.

J. Schmidhuber interrupted a conference workshop in 2016 – presented by I. Goodfellow, the main author of the above paper – and disputed the claims of the presenter.

A brief mention and a citation to Schmidhuber’s work on predictability minimization initially appeared in the Wikipedia page of GAN; but those were subsequently removed from the page and remain absent at the time of this writing [374]. We believe that the latter incident can be interpreted as a case of academic censorship.

There is grandeur in this view of life.

— Charles Darwin, *On The Origin of Species*, 1859.

3.1	The Imperative of Biology	65
3.2	Cell Theory	68
3.3	Neuron Theory and Elementary Neural Theory	69
3.4	Energy	72
3.5	Heredity	80
3.6	Homeostasis	81
3.7	Evolution	82
3.8	Conclusion	82
3.9	Historical and Bibliographical Remarks	83
	3.9.1 Attribution	85

Foundational Principles

3.1 The Imperative of Biology

Natural science is a branch of science that studies observable entities or processes in the physical world (*nature*). It is distinguished from abstract, formal or theoretical science, such as mathematics or philosophy, and it is primarily about the description and understanding of objectively measurable natural phenomena. Such knowledge is often expressed as an interpretation, hypothesis or a theory and it is derived from empirical evidence which is generated by observation and experimentation. There is an underlying relationship and an emphasis on both of those components. Empirical data without a theory do not constitute an understanding; and hypotheses or interpretations must be validated empirically and objectively to be regarded as an established or accepted scientific theory.

Biology derives from the Hellenic word βίος (vios, romanized as bios, “life”) and the suffix -λογία (logia, “study or speech”). It is the natural science that studies life and living organisms. It primarily considers the physical structure, chemical processes and molecular interactions, mechanisms and function, development and growth, origin, evolution, and distribution of living organisms. It contains more than thirty branches (including anatomy, ecology, evolutionary biology, genetics, microbiology, neuroscience, zoology and many more) and some inter-disciplinary branches such as biochemistry, biomedicine, biophysics, and biotechnology.

Despite the vast extent and complexity of biology, most of its branches can be unified under five main principles.

Definition 3.1. The foundational principles of biology can be briefly described as follows.

- A. *Cell theory* assumes that the cell is the fundamental unit of life, all life is composed of cells, and all cells derive from other cells.
- B. Energy is required for life and it transfers between living entities to other living entities and their environment.

- C. *Heredity* specifies that DNA is contained in the cells of all living organisms and its genetic information encodes the structure and function of the cells as well as the structure and function of tissue, organs, and organisms at a macroscopic level.
- D. *Homeostasis* is a state of balanced equilibrium between the organism and its environment. It represents a natural resistance to change when the organism is already in optimal conditions and derives from processes and mechanisms of the organism.
- E. *Evolution* is change, or rather learning, in hereditary information of biological entities over time. The information consists of genes or *gene expressions* – which are processes that synthesize gene products such as proteins using information stored in genes. Changes occur primarily by mutation and genetic recombination of the genetic material of the parents. The learning process is usually driven by selection (natural or sexual) or random sampling known as *genetic drift*. Evolution provides diversity at various levels of life – from molecules to species and ecosystems – and it is a fundamental and defining process of life.

In a sense, the above principles are the axioms of biology and provide a necessary level of discipline in the field. In fact, these principles represent five sets of axioms. We will elaborate upon them in various parts of the theory. At this point, we can apply the same axioms to nervous systems and brains.

Nervous systems consist of two types cells: *neurons* are cells that perform computations and process information while *glia* or *glial cells* maintain homeostasis, insulate and protect neurons (through myelin), facilitate neurogenesis and migration, and provide mechanical and metabolic support (such as nourishment and removal of waste products).

Energy transfers between neurons as communication signals that represent information. It also transfers between neurons and glia, which effectively represent the ‘environment’ of the neurons.

Genetic information specifies the overall structure of the brain but local connectivity, plasticity, and growth may originate at a local cellular level – as we shall postulate in this theory. In general, neurons are complex cells that dynamically encode a great deal of structural and architectural information.

Equilibria are essential for the operation of the brain. We also assume that they are essential for learning within the brain. This is a postulated type of balance that is distinct from homeostasis but it can also be treated as a homeostatic extension.

Finally, evolution and learning can be regarded at two distinct scales. At the micro-level, it is learning through parameter estimation, memory formation and knowledge acquisition, local plasticity and reorganization, and local growth. At the macro-scale, it is more consistent with the standard definition of evolution and it corresponds to adaptation, structural changes, architectural modifications or (re)arrangement, and overall growth; the latter

are considerably less frequent, they operate on a different time scale, and they can indicate a new species.

We can compare the above with artificial neural networks. Those consist of units that are relatively simple computational elements. Energy comes from a central energy source and it is used exclusively for computations and communication. The units do not encode any structural or architectural information other than pre-defined static connectivity patterns. They are often identical except for the values of their parameters. They do not ‘learn’ anything other than those parameters. The overall structure of each network is also pre-defined and static. The structure is often so plain and repetitive that it can be described by a *regular expression*.^{*} Equilibria are virtually absent except for those that arise from the local optima of a generic loss function. There is no learning other than parameter estimation; and there is no structural evolution other than manual modifications of the architecture that are almost always within the constraints of an original regular structure or variants of it. Even at the highest level and with minimal detail about nervous systems, artificial neural networks represent extreme over-simplifications of learning systems.

Arguably, this should not be surprising. Earth formed about 4.5 billion years ago and life emerged at least 3.7 billion years ago. Nervous tissue first appeared in wormlike organisms about 550 to 600 million years ago. Evolution has been operating for as long and upon billions or trillions of species and with tens or hundreds of millions of animal and plant species.[†] With such a timeline and vast resources, evolution has created an extraordinary level of complexity on Earth. Biological learning systems are highly sophisticated and thus not comparable with current neural networks or machine learning models at any level.

On the other hand, one would expect machine learning research to follow the path of natural sciences, embrace biological complexity, learn from it, and develop theories that describe and explain learning processes and at least some of the observed phenomena. We would also expect that such research would not exclusively focus on abstract mathematical paradigms, primarily from algebra and statistics, that are limited by the short horizon of deductive reasoning and usually lead to over-simplified solutions and systems.

Yet, such a rational scientific approach has not been the case. Instead artificial intelligence promoted a prejudice against neural networks, a super-

^{*}*Regular expressions* are expressions that contain the empty string and all symbols of an alphabet and they are closed under union, concatenation, and the Kleene star operation. They represent basic languages that are called *regular languages*. Those simply contain strings that can be described by a finite set of concatenations, unions, and repetitions of the symbols of the underlying alphabet.

[†]Estimates about the number of species on Earth vary significantly. The largest estimate is that about 1 trillion species currently live on Earth and they represent less than one percent of all species that have ever lived on the planet. Other estimates are considerably smaller. In all cases, the vast majority of species are microorganisms; but most estimates concur that there are about 5 to 10 million extant animal and plant species on earth.

ficial treatment of learning, and hype and reduced discipline. The result is such unremarkable progress that limited recent developments, such as training deeper networks with pre-existing learning algorithms, appear as breakthroughs.

Mathematics are essential in the development of theories for heavily computational domains such as learning. However, they are not as powerful or important as a ‘navigational’ tool. Intuition and understanding are limited to the facts that can be formally proven or conjectured. When we focus on formalisms, we can easily deprive ourselves of vision and foresight. This is particularly acute for domains of extraordinary complexity. For such domains, mathematics should not come first. They can provide indispensable tools and vehicles that can take us to destinations but they are less effective, if not hopeless, in identifying the destinations.

Biology is imperative. It can offer us an extensive record as well as principles, intuitions, and direction that are hard to derive or even imagine by reasoning. When we conduct research on learning and intelligence that ignores biology and the inherent underlying complexities we do not merely adopt a simplistic and naive approach; we commit a folly of remarkable proportions. The overall effect of biology is profound, affecting every aspect of learning. As such, it must be incorporated deliberately and carefully.

Let us now elaborate on the five principles of biology in the context of learning and learning systems and attempt to extract mathematical statements from them.

3.2 Cell Theory

The modern form of cell theory contains six basic tenets. The precise definition is shown below.

Definition 3.2. Cell theory states that

- A₁. The cell is the fundamental structural and functional unit of all organisms and known life forms.*
- A₂. All organisms are made up of one or more cells.
- A₃. All living cells arise from pre-existing cells by cellular division.
- A₄. Cells contain hereditary information (genetic material in the form of DNA and RNA) that is passed to new cells through cellular division.
- A₅. All cells have essentially the same chemical composition in organisms of similar species.

*Viruses are obviously an exception, if we assume them to be life forms. The latter is debatable. Viruses parasitically obtain biochemical aspects of life and they are thus near the border of chemistry and life. They can end up on either side depending on perspective and conventions.

A₆. All energy flow (metabolism and biochemistry) occurs within cells.

M. Schleiden and T. Schwann contributed the first two tenets of cell theory but they erroneously assumed spontaneous generation of cells. Schwann summarized his observations into the following three statements:

A'₁. The cell is the unit of structure, physiology, and organization in living things.

A'₂. The cell retains a dual existence as a distinct entity and a building block in the construction of organisms.

A'₃. Cells form by free-cell formation, similar to the formation of crystals (spontaneous generation).

The correct idea of cell formation by division was promoted by other researchers and was eventually proclaimed by R. Virchow – in his famous Latin dictum “*omnis cellula e cellula*”, all cells arise only from pre-existing cells.

Note that the dual existence of cells is not explicitly stated in the modern version of cell theory. It is rather implicit in it. Cells are the constituent units of life and they can operate individually, possibly centered on their own growth and development, and collectively in the organism serving aggregate or higher purposes. In a sense, each cell leads a double life – one individualistic and the other collective.

We call this the *elemental duality* because it can be applied to individual elements of any complex system.

In learning, we also call it the *learning duality* for it extends from neurons to the computational elements of all learning systems. It is a balancing principle that prevents the elements from becoming overly focused on either individual or aggregate conditions. Without it, learning systems can potentially reduce to degenerate forms: simplistic mechanisms that merely fit data or collections of disoriented computing elements.

In chapter 4, we will see that the learning duality can be a *multiplicity* with multiple collective objectives that correspond to various parts of neural partitions.

Growth is a second implicit assumption in cell theory. It is a direct consequence of the third tenet, the emergence of new cells from pre-existing cells. There is no spontaneous generation or external design. Cells simply generate other cells and the organism thus grows.

In the next section, we will transfer cell theory to nervous systems and form a similar and elementary theory that incorporates the above observations.

3.3 Neuron Theory and Elementary Neural Theory

Definition 3.3. The *neuron doctrine* or *neuron theory* is an extension of the cell theory to the nervous system. It can be summarized as follows.

N₁. The nervous system is made up of discrete cellular units that are called *neurons* or *nerve cells*. Neurons contain specific physiological features such as a cell body, an *axon*, and one or more *dendrites* as shown in figure 3.1. The axon (from Hellenic ἄξων, axis) or *nerve fiber* is a prolonged extension of a neuron that carries action potentials away from the cell body while the dendrites (from Hellenic δένδρον, tree) or *dendrons* are a branched extensions the propagate stimulation received from other cells to the cell body of the neuron. Neurons may have different shape, size, and structure per their location in the nervous system and their functionality.

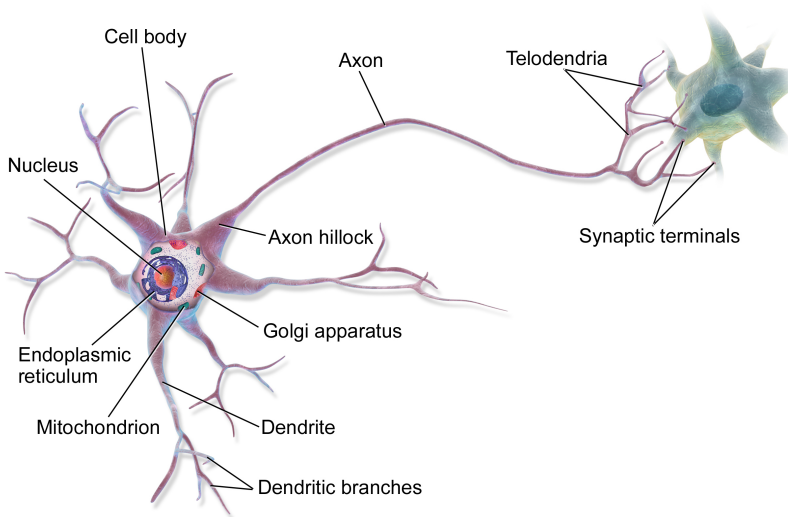


Figure 3.1: Anatomy of a neuron.

N₂. Neurons connect to other neurons through *synapses*. Synapses allow chemical or electrical signals to be passed from one neuron to another or to a target cell. (Axons may synapse onto a cell body, a dendrite, or another axon or its terminal, as well as into the bloodstream or diffusely into adjacent nervous tissue as shown in figure 3.2.) Synapses also represent a barrier that must be overcome for transmission to take place.

N₃. Neurons are generated by cell division from *neural stem cells*.*

N₄. The *nucleus* is the trophic center for the neuron and it is required for its survival.

*A *stem cell* is an undifferentiated or partly differentiated cell that can symmetrically divide into two new stem cells like itself, thus self-renewing, or can divide asymmetrically generating one or two differentiated cells. (*Differentiation* means that the new cell is more specialized in form and function.) Neural stem cells can self-renew indefinitely or differentiate into *progenitor cells* that can either self-renew a limited number of times or produce *macroglia* (which are large glia cells) or neurons.

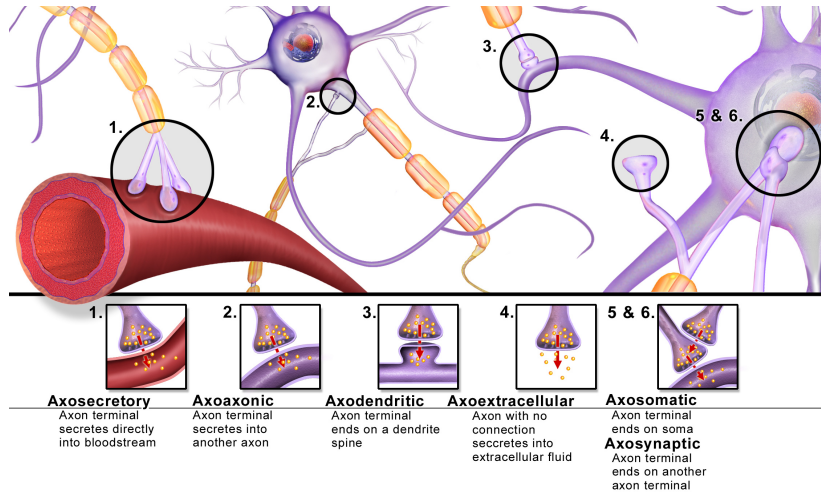


Figure 3.2: Synapse types.

N₅. *Axons* are outgrowths of neurons.

N₆. The axon can conduct in both directions but there is a preferred direction for transmission from cell to cell.

N₇. Each *nerve terminal* (or *axon* or *synaptic terminal*) releases a single type of transmitter.

N₈. A connection between two neurons can be excitatory or inhibitory but it will always be of the same type.

Once more, elemental duality and growth are implicit in the theory.

Arguably some of the principles of cell and neuron theories are specific morphological, physiological, or biochemical aspects of life as it has emerged on earth but they are not necessarily inherent in the underlying mathematical models. They may be more about the ‘implementation’ rather than the concepts or the mathematical structure. For example, we think that N₄ and N₇ may be such principles.

We also contest N₈ as a potential artifact rather than a fundamental principle. The type of a connection (excitatory or inhibitory) should ultimately be decided by learning. Of course, it is unlikely that the type of any given connection will change many times during its lifetime. Connections may thus appear to have largely fixed type in biological experiments of aged nervous systems. However, if a connection has been recently developed with random or ‘guessed’ type, it is not impossible that learning may alter the type. Although it is arguably a form of regulation to fix the type of a connection in advance, we cannot currently make a strong argument in favor of it. Thus we have omitted the principle from our theory.

We can now combine cell theory and neuron theory in a more abstract learning theory, emphasizing some implicit but important properties.

Definition 3.4. The *elementary neural theory* (ENT) asserts the following principles:

- n₁. **Neurons.** *Neurons* or *units* are the fundamental structural and functional elements of all neural systems; and they usually appear in large numbers in such systems. Each neuron is able to compute, process, and store information. Each neuron retains a dual individualistic and collective existence – as a distinct and independent entity as well as a constituent element and building block of the system. Explicit functional forms can allow each neuron to find a balance in its duality.
- n₂. **Connections.** Neurons should contain structures that enable connections and thus communication with other neurons. Each neuron should be able to grow new connections or prune existing ones over time. Connections could recursively be organized in structures or partitions of various functional types; neurons should be able to connect to such structures, too. Some neurons should be able to receive input from the environment or a host system (*input neurors*) and others should be able to generate output for the environment or host system (*output neurors*). All connections should be directional and as such define the *flow of computation* through the system. However, certain information should be allowed to move in the opposite direction (e.g. gradients for learning purposes).
- n₃. **Growth.** All neurons should arise from preexisting neurons or specific neural forms that we call *stem neurons*, or simply *stems*. The generation could be by division or other neuro-generative mechanisms (to be specified separately). All learning systems should grow from initial seed states; their growth – and reduction – should be inherent. Although growth may be more dramatic in the early stages of a learning system, it should remain as a main source of complexity and architectural sophistication in the system.
- n₄. **Computing.** All neurons should essentially share the same computational model albeit with considerable variation in their functional elements, overall functionality, connectivity, size, and structure. Such variation should derive not only from growth mechanisms and learning but also from the stems in a hereditary and evolutionary manner.
- n₅. **Information.** All information (about computations, learning, growth etc.) should be managed by the neurons.

3.4 Energy

There is a fundamental relationship between energy and information and it derives from thermodynamics. The second principle of biology asserts energy

as a prerequisite of life. This is almost a trivial statement given that life processes, stores, and retrieves information. However, there is an important implication: if life requires energy then different life forms will probably require different amounts of energy and some will inevitably use energy more efficiently. As a result, and when combined with the fifth principle (evolution), the second principle can only point in one direction.

Evolutionary pressure improves fitness for survival and thus leads to organisms that tend to use energy more efficiently. Organisms that can sustain functionality with lower energy consumption will have an evolutionary advantage. Functionality can also be reduced or altered if it significantly reduces energy expenditure without a significant reduction in the probability of survival.

The above process applies to every level of life from the body and organs to individual cells. It can affect not only morphology and physiology but also functionality and metabolism. The nervous system is no exception.

The purpose of the nervous system is relatively simple. It is meant to regulate an organism's internal milieu and help the organism survive and reproduce. Sophisticated nervous systems enabled animals to develop complex behaviors and survive in changing or challenging environments.

Such survivability could justify higher energy consumption. Indeed, energy consumption in neurons is relatively high compared to other *somatic cells* (cells of living organisms that are not reproductive cells). For example, the human brain only accounts for about 2% of total body mass but its metabolic consumption represents more than 20% the body's total energy production in resting state; for newborn infants, this percentage can reach 50%.

On the other hand, the brain only operates on about 20 watts.* This is the wattage of a small electric bulb. In comparison, modern supercomputers that achieve petascale computing (that is 10^{15} operations per second) usually require mega watts (MW) of power. Therefore, our brains are roughly 10^5 times more energy efficient than supercomputers.

For example, in November 2019, the *IBM Summit supercomputer* was the fastest supercomputer in the world with 200 petaFLOPS peak performance and 148.6 petaFLOPS on the Linpack benchmark.† It was the 5th most energy efficient supercomputer with 14.7 gigaFLOPS per watt on the benchmark. This corresponds to about 10.1 MW of power – its specifications state that it requires 13 MW. At the time, the most energy efficient supercomputer was the *Fujitsu A64FX prototype* with 16.9 gigaFlops/watt. It is a much smaller supercomputer with about 2 petaFLOPS performance on Linpack and power consumption at 0.1 MW.

Such comparisons cannot be precise, of course. The fundamentals of

*As a rough estimate, 2000 daily food calorie intake corresponds to 2000 kcal/day or 23.18 cal/sec; the latter is 96.85 W and 20% of that is 19.37 W.

†*Linpack* is a linear algebra benchmark. *PetaFLOPS* and *gigaFLOPS* represent 10^{15} and 10^9 floating point operations per second, respectively.

computers are dramatically different than those of nervous systems and there are many factors (such as protein or molecular processing) that are hard to take into account. Precision is not necessarily at the same level and it may be variable in the brain. There are no brain benchmarks either. Overall, the estimates can be more than one order of magnitude off; but we feel that they are largely conservative. In either case, we will attempt the comparison in order to get a rough idea of the differences.

Given that the brain contains about 86 billion neurons and each has about 10,000 connections on average, if we assume that each neuron fires f times per second on average and each synapse corresponds to 1 floating point operation (which is also a conservative estimate given the complexity of synapses and an apparent threshold and weight operation in each of them), the overall rating of the brain is $43,000f$ gigaFLOPS/watt.

The precise value of f is debatable. It is roughly estimated to be between 0.29 and 1000.* (This also implies a large range between average and peak performance in the brain.) In terms of FLOPS/watt, and given the above estimates, the ratio between the human brain and the IBM summit supercomputer, is between 848 and 2,925,170. The geometric mean of the above range is about 49,813 – which is consistent with our initial bulk estimate of 10^5 .

Irrespective of the exact ratios (and whether they can be estimated in a precise way or not), it is clear that the brain is orders of magnitude more energy efficient than digital (super)computers. This raises an obvious question. Why is there such a huge discrepancy? There are two sets of reasons, one has to do with computer architectures and the other with biology.

In general, von Neumann architectures have low thermodynamic efficiency. For each bit of processed information in the processing units of such architectures, the amount of energy dissipated as heat is relatively high. The processing units can actually heat up so quickly that they often require cooling systems to prevent them from overheating and burning; and the cooling systems consume additional energy aggravating the thermodynamic deficiency.

Modern computer architectures are considerably more complex in their treatment of memory (storage) than the abstract and simplistic idea of von Neumann architectures. For example, they do not define memory as a single or indivisible block; they use a memory hierarchy with on-chip registers, register windows, caches, multiple modules of random access memory, solid state drives and disks or redundant arrays of independent disks. However, all architectures that accept the von Neumann convention and create a dichotomy between memory and processing are bound to be limited and energy inefficient no matter how sophisticated they are.

*The *absolute refractory period* of a neuron is the minimum time between two successive firings and it is estimated to be 1-2ms. The *relative refractory time* is the minimum time between two successive firings that requires especially large stimulus for the second firing and it is estimated to be 3-4ms. Thus the maximum firing rate is between 250 and 1000. However, energy based estimates place f close to 1.

The inefficiencies are particularly acute for computing paradigms such as neural networks that are strictly local and have predefined, static, and linear memory access patterns.* Such memory belongs to the corresponding neurons or units. Placing it at a distance, sometimes in a different chip or device, or in random access memory, and accessing it through a bus or some other shared pathway, is computationally and thermodynamically wasteful.

The thermodynamic inefficiency of current architectures is indeed so high that the energy consumed to train large neural networks has a notable CO₂ emission effect. For example, one of the largest neural networks is a big transformer network that has been used in *neural architecture search* for machine translation and natural language processing. Training amounts to a total of 979 million steps and the base model requires 10 hours to train for 300k steps on one TPUv2 core. This equivalent to 32,623 hours of TPU or 274,120 hours on 8 P100 GPUs. It has been estimated that the above corresponds to 626,155 lbs of CO₂ emissions while the average emissions of a car for their entire lifetime is 126,000 lbs.

Memory is not the only source of above inefficiency. There are the limitations of digital processing and current computing paradigms. Without error checking or error correction codes, a single faulty bit can make an entire computer system crash. A single division by zero or a single numerical uncertainty (such as a *Nan*, which stands for “Not a Number”) can make an entire neural network fail when it is implemented on a computer.

In general, there is little tolerance for noise in the predominant computing paradigms in computer science; but operating under large *signal-to-noise ratio* (SNR)[†] requires greater amplitudes – to ensure that errors are extremely rare – and thus quadratically more energy. For example, a factor of 5 in the amplitude consumes 25 times more energy.

Overall, there is a trade-off between analog and digital representations in computing and communication. Analog representations are computationally and thermodynamically more efficient but they are more sensitive to noise. Their power consumption increases with noise – in order to maintain information – and they inevitably become less efficient after a certain level. Noise tends to accumulate over multiple computations or when signals travel larger distances. (Insulation can help.) Wires can have different width depending

*Plasticity is a complicating factor, of course, but memory access patterns change only when new connections or neurons are created or removed.

[†]For a signal S and noise N , the signal-to-noise ratio is defined as

$$\text{SNR}_2(S, N) = \frac{P_S}{P_N} = \frac{ES^2}{EN^2}. \quad (3.1)$$

When $S \geq 0$ and $EN = 0$, an alternative (but not equivalent) definition is

$$\text{SNR}_1(S, N) = \frac{ES}{\sigma_N}, \quad (3.2)$$

where σ_X indicates the standard deviation of a random variable X .

on their length and the importance of their communicated information.

Therefore, an architecture that is thermodynamically efficient must take noise and communication distances into account and use both analog and digital processing. It should have some insulation over communication wires and most likely a distribution of wire widths. It should allow for computations under low SNR, prefer computation models that are more resilient to noise, and instead of exclusively resorting to high amplitudes, consider redundancy as a remedy for low SNR.*

Current computer architectures are fully digital. There is hardly any insulation for communication wires even when those are long busses that run across a chip. Wire shapes are limited and generic. They are based on electric and geometric layout constraints but not on transmitted information or its importance. Furthermore, all information is treated as equally important. Computations with low SNR are not possible; and SNR is always kept high with large amplitudes. In the end, all those factors and design choices have one common effect. They reduce thermodynamic efficiency.

There are many other simplifications – such as constant precision and bandwidth, fixed coding, fixed wiring, absence of growth and adaptability – that affect energy consumption. Up to this point, the emphasis in computer design has mainly been on increased performance for specific applications in various areas of computer science; noise and energy consumption have been largely ignored or treated as less significant. Although energy efficiency is becoming increasingly more important with mobile applications, it is still at an early stage.

Let us now compare the above and rather simplistic state in computer architectures with the extraordinary energy efficiency and sophistication of nervous systems and brains. The following are important facts and key differences.

Hybrid processing. Nervous systems use chemical and electrical signaling within and between neurons. Chemical computing is direct, it has low thermodynamic cost, and it affords fast signals at short distances – but not for longer distances. Analog (graded) signals are used in neurons but pulsatile signals (electrical impulses that are called *action*

*Given a signal S , we can take the average of n different noisy versions $S_i = S + N_i$ for $i = 1, \dots, n$:

$$Y = \frac{1}{n} \sum_{i=1}^n S_i = S + \frac{1}{n} \sum_{i=1}^n N_i = S + N.$$

The SNR of the average is improved n or $\sqrt{n}$ times, depending on the definition:

$$\begin{aligned} \text{SNR}_2(Y, N) &= \frac{ES^2}{EN^2} = n \text{SNR}_2(S, N_1) \quad \text{and} \\ \text{SNR}_1(Y, N) &= \frac{ES}{\sigma_N} = \sqrt{n} \text{SNR}_1(S, N_1), \end{aligned}$$

assuming that the different versions of the noise N_i for $i = 1, \dots, n$ have zero mean and they are *independent and identically distributed (i.i.d.)*.

potentials) are used to transmit information across neural axons. Thus energy efficient analog computing is used locally in each neuron but noise-resistant pulsatile communication is used across neurons.

Fast communication. Action potentials allow signals to be propagated rapidly over relatively long distances. They have a higher tolerance for noise and they tend to prevent noise accumulation in cascaded transmissions. However, they incur a high thermodynamic cost. Various studies have concluded that most of the energy in neurons is consumed by action potentials (about 47%) and post-synaptic effects (about 34%). In other words, about 80% of the energy is consumed for the movement of ions and fast robust communication between neurons.

Signaling energy diversity. Total brain energy consumption seems to increase proportionally with the number of neurons across species but the signaling energy per neuron appears constant in different mammalian species. On the other hand, mammalian action potentials consume far less energy than the squid giant axon action potential. Moreover, computational modelling has revealed considerable differences in the energy consumption of different action potentials. Although the main factor has been identified as the overlap between inward and outward currents, the rationale behind such differences is not clear. However, and given that action potentials consume almost half of the energy, signal-based differentiation and diversity in wire designs can significantly affect energy consumption.

Sensitivity to energy. Neurons are sensitive to energy limitations as indicated by their strictly aerobic metabolism.

Uncertainty. The nervous system operates under considerable uncertainty: ligands bind, ion channels open, vesicles fuse, and spikes transmit, all of them probabilistically and without any evident synchronization. The underlying computational model must thus be probabilistically robust and resilient to noise.

Efficient coding. Energy consumption has a significant effect in neural coding and the overall coding of information in the brain. For example, energy costs incur even when a neuron is ‘at rest’ just to maintain its state and resting potential. However, resting costs are often significantly lower. Low activation requires less energy. Thus *sparse coding* – which a representation of items where only a small subset of the involved neurons generate strong activations – has a metabolic advantage. It is also more consistent with the statistics of stimuli, contiguity or continuity in the environment, and representations of increased specificity in later stages of neural processing. Not surprisingly, there is considerable evidence of sparse coding in the brain.

Redundancy. Redundancy (such as employing multiple neurons and averaging their outputs) can increase SNR indefinitely. However, there is an optimal range. As redundancy grows, fidelity gains would not justify their costs.

Neuron size. Size affects energy consumption in neurons. Small neurons have lower membrane capacitance but also less membrane area for ion channels and receptors and less volume for *mitochondria* – which are bound organelles and a main source of chemical energy for the cells through their supply of *adenosine triphosphate* (ATP). Thus, small neurons suffer from low bandwidth and increased noise and they operate at low information rates; but they consume less energy and they can be present in larger (redundant) numbers.

Axon diameter size. In order for an axon to double its information rate, it must more than double its firing rate and thus more than quadruple its volume (firing rate correlates linearly to diameter). If mitochondria volume grows likewise, as is the case in the optic nerve, energy capacity and arguably expenditure will grow proportionally with volume. Consequently, if energy optimization is a significant factor, transmission of information will be at or near the lowest possible rates. Thicker axons will thus be used only for information that requires greater rates. Not surprising, there is a broad distribution of axon diameters in nervous systems, differing 100-fold, from about 0.1 to 10 μm . Energy differences can thus be 10,000-fold. Thin axons are considerably more common and they typically have lower firing rates.

Synapse size. Experimental and computational studies have demonstrated that synaptic input size affects information rates in neurons. Larger synaptic inputs increase signal amplitude, improve SNR, and produce higher information rates; but they consume more energy.

Axon insulation and propagation. *Myelin* is a substance that surrounds and insulates nerve cell axons and increases the propagation speed of action potentials. In unmyelinated fibers, action potentials propagate as continuous waves at about 0.5 – 10 m/s. In myelinated fibers, sections of myelin sheath are separated by short gaps that are called *nodes of Ranvier* (see figure 3.3). As sodium ions enter a node, their electrical field pushes the ions of the axon toward the next node, the electrical signal rapidly arrives at the next node and creates another action potential that refreshes the signal. This process is known as *saltatory conduction* and makes the action potentials effectively hop from one node to the next speeding up propagation to about 150 m/s. Myelin also decreases capacitance and increases electrical resistance across the axon. Overall, it enhances signal propagation and enables longer communication distances.

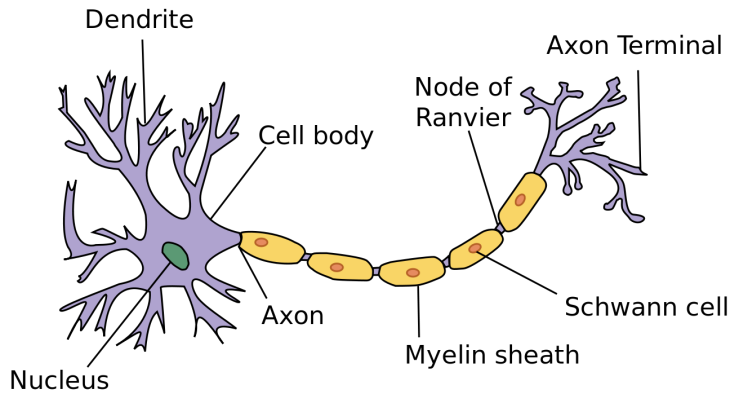


Figure 3.3: Structure of a neurons and its axon.

Wire economy. Energy consumption for signal transmission along wires (dendrites or axons) depends on the length of the wires. Studies have shown that the positions of neurons in insect neural networks are consistent with minimized wire length when the volume of the related components is taken into account. Length can be reduced by locality and neural partitioning (such as maps, circuits, layers, columns, stripes and neighborhoods). Overall, the placement of neurons within neural networks or regions within the cortex seems to relate to wire length and thus energy consumption.

Energy vs. information. Maximal information rates correspond to disproportionately high energy consumption. Increased signaling speed (bandwidth) implies greater conductance of the membrane and thus increased energy consumption. Communication at low rates and with a constraining energy budget is more common.

Learning and adaptation. Nervous systems use synapse thresholds and other mechanisms to reduce transmitted information, match channel capacity to required bandwidths, adapt their I/O curves to maintain efficiency under changing conditions, store important information and statistics, and forget some information in order to improve storage capacity and energy expenditure. There are various levels of learning and adaptation. Synapses can adapt in milliseconds or minutes by altering vesicles or receptor proteins. Neurons can adapt in minutes, days or years by changing protein circuits, synthesizing new postsynaptic receptors in the membrane, or growing new synapses or pruning existing ones. New neurons can be generated from neural stem cells. Production of new tissue consumes substantial amounts of energy. Even the simplest synthetic processes for cellular maintenance (such as protein or lipid synthesis) consume non-trivial amounts of energy. Although adaptation and learning are not driven exclusively by energy, consumption and

metabolic efficiency are important factors that are firmly established by evolutionary pressure. In general, energy consumption seems to be a significant factor for growth and plasticity, connectivity, structure and architecture, learning and adaptation, and ultimately behavior.

The above factors only emphasize the increased complexity of nervous systems. More importantly, they specify fundamental designs and principles. Insects appeared about 479 million years ago; mammals about 178 million years. Their brain designs had ample evolutionary opportunity to diverge but they seem to have followed the same rules and patterns. Persistence across such scale and diversity is unlikely to be a random or incidental outcome. We assume that it is rather the result of fundamental underlying principles.

It is precisely those principles and their inherent complexity and efficiency that render current computer architectures simplistic and largely irrelevant for neural computing. They explain the dramatic differences in thermodynamic performance and energy consumption. Systems based on main processing units (such as CPUs, GPUs, or TPUs), random access memories, and related software frameworks are not a viable solution for neural networks of biological scale – especially when those networks must operate locally on home or personal devices.

Neuromorphic computing refers to analog, digital, or hybrid analog and digital VLSI and software systems for the implementation of neural networks. Undoubtedly, it is a more efficient approach. There is currently increased interest in the field. However, there has been a considerable degree of hype and misguided research. Results are mixed and there has not been a clear winner. However, we expect the situation to change in the coming years or decades. Optical solutions are also a possibility but they are still at an early stage.

We will postpone further discussion on energy, hardware, and frameworks until volume III of the theory. However, we should keep in mind that energy is an inevitable quantity and an indispensable parameter that affects not only the overall design and performance but also the underlying mathematical theory and its foundational components.

3.5 Heredity

At an abstract mathematical level, *heredity* or *inheritance* is the transfer of traits or properties within a hierarchy. In nature, genetic information is encoded in the DNA inside the cells of living organisms and it is passed on to offspring from their parents. It enables generational autonomy and self-sufficiency and it is thus essential in all systems that can self-generate from a seed state. It is as essential for adaptation and evolution, too.

The generation process may be deterministic at a microscopic level but it is stochastic at higher levels. During its development, the system interacts

with its environment, drawing energy from it and processing stimuli. Those interactions affect form and growth. For example, monozygotic (identical) twins do not have the same fingerprints because the fetuses touch different parts of their environment in the womb and thus develop variations in their prints that are unique. Similarly, two monozygotic twins may have the same genes for height, but they may end up with different height if they do not receive the same amount of nutrients while in the womb. Furthermore, monozygotic twins may be genetically similar but they are usually not genetically identical. Mutations during the embryonic period or during development can generate differences. The earlier such changes occur, the more cells that may be affected. Finally, there is potential *epigenetic modification*, which refers to heritable changes that do not involve changes in the DNA sequence.

The above factors apply to the nervous system, too. Neural growth derives from three components: the first is hereditary, the second is environmental, and the third is mutational or stochastic. Heredity provides the neural architecture but variations of the overall form, structure, and connectivity as well as various other details and functions of the system are largely environmental and stochastic.

From a computing perspective, a neuron can be seen as an *object** that not only contains its data and functionality but also information that allows the neuron to generate its entire nervous system from scratch (given a proper environment and sufficient resources). This completely removes the need for external developers and it is thus crucial for self-sufficiency.

We will not consider self-sufficiency and heredity any further for the rest of volume I of this theory. We will use neurons or units that have zero hereditary information and have preallocated positions in predefined neural networks. However, we will gradually introduce hereditary information later in the theory as well as an algebra to describe such information and grow the networks from neural stems.

3.6 Homeostasis

In section 3.1, we defined homeostasis as “a natural resistance against change when the organism is already in optimal conditions.” In its most limited form, homeostasis may appear as a property of stability around certain variables or conditions that are deemed normal or optimal – such as temperature, blood pressure, heart rate etc. It may thus be viewed as important or essential for the proper function of a system or an organism but otherwise irrelevant for learning.

* *Object-oriented programming* is a programming paradigm where data are organized as a collection of objects that contain not only the data themselves but also the functions that operate on the data. The functions are associated with the data and are often bound to the data instead of being defined independently from them. In a sense, data and functionality grow together while an object-oriented system is developed.

However, homeostasis can also be seen as a more general tendency to retain information and state that is associated with a current equilibrium and resist change and transfer to a different state or equilibrium when conditions change. In this expanded consideration, homeostasis has a memory role. It represents resistance to ‘forgetting’ previous information. It exists not only to maintain state but also to combat interference between different equilibria or learning states.

Homeostasis as a learning principle is one of the least developed segments of this theory. It is work in progress. Our methods are currently at an early stage and they are rather simplistic or rudimentary. Depending on progress, we may include it in volume III of this theory or leave it as a principle and open research problem to be explored by future extensions of the theory.

3.7 Evolution

Evolution is learning that occurs at the level of species and ecosystems or more abstractly in a hierarchy of species or systems. It is a specific form of learning that, at any given time, selects among a finite set of available alternatives based on criteria about survival (natural selection) and reproduction (sexual selection).

Evolution can apply to every heritable characteristic and, of course, it applies to the nervous system. It can affect every aspect of it including its size, structure, and function. More importantly, if a learning system must be self-sufficient and able to grow from a seed state, it must necessarily start with smaller or trivial structures that have limited functionality and evolve into more complex and sophisticated forms.

In this theory, we will initially ignore evolution and start with specific neural structures that are based on evidence and results from machine learning, mathematics, biology, and neuroscience. In a sense, we will jump start the process, avoiding the early evolutionary stage that operated on tiny or small nervous systems. However, as we develop the theory further, we will gradually introduce evolutionary elements and selection criteria and propose experiments and methods that might generate the same or similar neural structures from scratch. By definition, evolutionary processes cannot precede the introduction of hereditary information. They depend on the underlying algebra, and they will be defined and developed in parallel.

3.8 Conclusion

At this point, it should be clear that neural networks and learning – as defined in chapters 1 and 2 as well as in machine learning literature – are simplistic constructs. Although they constitute a legitimate exploratory approach, they cannot support complex cognitive functions as well as various essential processes of life and intelligence.

The elementary neural theory, energy efficiency, homeostatic processing, heredity, and evolution are not incidental properties of cells, neurons, or nervous systems. They are indispensable principles of learning – and life too. Without them learning is not merely less, it is not learning at all.

Optimization is certainly an important part of the process, and it occurs in various forms – from energy and homeostasis to evolution – but it is strictly a constituent. The reduction of learning to the level of optimization (as in section 2.4 or in machine learning literature) is untenable and largely the result of disciplinary separation and an educational deficit.

In the rest of this theory, we attempt to correct the situation and close or reduce the gap. We will amend and redefine methods, structures, and systems, and we will gradually introduce specific instances of the above principles as well as additional ones that have intentionally been omitted until we have laid the necessary groundwork.

3.9 Historical and Bibliographical Remarks

The term biology appeared in various European texts in the years between 1736 and 1802 but the term βιολογία did not exist in ancient Hellenic. An overview of the field and its history can be found in wikipedia or biology textbooks [368, 353]. Descriptions of the various branches of biology can also be found in [369, 257].

Estimates about the origins of life on earth and the number of species appear in [266, 219, 372]. The estimate for trillions of species appears in [219] but there is an associated letter that disputes the validity of the approach as well as a reply of the authors that defends the approach.

Bilaterian animals represent the great majority of existing species and they have presumably descended from a common wormlike ancestor that had nervous tissue and appeared in the Ediacaran period 550–600 million years ago [259, 18].

An overview of nervous systems and their cells appears in wikipedia [375] and in [119].

M. Schleiden, T. Schwann and R. Virchow are often credited with the original formulation of cell theory. Their theory, however, was a generalization of previous research and ideas by L. Oken, C. Mirbel, J. Lamarck, H. Dutrochet, and P. Turpin. See [312] for more details.

The neuron theory or neuron doctrine was initially proposed by S. Ramon y Cajal. The term was introduced by H. Waldeyer-Hartz based on research by C. Colgi, F. Nissl, S. Ramon y Cajal, A. Forel and others [313, 221]. The development of a silver staining technique known as *black reaction* or more commonly as *Colgi stain* by C. Colgi in 1873 allowed nervous cells with their axons and dendrites to be visualized. C. Colgi supported the *reticular theory* which claimed the nervous system to be a single and contiguous network. However, S. Ramon y Cabal began working on nervous systems using the

Colgi stain in 1887 and, the following year, he reported that nerve cells were not contiguous in the brain of birds. The discovery led to the formation and acceptance of the neuron theory but C. Colgi maintained his support for the reticular theory. The two researchers were jointly awarded the 1906 Nobel Prize for Physiology or Medicine but the controversy between the two theories was not resolved until the 1950s with the development of the electron microscope.

The concept of the regulation and stability of internal state in organisms was introduced by physiologist C. Bernard in 1849 as *milieu intérieur* [138]. The related concept of homeostasis was coined by W.Cannon in 1926 [43].

Regular languages were first described by S. Kleene in 1951 using *regular events* which were a mathematical notation similar to regular expressions [189].

An overview of the IBM Summit supercomputer appears in [381] while a review of supercomputer performance per watt appears in [1, 379]. Estimates for neuron refractory periods are given in [270]. Estimates on the average firing rates of neurons in the human brain are presented in [169]. A wide range of estimates of the FLOPS rating of the human brain are referenced in [168].

The training of a deep transformer network for neural architecture search is described in [319]. Estimates of energy consumption and CO₂ emissions of large deep neural networks appear in [331].

Principles of Neural Design [326] is a great textbook, an analytical and compositional source of neural design principles. Our argument (in section 3.4) about the emergence of principles from the biological record is directly derived from the book:

“Despite half a billion years of evolutionary opportunity to diverge, brain designs in insects and mammals seem to have followed the same rules.

For designs to have persisted across this immensity of time and spatial scale implies that they are neither arbitrary nor accidental. Rather, they must have emerged as responses to some broad constraint. That is what elevates the shared responses to the status of principles.

”

Estimates about the energy consumption in the brain appear in [184, 320, 286, 15]. Reviews of energy consumption in neurons and nervous systems are presented in [253, 15]. A study of axon diameter distribution in 16 fiber groups across five species (guinea pig, rat, monkey, locust, octopus) is in [268]. Energy consumption in a homeostatic context is discussed in [350]. A tutorial on energy production in the brain is available online [249]. A discussion about analog and digital trade-offs and energy consumption in neural computing appears in [308]. In [215], it has been argued that many aspects of neural anatomy and physiology can be explained as “a compromise between the rate of information processing and the energy consumed.”

Evidence that total brain energy consumption increases proportionally with the number of neurons across species is given in [155]. Furthermore, in [166], evidence is presented that the signaling energy per neuron is constant across several mammalian species.

An overview of sparse coding appears in [256] along with a demonstration that sparse coding of natural images produces oriented filters that are similar to the receptive fields of simple cells in the visual cortex. Sparse codes tend to represent a compromise between dense and local codes and they have considerable advantages [121].

Myelin insulates axons and allows for faster propagation of action potentials through saltatory conduction [378].

Neuromorphic engineering is an alternative term for neuromorphic computing and it was introduced by C. Mead in the late 1980s (published in 1990 [230]).

A study that demonstrates the stochastic nature of heredity appears in [216]. It contains an analysis of mutations in the white blood cells of 66 pairs of monozygotic twins.

3.9.1 Attribution

The following are copyright attributions.

The images in figures 3.1 and 3.2 are from [325]. The image in figure 3.3 is by Quasar Jarosz at English Wikipedia and licensed under the Creative Commons Attribution-Share Alike 3.0 Unported license (<https://creativecommons.org/licenses/by-sa/3.0/deed.en>).

πάντων γὰρ ὅσα πλείω μέρη ἔχει καὶ μὴ ἐστὶν οἷον σωρὸς τὸ πᾶν ἀλλ' ἐστὶ τι
τὸ ὅλον παρὰ τὰ μέρη, ἐστὶ τι αἷτιον

(for all things which have multiple parts, and which are not merely a pile but
a whole beyond the parts, there is a cause).

— Aristotle, *Metaphysics*, 8.6, 1045a, c. 360 BCE.

4.1	Structure	87
4.2	Synergy	89
4.3	Graph Cuts	91
4.4	Neural State and Synergy	93
4.5	A Neural Algebra	96
	4.5.1 Neural Output	97
	4.5.2 Neural Input	98
	4.5.3 Neural Parallelism	98
	4.5.4 Neural Repetitions	99
	4.5.5 Neural Equivalence	99
	4.5.6 Neural Synergy	100
	4.5.6.a Parallel Neural Synergy	100
	4.5.6.b Neural Rigidity	101
	4.5.6.c Sequential Neural Synergy	103
	4.5.6.d Neural State Selection	103
	4.5.7 An Algebra for Synthesis	104
	4.5.8 Is an Algebra Necessary?	106
4.6	Cooperative Learning	107
	4.6.1 Homologous and Heterologous Learning	109
4.7	Cooperative Game Theory	112
4.8	Examples of Synergetic Learning	113
	4.8.1 Ensembles	113
	4.8.2 Generative Networks	116
4.9	Diversity	119
	4.9.1 Voting and Condorcet's Theorem	122
	4.9.2 A Theoretical Justification for Diversity	124
4.10	The Synergy Conjecture	127
4.11	Extrinsic Diversity	129
4.12	Definitions of Diversity	132
4.13	Diversity Algorithms	136
4.14	Conclusion	139
4.15	Historical and Bibliographical Remarks	139

4

Synthesis

4.1 Structure

Complex systems are not atomic monoliths. They contain multiple parts. The parts may contain other parts which, in turn, may contain subparts and so on down to a basic building block. This is reflected in cell theory's principle that the cell is the fundamental structural and functional unit of life – and in the corresponding principles of neuron theory and the elementary neural theory that define the neuron as the basic unit.

Furthermore, complex systems do not consist of identical units. They are not unstructured or incidental piles of their units either. On the contrary, they contain a multitude of differentiated units and various levels of structure and organization.

For example, the human body contains approximately 37.2 trillion cells. Those cells have descended from a single cell but they are highly differentiated. They form different tissues, organs, and systems. The exact number is debatable but there are at least 78 organs in the human body and 11 systems. They perform various functions (such as perception, movement, breathing, digestion, metabolism, reproduction, etc.) that are essential for the existence of the species in the ecosystem.

Similarly, the human brain contains about 86 billion neurons and many parts. Anatomically, it is usually divided into three parts: the *cerebrum*, *cerebellum*, and *brainstem*. The cerebrum is the largest part of the brain (by volume) and it consists of the right and left hemispheres. The hemispheres are joined by a structure of fibers that is called the *corpus callosum*. Each hemisphere is also divided by fissures into four lobes: the *frontal*, *parietal*, *occipital*, and *temporal lobes*. Each lobe, in turn, is divided into areas that perform specific functions as shown in figure 4.1.

There are more parts and further structure and organization in the brain. For example, there is a 'horizontal' partitioning into layers. The *cerebral cortex* is the outer layer of the cerebrum and it contains the *neocortex* and the *allocortex*. The neocortex consists of six layers that are segregated by

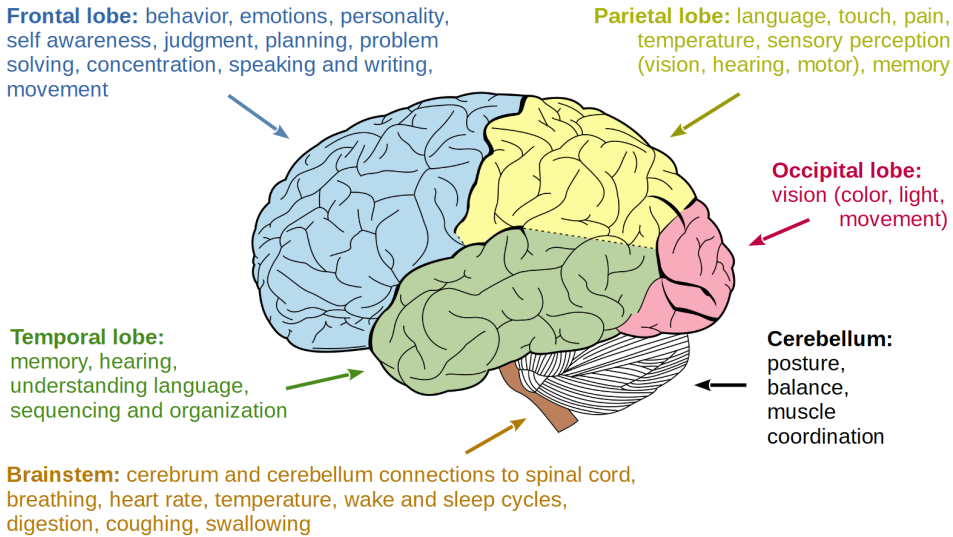


Figure 4.1: Human brain structure and function.

connectivity and cell type. The allocortex has three or four layers. Partitioning may continue to a finer level down to small sets of neurons and in ways that may not be anatomically visible.

In short, there is great degree of structure and organization in the brain. It is not a pile of neurons but rather a sophisticated whole that is clearly beyond its parts. As Aristotle observed in his *Metaphysics*, there is always a cause for such a synthesis.

To some extent, we can explain structure as an outcome of variation and composition. If a system is to perform multiple tasks, it may contain multiple components, one per task or per set of similar tasks. Those components form a partition of the system and initiate structure in it.

However, this is only a partial explanation. Structure does not appear only at high levels of functionality but rather in all levels and all parts of a system down to its elementary units. The units themselves are often differentiated and arranged. As a phenomenon in life, structure is ubiquitous because it is an inevitable consequence of the foundational principles of learning and biology. It derives from cell theory and energy and it is augmented and improved by growth and evolution.

The nervous system is a prime example. Neural structure and complexity are inextricably intertwined with functionality, energy consumption, growth, and evolution. The same probably applies to all neural systems that are capable of complex cognitive tasks. We cannot possibly understand such systems or reproduce their functionality and properties if we ignore their structure. Approaches that treat neural systems or their parts as unstructured arrays or homogeneous collections of neurons and random variables as well as approaches that try to design such systems based on similar arrays or

collections are evolutionarily primitive and ineffective. Complex learning systems are not fixed walls of carefully arranged bricks. They are structurally more similar to forests of diverse flora growing in various and rather unique forms, shapes, and sizes.

At a more abstract level, we can understand the necessity of structure from a top-down design. We can view the overall process as a *divide-and-conquer strategy*. A complex problem or functionality can often be divided into simpler ones, subsystems may then be developed to solve the simpler problems, and the overall system may aggregate the subsystems to derive a solution for the overall problem.

In nature, things usually occur (or rather evolve) the other way around, in a bottom-up fashion. There is no designer and no process to specify a complex or remote goal. The only goals are reproduction and survival. Simpler organisms and systems appear first because they are more likely to appear. Yet, sooner or later, more complex organisms emerge and those that have a substantial evolutionary advantage may subsume the simpler ones. Structure, functionality, and fitness usually grow in tandem. Even if an organism started as an unstructured blob of cells, it may eventually evolve into a specialized and highly structured being.

Irrespective of a top-down or a bottom-up approach, the result seems to be the same. Evolution is largely a monotonic process, climbing up a ladder of complexity and structure. There may be occasional setbacks. Environmental, geological, or extraterrestrial (and recently anthropocene) events may place a severe strain on life and may lead to extinctions. Yet, in the end, if life does not become extinct, it will keep growing and it will generate more structure, more complexity, and greater efficiency in the ecosystem.

Life is a synthesis, as cell theory dictates, and synthesis implies structure, complexity, and efficiency.

4.2 Synergy

Adversarial relationships are common in nature. Viruses and bacteria infect organisms. Predators feed on prey. There is struggle for survival between and within species. There is struggle for mating opportunities. Plants compete for soil. In dark forests, there is competition for light; trees grow taller to reach into the sunlight zone above the canopy.

The environment cannot support unlimited population growth. Space and habitat are finite. Resources are finite. Hence, there is competition for survival and reproduction. (In biology, *competition* is defined as a relationship between organisms in which the fitness of one is lowered by the presence of the other.) *Natural selection* states that organisms that are more fit (better adapted) to their environment tend to survive and reproduce more often.

Yet, life is not inherently adversarial. On many levels, it is primarily or exclusively synergetic. Synergy derives from the hellenic words συν- ‘together’

and ἔργον ‘work’. It literally means to work together but it often has the additional meaning that the cooperating elements produce a combined effect that is greater than the sum of their separate effects.

Inside an organism, cells must work together – to form tissue and organs and provide functionality that they cannot individually achieve. Their existence is primarily synergetic. Organs also cooperate with each other. Adversarial relationships are virtually nonexistent at their level. This is not surprising. Organs are functionally separate but mutually dependent. If they were to compete against each other for resources, organisms would perish quickly and by the actions of their own organs. As it might be expected, the situation in the nervous system is more complex but largely synergetic too.

Neurons can sometimes inhibit other neurons. Entire neural regions can provide inhibition. For example, visual and auditory cues can affect perception of taste. Identification of specific flavors is often less accurate when the flavors are paired with inappropriate colors. Visual cues are typically available before food is consumed. Biting or chewing sounds are available at the moment of consumption. Those create expectations and expectations can affect perception and potentially reduce or diminish contrary sensations.

The above can be perceived as adversarial relationships. However, the situation is more nuanced. The interactions are primarily consequences of *relative logical polarity** or, more generally, negative or positive correlations between random variables. The elements of multisensory information, memory, and perception are not always correlated in the same way. On the contrary, there is a complex range of relationships and, depending on the situation, they can have an overall excitatory or inhibitory effect or multiple and sometimes conflicting effects.

Without plasticity, it is a stretch to consider such effects as adversarial. On the other hand, when there is plasticity in the system, activation can play a role and inhibition may lead to loss of connectivity, which is an adversarial consequence when examined at the level of individual neurons.

On the other hand, the involved components and their neurons have largely a synergetic relationship, working together to determine reality. Underlying correlations are details. The components must work together in the end. The same virtually applies to all neural systems. Individual neurons cannot support complex cognitive tasks. Neural partitioning (at various and nested levels) and cooperation between the parts is required. Operating in the world involves multisensory perception, understanding, judgment, reasoning, planning, communication, concentration, balance, coordination, movement and various other functions. Distinct but interconnected and interrelated neural structures work together to fuse all that information and related

*In mathematical logic, literals can have preassigned polarity, either negative or positive. A logical expression A can have positive polarity with respect to an expression B if and only if A can only retain its logical value or become true when B changes from false to true. Similarly, negative polarity indicates that A can either retain its logical value or become false when B changes from false to true.

computations to create an effective cognitive whole.

As with life itself, neural systems represent an extensive synthesis. Synergy is simply a direct and inevitable consequence of the synthesis.

4.3 Graph Cuts

Synergy can significantly be affected when the synergetic parts have internal state that is shared among them. In this section, we will introduce some concepts from graph theory that will help us define such state for neural structures.

In graph theory, it is sometimes useful to consider subsets of the vertices of a graph or partitions of the vertices into two subsets.

Definition 4.1. A *cut* C of a directed graph $G = (V, E)$ is a subset of V , which implicitly defines a partition $(C, V - C)$ of V . The *cut-set* E_C of C is the set of edges between vertices of C and vertices of $V - C$

$$E_C = \{(c, v) \in E / c \in C \text{ and } v \in V - C\} = E \cap (C \times (V - C)).$$

The *periphery* V_C of the cut C is the subset of C that participates in E_C . In other words

$$V_C = \{c \in C / (\exists v \in V - C) (c, v) \in E_C\}.$$

(An equivalent definition is that V_C is the minimal set $S \subseteq C$ such that $E \cap ((C - S) \times (V - C)) = \emptyset$.)

C is called a *complete cut* (and E_C is called a *complete cut-set*) iff there are no edges from $V - C$ to C ; that is

$$E_{V-C} = \emptyset.$$

Furthermore, C is called an *input-output cut* iff all vertices of V with no predecessors are in C and all vertices of V with no successors are in $V - C$:

$$I_G \subseteq C \text{ and } O_G \subseteq V - C.$$

Finally, we can define the *peripheries* $\mathbb{P}_G$ of the graph G to be the set of the peripheries of its cuts. In other words

$$\mathbb{P}_G = \{V_C / C \subseteq V\}.$$

An example is shown in figure 4.2. The graph is drawn in layers arranged vertically from left to right. The set of vertices is $V = \{a, b, \dots, l\}$. The subset $C = \{a, b, c, e, f, h\}$ is a complete input-output cut and it is marked by a dashed outline. Its cut-set is

$$E_C = \{(a, d), (b, d), (f, j), (h, i), (h, j), (h, k), (h, l)\}.$$

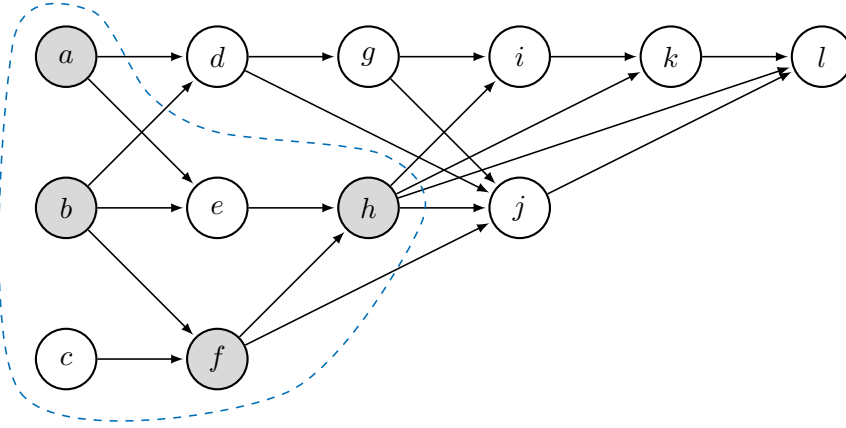


Figure 4.2: Graph cuts and peripheries.

The periphery of the cut is $V_C = \{a, b, f, h\}$ and its vertices are shaded gray.

Intuitively, if a complete cut C contains a layer, it must contain all layers before it (so that $E_{V-C} = \emptyset$). And if the cut C contains only entire layers, the periphery of the cut is its last layer along with the units in previous layers that provide cross-connections to $V - C$, if any.

Peripheries of complete input-output cuts represent a significant concept for neural networks as we shall shortly explain. Input-output cuts are desirable because they separate input from output. Complete cuts are useful because they have the following properties.

Lemma 4.1. *If $G = (V, E)$ is a graph, all complete cuts C of G have the following properties:*

- Paths that start in $V - C$ stay in $V - C$ (that is $G^+(V - C) \subseteq V - C$).*
- Paths that end in C start in C (that is $G^-(C) \subseteq C$).*
- Paths that connect C and $V - C$ cross between the two sets through the cut periphery V_C and only once.*

Proof. If any of the first two properties is violated then the violating path in G must contain a subpath $(v_0, \dots, v_n)$ such that $v_0 \in V - C$ and $v_n \in C$. However, if we define $m = \min\{i \leq n/v_i \in C\}$ then $m > 0$, $v_{m-1} \in V - C$, $v_m \in C$, and $(v_{m-1}, v_m) \in E$. Thus $(v_{m-1}, v_m) \in E_{V-C}$, which is a contradiction because C is a complete cut.

Similarly, for the third property, if $(v_0, \dots, v_n)$ is a path in G that starts in C and ends in $V - C$, then it must be $v_0 \in C$ and $v_n \in V - C$. And if we define $m = \min\{i \leq n/v_i \in V - C\}$ then $m > 0$, $v_0, \dots, v_{m-1} \in C$, $v_m \in V - C$, and $(v_{m-1}, v_m) \in E$. Thus $(v_{m-1}, v_m) \in E_C$. The latter implies that $v_{m-1} \in V_C$. Furthermore, the path $(v_m, \dots, v_n)$ is a path that starts in $V - C$ and, per the first property of this lemma, it must stay in $V - C$. In other words, it is $v_m, \dots, v_n \in V - C$. $\square$

We can understand the utility of the above properties if we consider the graph to be a computational model – as in our neural network definition 1.2. Input-output cuts partition the graph vertices into two non-empty sets with the input vertices in one of them and the output vertices in the other. Complete cuts imply that results that are computed in the cut C use information and resources only from C , while computations that start in $V - C$ remain in $V - C$ and thus use information and resources only from that set. Finally, computations that transfer information across the cut perform their transfer only once and through the cut periphery. In other words, the periphery defines a binary partition of the graph’s functionality as well as a communication channel between the parts that is used once for each computation that spans both parts.

Concluding, we should add that our definition of the cut-set is different from the standard definition. In most textbooks and graph theory literature, the cut-set includes edges in both directions and it is thus the union $E_C \cup E_{V-C}$. However, we chose to distinguish between the cut’s forward edges (E_C) from the backward ones (E_{V-C}). The two definitions are equivalent for complete cuts (since $E_{V-C} = \emptyset$) but not for the rest of the cuts. The distinction will become more relevant in later stages of the theory – especially for recurrent networks and incomplete cuts into their recurrence.

4.4 Neural State and Synergy

A simple form of synergy between neural networks or the parts of a neural partition can take place if the networks or parts share their outputs. However, this is a significant restriction on the information passed between such components and thus on their level of synergy. Neural networks usually develop sophisticated internal representations that contain considerably more information than their outputs. As a result, if two or more networks are to cooperate in a profound or maximal level, they should share some form of their internal representations.

In the extreme, the networks can share all their intermediate results. However, this is not effective or efficient and it is most likely detrimental. Intermediate results are computed from each other and they are thus highly redundant. It would be hard for each receiving network to process all that information, discard redundancies, identify important parts, and learn from them. Providing excessive amounts of information to a network usually has an adverse effect on learning. The underlying optimization problems and the corresponding learning tasks are made unnecessarily hard by the introduction of many additional dependent dimensions.

Better synergetic solutions exist between the above two extremes. We can find such a solution if we consider that the computation of a neural network takes place in stages. For example, the stages could correspond to the layers of the network. We can then argue that networks can cooperate

more effectively when they share the results of one or more of their stages – but not too many of them.

Flexibility suggests that a network must be free to select and receive any part of another network as shared information. Modularity implies that the decision of what constitutes shared information must be encapsulated in the providing network. For various reasons, including mathematical and hereditary simplicity as well as modular structures in growth and design, we have opted for the latter.

We can now be more formal and introduce the concept of state as an instrument for synergy.

In its most unconstrained form, *neural state* is simply a subset of the computations of a neural network. Given a neural network (G, F, Y) over sets of functions $(\mathcal{F}, \mathcal{P}, \mathcal{A})$, its *state* is simply a subset of Y_* (or an index in Y_{**}). If the graph G contains a unit for each element of Y , or if the state only includes elements that are associated with units, then the state is also a subset of the vertices of G . Thus it may correspond to a cut or a periphery of the graph. (Peripheries of complete input-output cuts are the most common neural states used in this theory.)

The above defines the internal state of a network. However, synergy requires the ability to integrate and process the states of other networks. If we add those states as input to the network, we may have to change the dimensionality of the input array. Worse the dimensionality of the incoming state may be incompatible with the dimensionality of the input. Even worse, state information is qualitatively different than input in most cases. In short, even though incoming state is a form of input, it should not be treated the same way as regular input.

The above raises an obvious question: how can incoming state be added to an existing network?

The answer is in the aggregating functions. Those functions provide alternative entry points (a back-door of sorts) into a neural network. For increased flexibility, we should use an additional aggregating function to combine the incoming states into a single aggregate state, if the network receives state from multiple networks. We must also indicate which aggregating functions of the network will receive the combined incoming state.

In order to support such functionality, we have to amend definition 1.2 by adding the state tuple S that defines internal and incoming state. For completeness, we provide the whole definition below. (Items A through E are exactly the same as before.)

Definition 4.2. Assume that $\mathcal{F}$, $\mathcal{P}$, and $\mathcal{A}$ are sets of non-polynomial, polynomial, and aggregating functions, respectively. Furthermore assume the identity function is contained in both sets $\mathcal{P}$ and $\mathcal{F}$ (as the only polynomial member of the latter). Then a *feed-forward neural network* over $(\mathcal{F}, \mathcal{P}, \mathcal{A})$ is a tuple (G, F, Y, S) that satisfies the following conditions:

- A. $G = (V, E)$ is a directed acyclic graph with a maximal forward partition $L_0, \dots, L_n$ for some $n \in \mathbb{N}$.
- B. $F = (F_1, \dots, F_n)$ and, for all $k \in \{1, \dots, n\}$, $F_k = f_k \circ p_k \circ a_k$, $f_k \in \mathcal{F}$, $p_k \in \mathcal{P}$, and $a_k \in \mathcal{A}$.
- C. Each vertex $u \in V$ represents a computing element that produces an output. $Y = (Y_0, \dots, Y_n)$ is a tuple of arrays such that Y_k stores the output of layer L_k for $k = 0, \dots, n$. Given that units are uniquely allocated to layers, we allow Y to be indexed by units directly. Thus we write, Y_u to indicate the output of unit u (instead of $Y_{k,j}$ when $u \in L_k$ and j is the index of u in L_k .) Similarly, we write Y_B to indicate the outputs of a set of units $B \subseteq V$. The dimensionality of Y_B is then implicitly derived from the units in B . (Thus Y_k is the same as Y_{L_k} .)
- D. $Y_0 = Y_{I_G}$ is the input the network and $Y_n = Y_{O_G}$ is its output.
- E. For each layer L_k , the set $I_k = G^{-1}(L_k)$ is the set of units that provide input to the layer. When $k > 0$, $S_k = I_k \cap L_{k-1}$ and $C_k = I_k - L_{k-1}$ are the sets of units that provide input to the layer through successive or cross-connections, respectively. Similarly, for each unit $u \in L_k$, $I_u = G^{-1}(u)$ is the set of units that provide input to u . When $k > 0$, $S_u = I_u \cap L_{k-1}$ and $C_u = I_u - L_{k-1}$ are the sets of units that provide input to u through successive or cross-connections, respectively. (Obviously, $I_0 = S_0 = C_0 = \emptyset$ and $(\forall u \in L_0) I_u = S_u = C_u = \emptyset$.)
- F. $S = (\sigma, \alpha, \kappa)$ is the *state tuple*. Its contents are as follows.
- σ defines the state of the network. Thus $\sigma \subseteq Y_*$. Y_σ is the value of the state that can be used by other networks.
- α is the aggregating function for the incoming states. Thus, if K is a tuple with the incoming states, $\alpha(K)$ is the aggregate incoming state.
- κ is the set of the indices of the network's aggregating functions that assimilate the aggregate incoming state. 0 can be included in κ to indicate that incoming state overrides the input to the network. Thus $\kappa \subseteq \{0, \dots, n\}$. The incoming state into each layer can then be defined as follows:

$$(\forall k \in \{1, \dots, n\}) \quad \Theta_k = \left\{ \begin{array}{ll} \alpha(K) & \text{if } k \in \kappa \\ \emptyset & \text{otherwise} \end{array} \right\}.$$

And the input the network is redefined:

$$Y_0 = \left\{ \begin{array}{ll} \alpha(K) & \text{if } 0 \in \kappa \\ Y_{I_G} & \text{otherwise} \end{array} \right\}.$$

- G. The computations at each layer satisfy the property

$$(\forall k \in \{1, \dots, n\}) \quad Y_k = F_k(Y_{S_k}, Y_{C_k}, \Theta_k) = F_k(Y_{I_k}, \Theta_k).$$

A neural network is *stateless* iff it has no state, that is when $\sigma = \emptyset$. It is *state sensitive* or insensitive iff it admits or does not admit incoming state, that is when $\kappa \neq \emptyset$ or $\kappa = \emptyset$, respectively.

Although there are no actual constraints, we think that the most common situation for incoming states would be when they are passed into a single layer; that is when κ is a singleton containing the index of a layer to process incoming states. We also think that the most common situation for internal state would be a periphery of the graph of the network, most likely a layer near the output layer when there are no cross-connections that bypass the layer.

The output layer can also be used as state. This is an extreme case (the cut is the whole graph) but it is meaningful in certain situations and for small data sets. *Ensemble models* are a special case of such state configuration except that, in those models, the individual networks are trained separately and thus there is no direct synergy. However, there is often implicit synergy through random sampling of the data set (*bagging*) or through learning emphasis on misclassified samples (*boosting*).

In this theory we are more interested in explicit synergy.

Definition 4.3. *Neural synergy or cooperation* occurs when two or more neural networks (or partitions of a single network) share their state. In that case, we call the networks *synergetic* or *cooperative*.

In general, synergetic neural networks may have their own outputs or they may only exist to provide state. If they have outputs, they may try to solve the same problem(s) as the overall network, address individual sub-problems, or tackle independent problems in a multi-task system. They can thus have their own loss functions, participate in an overall loss function, or contribute to a set of partial loss functions. We will examine the learning process in more detail in section 4.6.

4.5 A Neural Algebra

The word algebra is of Arabic origin and it means “reunion of broken parts”. In mathematics, *algebra* is a branch where letters or other symbols are used to represent mathematical entities (such as quantities or relations), rules are provided for the manipulation of those symbols, and theorems are proved to demonstrate various properties of the mathematical structure behind the entities.

New theories often contain new formalisms to describe their subject matter. Sometimes the theories are broad enough and the formalisms sufficiently generic to constitute an algebra. For example, by the 5th century BCE, and long before the word algebra was coined, Hellenic mathematicians had already introduced a *geometric algebra* to represent simple geometric objects and prove relations between them. In the 3rd century CE, Diophantus published a

series of books called *Arithmetica* where he introduced an *arithmetic algebra* that was applied to algebraic equations. Various other algebras have been proposed since then. One of the most recent examples is the *relational algebra* for databases that was proposed in 1970.

Mathesis contains a new *neural algebra* that is used to represent and study neural networks, their structure, synergy, learning, growth, plasticity, heredity, and various other aspects and properties. We implicitly initiated the algebra with the definition of neural networks (definition 1.2). We will add more to it in the remaining of this chapter and we will augment it a number of times as we advance further into the theory.

In the following, we will use the letters N , M , and R to indicate neural networks with state as in definition 4.2.

We allow the networks to be indexed by named subscripts. Thus, N_G , N_F , and N_Y denote the graph, function, and array of the neural network N while N_S , N_σ , N_α , and N_κ denote the state tuple, state index, incoming state aggregating function, and incoming state index of N . Furthermore, N_Θ indicates the incoming state for each layer of N and N_s stands for the state of the network. Thus $N_s = (N_Y)_{N_\sigma}$.

In order to simplify notation, we also use the index i to refer to members of the network N_i (for $i = 1, 2, \dots$). Thus, $\tau_i = (N_i)_\tau$ where τ is G , F , Y , S , σ , α , κ , Θ , or s . For example, $s_i = (N_i)_s$ and $\sigma_i = (N_i)_\sigma$.

We also split the network in two parts, one containing the state and the other containing the part after the state. Thus if a network N consists of layers $L_0, \dots, L_n$, N_- is defined to be the subnetwork of layers $L_0, \dots, L_m$ while N_+ is defined to be the subnetwork of layers $L_{m+1}, \dots, L_n$, where L_m is the last layer that contributes to state (that is $m = \max\{k \leq n \mid L_k \cap N_\sigma \neq \emptyset\}$).

The above defines some basic notation for the networks of our neural algebra. The algebra itself consists of six components, which will be introduced in the following subsections.

4.5.1 Neural Output

Let us consider the input and the output of a neural network. We write $N(X)$ to indicate the output of the neural network N on input X . (We will also use the same notation to refer to the computation of N upon input X when we can disambiguate the expression from the context; otherwise we will explicitly refer to the computation.)

We write $N_1 \star N_2$ to indicate the network that includes networks N_1 and N_2 and whose output is $(Y_1)_{O_{G_1}} \star (Y_2)_{O_{G_2}}$, where $\star$ can be any valid operation between the outputs $(Y_1)_{O_{G_1}}$ and $(Y_2)_{O_{G_2}}$ of the two networks. As a result, $(N_1 \star N_2)(X) = N_1(X) \star N_2(X)$. For example, if the outputs of the networks have the same size and dimensionality, $N_1 + N_2$ represents the ‘sum’ of the two networks and $\frac{N_1 + N_2}{2}$ represents their ‘average’.

The extension to an arbitrary number of networks is straightforward

$$(N_1 \star_1 N_2 \star_2 \cdots N_\omega)(X) = N_1(X) \star_1 N_2(X) \star_2 \cdots N_\omega(X).$$

4.5.2 Neural Input

In general, we say that a network M provides *neural input* to network N (or, equivalently, that N receives neural input from M) iff any part of the output of M is placed into the input layer of N .

We write $N(M)$ or $N \circ M$ when the output of network M is the input to N . Thus $(N \circ M)(X) = N(M)(X) = N(M(X))$.

When N also receives the input X , as in the expression $N(X, M(X))$ we use the notation $N(I, M)$, where I represents the identity function (over arrays or tuples). Therefore,

$$N_{\omega+1}(I, N_1, \dots, N_\omega)(X) = N_{\omega+1}(X, N_1(X), \dots, N_\omega(X)).$$

Of course, each subnetwork N_i can use a separate part of the input. In that case, it is

$$N_{\omega+1}(I, N_1, \dots, N_\omega)(X) = N_{\omega+1}(X, N_1(X_{j_1}), \dots, N_\omega(X_{j_\omega})),$$

where j_i is an index that selects the part of the input X that is intended for the network N_i – which can be anything from a single element to the entire input X .

Given that the input X can be factored out as described above, it is omitted (along with the identity function I) from the remaining equations of this chapter. (We will only demonstrate it once more.) On the other hand, input from other networks is essential in our algebra and it cannot be omitted.

4.5.3 Neural Parallelism

Neural networks can operate in sequence, one after another, or in parallel. For example, when a set of neural networks are arranged in a sequence so that the output of each network is the input of the next, then the networks must operate in that sequence. On the other hand, when there are no dependencies between the networks, they can operate in parallel.

We write $M \triangleright N$ to indicate a *sequential operation* where the network M must complete its computations before N can start its own. Conversely, we write $M \parallel N$ to indicate that the computations of the two networks can take place in *parallel*.

Parallel operations have higher notational precedence. Thus

$$N_1 \parallel N_2 \triangleright N_3 \triangleright N_4 \parallel N_5 \parallel N_6 = (N_1 \parallel N_2) \triangleright N_3 \triangleright (N_4 \parallel N_5 \parallel N_6).$$

The above expression indicates a parallel operation of networks N_1 and N_2

followed by the operation of network N_3 , which is followed by a parallel operation of networks N_4 , N_5 , and N_6 .

In order to write formulas more concisely, we will also use the operators $\sqparallel$ and $\sum$ to indicate parallel and sequential operation. Thus

$$\begin{aligned} \sqparallel_{i=1}^{\omega} N_i &= N_1 \parallel \cdots \parallel N_{\omega} \quad \text{and} \\ \sum_{i=1}^{\omega} N_i &= N_1 \triangleright \cdots \triangleright N_{\omega}. \end{aligned}$$

4.5.4 Neural Repetitions

We write N^{θ} to indicate the operation of a neural network N with incoming state θ . ($N^{\theta}(X)$ indicates the operation of N with incoming state θ and input X .) When $\theta = \emptyset$, we simply omit it. Thus $N^{\emptyset} = N$.

Definition 4.4. *Neural repetition* refers to the repetition of the computations of a neural network N , first with a given incoming state θ and subsequently with its own state as incoming state. We write $(N^{\theta})^r$ (or $N^{\theta r}$ for notational simplicity) to indicate the operation of the network N *repeated* r times. $r = 0$ indicates no computation. For one or more repetitions, it is

$$\begin{aligned} N^{\theta(r+1)} &= N^{\theta} \triangleright N^{N_s r} \\ &= N^{\theta} \triangleright \sum_{i=1}^r N^{N_s}. \end{aligned} \tag{4.1}$$

In general, it should be $N^{\theta(r+1)} = N^{\theta} \triangleright N^{\gamma r}$, where γ is a subtuple* of the state N_s . However, the aggregating function N_{α} can pick any subtuple of N_s , if we pass N_s as the incoming state. In other words, with equation (4.1), we have opted to include the selection process in the aggregating function instead of explicitly specifying it in the recursion of repetitions. State selection is a property that can be defined within a network. It makes the network more modular. We see no reason to have such a definition outside the network.

4.5.5 Neural Equivalence

An important concept in our algebra is the operational equivalence of neural networks.

Definition 4.5. Two neural network representations N and M are *equivalent* iff for all valid inputs X and incoming states θ the networks produce the same state and output. In that case, we write that $N \equiv M$.

*A tuple x is a *subtuple* of a tuple y iff there are distinct indices $i_1, i_2, \dots$ such that $x = (y_{i_j}/j = 1, 2, \dots)$. In that case, we write $x \subseteq y$. If the indices are increasing, x is called an *ordered subtuple* of y and we write that $x \subseteq_{\uparrow} y$.

Neural equivalence can be used to remove redundant computations in our neural algebra. For example, if a network N disregards incoming state (that is when $N_\kappa = \emptyset$), it is

$$(\forall r \geq 1) \quad N^{\theta r} \equiv N^\theta \equiv N.$$

4.5.6 Neural Synergy

Neural state and synergy complicate dependencies and thus parallelism and the order of computation. Synergy can be parallel or sequential. It can be unidirectional or bidirectional and thus recursive. It can also be complete or partial with selection mechanisms. We shall discuss those differences in the following subsections.

4.5.6.a Parallel Neural Synergy

Definition 4.6. *Parallel neural synergy* occurs when two or more neural networks share their state but the state dependencies do not force the networks into a sequential operation. (The state dependencies permit the networks to operate in parallel if the rest of their dependencies are removed.) *Parallel synergetic neural networks* are parallel networks with parallel synergy.

State feedback refers to state interactions that allow the state of a network to affect another state or itself. Parallel synergy can contain state feedback or not. If there is feedback, it can be repeated a number of times.

In general, the equations for parallel synergy with r rounds of feedback ($r + 1$ iterations total) have the following form.

$$\left(\bigstar_{i=1}^{\omega} N_i \right)^{\theta(r+1)}(X) = \bigstar_{i=1}^{\omega} N_i^\theta(X_{j_i}) \triangleright \left(\bigstar_{i=1}^{\omega} N_i^s(X_{j_i}) \right)^r, \quad (4.2)$$

where $s = (s_1, \dots, s_\omega)$ is the state of the overall network, $\bigstar$ is either $\coprod$ (for parallel networks) or $\sum$ (for sequential networks), θ is the initial incoming state for the network, and j_i is an index that selects the part of the input X that is intended for the network N_i . (Note that θ and s do not need to have the same size or dimensionality. Aggregating functions operate on arbitrary tuples. Initial networks in a broader neural synergy will probably have no incoming state (that is $\theta = \emptyset$) and they will generate their own state.)

In the above equation, we included the input X just to demonstrate a full example. We omit the input in the rest of this chapter. Thus we write equation (4.2) as

$$\left(\bigstar_{i=1}^{\omega} N_i \right)^{\theta(r+1)} = \bigstar_{i=1}^{\omega} N_i^\theta \triangleright \left(\bigstar_{i=1}^{\omega} N_i^s \right)^r. \quad (4.3)$$

Let us now explain the concepts of repetition and feedback and their utility in more detail.

When $r = 0$, there is no state feedback between the constituent networks N_i for $i = 1, \dots, \omega$. Each network computes its own state and output. Those can then be accessed by larger networks that include the constituents as subnetworks but the state s_i of network N_i does not affect the state s_j of network N_j for all $i \neq j$.

When $r = 1$, there is a single round of feedback. The constituent networks first compute their output and their states s_i for $i = 1, \dots, \omega$ based on the initial incoming state θ and their input X . Then they update their incoming state to be their combined state $s = (s_1, \dots, s_\omega)$ and they recompute their output and state. In other words, the constituent networks receive the state of the other constituents as incoming state and they use that information to update their own state. This is effectively a feedback mechanism.

When $r > 1$, the feedback loop is repeated r times. However, there is a qualitative difference. The first repetition uses an incoming state s that was generated from the initial incoming state θ while the other repetitions use an incoming state s that was generated from the previous value of s . Each network is thus ‘aware’ of the states of the other networks in each repetition but the incoming state to the first repetition was not generated under such ‘awareness’.

A large repetition number r will probably have no significant effect and it can even be detrimental.* However, the cases where r is 0, 1, 2, or some small number can result in potentially significant differences depending on the complexity of the information in the states, the training data, and the complexity of the underlying problem.

4.5.6.b Neural Rigidity

Feedback is not always possible through states.

Definition 4.7. A network N where the incoming state does not affect N_- is called *rigid*. The formal definition for rigidity is thus

$$N_\kappa = \emptyset \quad \text{or} \quad N_\sigma \subseteq \bigcup_{i < \min(N_\kappa)} L_i.$$

Rigid neural networks cannot receive state feedback because their incoming state does not affect their state. Repetitions with updated incoming state will only affect the output of the networks but not their state.

Consequently, rigidity can make some computations redundant and enable algebraic reductions as shown by the following lemma.

Lemma 4.2. For rigid networks $N_i, i = 1, \dots, \omega$, it is

$$\left(\prod_{i=1}^{\omega} N_i \right)^{\theta(r+1)} \equiv \prod_{i=1}^{\omega} N_{i-} \triangleright \prod_{i=1}^{\omega} N_{i+}^s.$$

*Hint: consider the concept of a *Krylov subspace* with a non-linearity introduced at each iteration.

Proof. The proof derives from the following three facts for a rigid neural network N :

- (a) $N^s = N_- \triangleright N_+^s$ because N_- is not affected by incoming state.
- (b) Repeated executions of N_- are identical (nothing changes from one repetition to the next). Thus $N_- \triangleright M \triangleright N_- \equiv N_- \triangleright M$ for any neural network or operation M that does not affect N_- .
- (c) When N_+^θ is followed by N_+^s the second operation overrides the first. In other words $N_+^\theta \triangleright M \triangleright N_+^s \equiv M \triangleright N_+^s$, for any neural network or operation M that does not depend on N_+ or affect N_+ . Similarly, repeated executions of N_+^s are identical, if the incoming state s does not change (since everything else remains the same). Thus $N_+^s \triangleright \dots \triangleright N_+^s \equiv N_+^s$.

Therefore

$$\begin{aligned}
 \left(\prod_{i=1}^{\omega} N_i \right)^{\theta(r+1)} &= \prod_{i=1}^{\omega} (N_{i-} \triangleright N_{i+}^\theta) + \left(\prod_{i=1}^{\omega} (N_{i-} \triangleright N_{i+}^s) \right)^r && \text{by (a)} \\
 &\equiv \prod_{i=1}^{\omega} (N_{i-} \triangleright N_{i+}^\theta) \triangleright \left(\prod_{i=1}^{\omega} N_{i+}^s \right)^r && \text{by (b)} \\
 &\equiv \prod_{i=1}^{\omega} N_{i-} \triangleright \prod_{i=1}^{\omega} N_{i+}^s && \text{by (c).}
 \end{aligned}$$

□

Note that the above lemma does not apply to sequential networks because such networks provide input to each other and, as a result, each repetition operates on possibly changed inputs.

In general, rigidity is a good property for networks that provide low-level state that is mostly independent of context or other states. Such networks usually operate in parallel and with parallel synergy. Thus lemma 4.2 applies. Irrespective of other factors or the environment, evolution of such structures is then bound to converge to a scheme with no feedback (non-recursive) as long as they remain rigid. Feedback simply has no effect on rigid parallel synergetic networks.

The above may seem to be contradicted by the proposal of recurrent and state-dependent models for the early layers in the vertebrae visual system, which we briefly discussed in section 1.5. However, this is not actually a contradiction. An evolved neural system would probably use a mixture in its early layers. It would use stateless or rigid parallel synergetic neural networks for properties that can efficiently be computed by such networks. However, it would also use other types of networks for properties that require context or feedback. Those networks should be able to process contextual or feedback interactions and thus should have a form of recursion. Consequently, if we

examine the early layers of such a system we would most likely find evidence of a distribution, from stateless and rigid structures to fully recursive components. This is consistent with studies in neuroscience that have examined related evidence.

In the end, rigidity is bound to be a limiting factor at higher levels or in largely contextual neural states. We anticipate a reduced chance of occurrence at such levels or situations – either by design or evolution.

4.5.6.c Sequential Neural Synergy

Sequential and parallel neural synergies are expressed with similar notation. However, there are two main semantic differences. Parallelism is now impossible because the states require sequential treatment. And second, the incoming state for the first iteration of the network N_i is not necessarily s_0 ; it can be any subtuple of $(s_0, \dots, s_{i-1})$, where s_0 is the incoming state to the overall network. This is because the networks operate in sequence and thus the states $s_1, \dots, s_{i-1}$ as well as the initial incoming state s_0 are available to N_i . Thus the equation for sequential synergy is

$$\left(\bigstar_{i=1}^{\omega} N_i \right)^{s_0(r+1)} = \sum_{i=1}^{\omega} N_i^{\beta_i} \triangleright \left(\sum_{i=1}^{\omega} N_i^s \right)^r, \quad (4.4)$$

where $\bigstar$ is either $\prod$ or $\sum$, $s = (s_1, \dots, s_{\omega})$, and $\beta_i = (s_0, \dots, s_{i-1})$ for $i = 1, \dots, \omega$.

Note that applying sequential synergy to parallel networks would force them to be sequential. Such synergy should be used sparingly and only when parallel synergy is considerably worse. In that case, hybrid forms should also be considered. Long chains or repetitions of sequential synergy across parallel networks are unlikely to occur because they would force equally long chains of inference. For example, sequential synergy across various components of a lobe in the brain or across lobes would force all those components to operate sequentially thus slowing down perception and cognition by a factor that is linear to the number of components. Evolutionary pressure would eliminate such an organization, if it ever occurs.

4.5.6.d Neural State Selection

In his novel, *Anna Karenina*, L. Tolstoy wrote that “Happy families are all alike; every unhappy family is unhappy in its own way.” The same can perhaps apply to synergy and state selection.

As we discussed in subsection 4.5.4, each neural network can select among its incoming state using its aggregating function α . In equation (4.3), the incoming state s has ω constituent states s_i , one from each network N_i , for $i = 1, \dots, \omega$. Each of those states can be a large tuple or array. Even if we limit the networks to select some among the states s_i , there are 2^{ω} possibilities

for each network N_i ; and thus there 2^{ω^2} possibilities total. Sequential synergy implies even more possibilities: $\prod_{i=1}^{\omega} 2^{i+\omega} = 2^{\omega^2 + \omega(\omega+1)/2} \simeq 2^{3\omega^2/2}$ total. Those are a prohibitive quantities even for modest values of ω .

On the other hand, the above estimates are theoretical combinations. Evolutionary or empirical processes cannot and should not explore such magnitudes or sparsities. The vast majority of the combinations would be unfortunate or insignificant. Metaphorically, they correspond to the ‘unhappy families’ in Tolstoy’s quote. Either the full state s or β_i or some locality (e.g. (s_{i-1}, s_i)) should be selected by the aggregating functions α_i .

4.5.7 An Algebra for Synthesis

We can now combine all the elements to create our neural algebra.

In general, a synergetic neural network consists of three parts. The first part describes all the subnetworks N_i that are contained in the synergetic network. The second part describes the structure of the overall network through the input/output connections between its subnetworks. The third part describes the synergy between the subnetworks.

We can better understand all this with an example.

Definition 4.8. *Hierarchical synergy* is the synergy in a set of parallel synergetic networks that provide input to a sequential synergetic network. Thus the network has the form

$$N_{\omega+1}(N_1, \dots, N_{\omega})$$

and the synergy has the form

$$\prod_{i=1}^{\omega} N_i \triangleright N_{\omega+1},$$

where $N_1, \dots, N_{\omega}$ are the parallel synergetic networks, called *synergetic children*, and $N_{\omega+1}$ is the sequential synergetic network, called the *synergetic parent*.

Using the synergy formulas from the previous subsections, we can derive the synergy for the hierarchical network.

$$\left(\prod_{i=1}^{\omega} N_i \triangleright N_{\omega+1} \right)^{s_0(r+1)} = \prod_{i=1}^{\omega} N_i^{s_0} \triangleright N_{\omega+1}^{\beta} \triangleright \left(\prod_{i=1}^{\omega} N_i^s \triangleright N_{\omega+1}^s \right)^r, \quad (4.5)$$

where s_0 is the incoming state to the network, $\beta = (s_0, \dots, s_{\omega})$, and $s = (s_1, \dots, s_{\omega+1})$.

Networks that receive their own state as part of their incoming state and choose to use it are called *reflexive*. Thus, if $\alpha_{\omega+1}$ uses $s_{\omega+1}$ from its incoming state $s = (s_1, \dots, s_{\omega+1})$ then the network $N_{\omega+1}$ is a reflexive subnetwork of

the whole network. However, if $\alpha_{\omega+1}$ does not select $s_{\omega+1}$ then $N_{\omega+1}$ is not reflexive.

Now, if $N_{\omega+1}$ is rigid and non-reflexive and feedback is only among the children or from the children to their parent (which means that the children do not use the state $s_{\omega+1}$ of their parent) then equation (4.5) reduces to the following:

$$\left(\prod_{i=1}^{\omega} N_i \triangleright N_{\omega+1} \right)^{s_0(r+1)} \equiv \left(\prod_{i=1}^{\omega} N_i \right)^{s_0(r+1)} \triangleright N_{\omega+1}^{(s_1, \dots, s_{\omega})}. \quad (4.6)$$

In other words, rigid non-reflexive networks localize repetitions to the networks that precede them. Consequently, repetitions over a larger network reduce to repetitions on the parts of the network that are non-rigid or reflexive. If we understand the effect of the repetitions on those parts we can then infer the effect on the overall network.

This is a useful property, but rigidity is a severe constraint. A rigid network does not propagate incoming state through its own state, only implicitly through its output. However, the output is not necessarily a good representation for state.

In general, we would like to be able to define hierarchical synergy to have the form of equation (4.5) or the form of equation (4.6) or various other forms irrespectively of rigidity.

For example, in our experiments, we have used a form of synergy that separates state processing and feedback into distinct subnetworks without affecting the original networks and their outputs. Thus the synergy is defined as

$$\left(\bigstar_{i=1}^{\omega} (N_i, M_i) \triangleright N_{\omega+1} \right)^{\theta(r+1)} = \bigstar_{i=1}^{\omega} N_i^{\theta} \triangleright \left(\bigstar_{i=1}^{\omega} M_i^s \right)^{r+1} \triangleright N_{\omega+1}^s, \quad (4.7)$$

where $s = (s_1, \dots, s_{\omega})$, $\bigstar$ is either $\prod$ or $\sum$, θ is the initial incoming state for the network, and, for $i = 1, \dots, \omega$, N_i and M_i , are the original and state processing subnetworks, respectively.

Consequently, synergy should not be derived by the structure of the network and generic equations (such as equations (4.3) and (4.4)) but rather be defined individually for each synergetic network. Synergy defines the interactions between neural states and such interactions may be diverse or arbitrary. They could be different for similar or identical neural structures if those structures address different problems or represent distinct functionality.

In general, the formulas that describe the synergy in a synergetic network can be arbitrary expressions of our algebra. Of course, if synergy is to be acquired through learning and evolution, there should be some bias that limits the number of possible forms. However, a one-to-one mapping between structure and synergy is unlikely to be an effective bias.

Hierarchical synergetic neural networks represent one of the most natural

form of synergy and they were tested first in this theory. In most of our experiments, either the parent did not have its own state or there was no feedback from the parent to the children. State information only traveled in the opposite direction (and among the children, too). However, this is not necessarily the best synergy in all hierarchical cases. A custom synergy definition can define feedback that goes either way or both ways. It is then an empirical and evolutionary problem to identify the best form.

From a system viewpoint, a neural network framework should contain default synergy models but, more importantly, it should allow for custom synergy, too. The framework should thus support an algebra for synergy. Connectivity and functionality alone are not sufficient.

Sequential synergy can significantly increase the effective depth of a network. It can similarly affect latency, interpretation, and structure. Therefore, it should be used with caution, especially within plasticity and evolutionary methods. Platforms should always issue warnings in such situations and evolutionary methods should discount the probability of occurrence of sequential synergy so that it emerges either infrequently or rarely.

4.5.8 Is an Algebra Necessary?

From a reductive viewpoint, we can perceive a large synergetic network as a recursive system. Repetitions are merely iterations of the recursions. Synergy can be modeled through cross-connections. Those are already processed by aggregating functions, which can be used to select among connections and thus elements or subtuples of state. In fact, in this proposed form, we do not need neural state at all. The state is implicit in the recursion and it is accessible through cross-connections from and into the recursion. Of course, our new recursive network would no longer be based on an acyclic graph but that is a manageable complication.*

The above neural form seems to put in question the entire synergetic approach as it may render it a superfluous formalism. However, such reductive approaches are misleading and conceptually perilous. They should not be used to evaluate systems and formalisms. They are not useful at such a level. Let us understand why.

In a similar reductive approach, we can view division as multiplication and multiplication as addition. Thus we can view division as addition! In fact, we can reduce all of rational functions to addition. Yet, such a reduction is not useful except perhaps for axiomatization purposes at a low level.

Indeed, all of set theory (and thus all of mathematics) can be built from the concept of the empty set. However, only few proofs early in the construction of set theory operate at such a low level. The rest of set theory and mathematics

*We will introduce such cyclic graphs later in the theory. If repetitions are predefined and bounded by a small number, we can always ‘unfold’ the corresponding recursion into multiple layers and have a non-recursive network and thus an acyclic graph – albeit at the cost of shared parameters.

are built on top of more elaborate constructs – symbols, operators, functions, relations, graphs, measures, groups, fields, manifolds, etc.

Reductionism is contrary to the basic tenets of algebra – and mathematics as well. We use symbols, operators, and entire algebras because those allow us to describe mathematical structures in concise and interpretable ways. And when we introduce new and more effective formalisms, we make progress in science.

A large network can be defined in terms of partitions, structure, and synergy so that it is meaningfully composite. Or it can be defined at the level of individual units and layers with a number of connectivity patterns and multi-unit recursions in different segments and various cross-connections into and from those recursions. However, this latter definition is more cluttered than disciplined and more cumbersome than useful. It is similar to a software application with source code that does not consist of modules such as libraries, classes, and functions but rather it is written as a continuous sequence of instructions inside a few long functions that span thousands or millions of lines of code.

It is hard to work with such constructs or understand what they do. It is even harder or completely intractable to evolve them through an automated process. Design, learning, and evolution become unnecessarily hard and often impossible.

In short, synergy and the associated neural paradigm are direct consequences of synthesis. Structure, synergy, and synthesis are fundamental elements of a larger process that includes learning and evolution. They are essential tools for managing complexity. Without them, neither design nor evolution can go very far.

4.6 Cooperative Learning

Synergy is not complete without some form of learning – or, at least, optimization. Learning in synergetic networks is called *synergetic* or *cooperative learning*.

The principle of learning duality (from section 3.2) can be extended to the subnetworks of a synergetic network and it implies that those subnetworks maintain a dual individualist and collective existence. Thus they may participate in an overall optimization process through a loss function but they may also have their own optimization goals and loss functions, too.

In general, if a synergetic neural network $N_{\omega+1}$ contains subnetworks $N_1, \dots, N_\omega$, we can associate a loss function ℓ_i with each network N_i for $i = 1, \dots, \omega + 1$. (If a subnetwork N_i is only collective, it would be $\ell_i = 0$.)

Each loss function can have its own learning rate λ_i and compute the partial derivative $\frac{\partial \ell_i}{\partial w}$, where w represents the learnable parameters of the entire network.* Then it can contribute its own update, $(\Delta w)_i = -\lambda_i \frac{\partial \ell_i}{\partial w}$.

*For computational efficiency, we can use w_i to represent the learnable parameters of

Alternatively, we can use a composite loss function ℓ that is a function of $\ell_1, \dots, \ell_{\omega+1}$. In other words $\ell = \ell(\ell_1, \dots, \ell_{\omega+1})$. In that case, the derivative is obtained through the chain rule

$$\frac{\partial \ell}{\partial w} = \sum_{i=1}^{\omega+1} \frac{\partial \ell}{\partial \ell_i} \frac{\partial \ell_i}{\partial w} \quad (4.8)$$

and the update is

$$\Delta w = -\lambda \frac{\partial \ell}{\partial w}. \quad (4.9)$$

We will refer to the above two loss configurations as *individual* and *composite loss*, respectively.

When we use a stateless optimization method (such as stochastic gradient descent), individual and composite loss configurations are equivalent. We can simply pick $\lambda_i = \lambda \frac{\partial \ell}{\partial \ell_i}$, which corresponds to a linear composite loss function

$$\ell = \sum_{i=1}^{\omega+1} \frac{\lambda_i}{\lambda} \ell_i. \quad (4.10)$$

Indeed, when equation (4.10) is true, it is

$$\Delta w = -\lambda \frac{\partial \ell}{\partial w} = -\lambda \sum_{i=1}^{\omega+1} \frac{\partial \ell}{\partial \ell_i} \frac{\partial \ell_i}{\partial w} = -\lambda \sum_{i=1}^{\omega+1} \frac{\lambda_i}{\lambda} \frac{\partial \ell_i}{\partial w} = \sum_{i=1}^{\omega+1} (\Delta w)_i.$$

The above assumes that computations of derivatives $\frac{\partial \ell_i}{\partial w}$ precede all updates $(\Delta w)_i$, for $i = 1, \dots, \omega + 1$. In general, this is a safe practice. (In stochastic situations, the constraint may be relaxed and derivatives of a subnetwork be computed asynchronously with the updates of other subnetworks especially in later stages of training where there is limited change in the output of each network by each update. Of course, this can lead to errors and numerical inconsistencies. The approach effectively trades correctness and robustness for efficiency and it should be used with caution.)

On the other hand, for learning purposes, stateless optimization is limiting. More advanced optimization methods, such as those described in chapter 7, maintain an internal state based on prior values of derivatives or updates. Such methods break the equivalence of the above two loss configurations and introduce a trade-off.

In general, individual and composite losses are not the only options. Multiple composite losses can be used in a hybrid approach where groups of losses are represented by composite losses; and the composite losses can then have their own updates.

N_i and use the derivative $\frac{\partial \ell_i}{\partial w_i}$ for $i = 1, \dots, \omega + 1$. Mathematically, there is no difference, assuming that each loss function ℓ_i is local in each network N_i , which means that ℓ_i does not operate on the outputs of any network other than N_i .

For simplicity, we will often use a single composite loss function ℓ and a single update Δw . However, we will sometimes provide dual definitions and point out differences between individual and composite losses when it clarifies the discussion.

4.6.1 Homologous and Heterologous Learning

Neural synergy is primarily a structural property. It only requires that two or more neural networks share their state (see definition 4.3). It does not imply similar or compatible learning goals.

In general, learners can have contrasting, similar or even identical objectives. For example, in the adversarial approaches of section 2.3, learners have contrasting objectives.

Let us now define learning types more formally.

Definition 4.9. We assume that a learning process $\mathcal{L}$ contains a loss function ℓ and a learning rate λ . We also assume that a (synergetic) learner N contains one or more learning processes. We can then define the following:

Polarity. The *loss polarity* (or simply *polarity*) of a loss ℓ is the sign of its supremum (or -1, if the supremum is 0):

$$p^{\pm}(\ell) = \begin{cases} +1 & \text{if } \sup\{\ell\} > 0, \\ -1 & \text{otherwise,} \end{cases}$$

where $p^{\pm}$ is the *polarity operator* that returns the polarity of its argument.

The *learning polarity*, or *polarity*, of a learning process $\mathcal{L}(\lambda, \ell)$ with learning rate λ and loss ℓ is

$$p^{\pm}(\mathcal{L}) = \text{sign}(\lambda) p^{\pm}(\ell).$$

The *learning polarity*, or *polarity*, of a learner is the polarity of its learning processes. If the learner contains two or more learning processes with different polarities, the learner is called *bipolar*.

Homology. Two or more losses or learning processes are called *homopolar* iff they have the same polarities. The processes are called *homologous* iff they are homopolar and they operate on the same input domain.

Two learners are called *homopolar* or *homologous* iff there is a bijection between their learning processes that associate homopolar or homologous processes, respectively.

Three or more learners are *homopolar* or *homologous* if they are pairwise homopolar or homologous, respectively.

Heterology. Two losses or learning processes are called *heteropolar* iff they have different polarities. The processes are called *heterologous* iff they are not homologous. (Thus they are either heteropolar or they operate on different input domains.)

Two learners are called *heteropolar* or *heterologous* iff there is a learning process in one of them that is heteropolar or heterologous, respectively, to every learning process in the other learner.

Note that the above definition applies to individual losses. In the case of a synergetic network with a composite loss function $\ell(\ell_1, \dots, \ell_{\omega+1})$ and learning rate λ , we can define $\lambda_i = \lambda \frac{\partial \ell}{\partial \ell_i}$, for $i = 1, \dots, \omega + 1$, and apply the definition as if the losses were individual.

The definitions of loss and learning polarity may seem peculiar at first but there is a justification for them. In general, we should not define learning polarity based on learning rates only because minimizing a loss ℓ with a learning rate λ is equivalent to maximizing $-\ell$ with learning rate $-\lambda$.* Thus the polarity of a learning process is effectively the sign of the product $\lambda\ell$. This coincides with our definition when the loss ℓ does not change sign.

However, a definition of learning polarity as $\text{sign}(\lambda\ell)$ is problematic when ℓ can be zero or change sign. In the later case, negative values do not signify a reversion of polarity in the loss function. We effectively cannot tell whether $-\ell$ or ℓ is the negated form of the intended loss. As a result, we must resort to a convention for such losses. With definition 4.9, it is $p^\pm(\ell) = p^\pm(-\ell) = +1$, when ℓ can take both negative and positive values. Polarity is inherently different for such losses. For example it is not multiplicative and it does not depend on the sign of numeric constants.

We can understand the role of polarity better if we examine the mechanics of gradient descent in the context of synergetic networks.

We assume a synergetic network $N_{\omega+1}$ that contains ω subnetworks $N_1, \dots, N_\omega$ and employs losses $\ell_1, \dots, \ell_{\omega+1}$ and learning rates $\lambda_1, \dots, \lambda_{\omega+1}$. We also assume u to be a parameter that appears only in network N_i for some $i \in \{1, \dots, \omega\}$. We furthermore assume that $\ell_{\omega+1} = 0$ or $\lambda_{\omega+1} = 0$ in order to avoid interactions between ℓ_i and $\ell_{\omega+1}$ on u . For simplicity, we assume that all losses (except $\ell_{\omega+1}$) have positive polarity and they do not change sign.

For individual losses, it is $\Delta u = (\Delta u)_i = -\lambda_i \frac{\partial \ell_i}{\partial u}$. Assuming λ_i to be sufficiently small, we get

$$\Delta \ell_i \simeq \frac{\partial \ell_i}{\partial u} \Delta u = -\lambda_i \left(\frac{\partial \ell_i}{\partial u} \right)^2.$$

Therefore, $\Delta \ell_i < 0$ iff $\lambda_i > 0$. As a result, when $\lambda_i > 0$, the update $\Delta u = -\lambda_i \frac{\partial \ell_i}{\partial u}$ corresponds to gradient descent on ℓ_i ; but when $\lambda_i < 0$, it

*This is not true for all learning algorithms but it applies to gradient descent and the algorithms that we will consider in this theory.

is gradient ascent. In other words, positive and negative polarity indicate whether we minimize or maximize the corresponding loss function, respectively.

Similarly, if we assume a composite loss $\ell(\ell_1, \dots, \ell_\omega)$ with learning rate λ , it is $\Delta u = -\lambda \frac{\partial \ell}{\partial u} = -\lambda \frac{\partial \ell}{\partial \ell_i} \frac{\partial \ell_i}{\partial u}$, since u appears only in N_i . Consequently, it is

$$\begin{aligned}\Delta \ell &\simeq \frac{\partial \ell}{\partial u} \Delta u = -\lambda \left(\frac{\partial \ell}{\partial u} \right)^2 \quad \text{and} \\ \Delta \ell_i &\simeq \frac{\partial \ell_i}{\partial u} \Delta u = -\lambda \frac{\partial \ell}{\partial \ell_i} \left(\frac{\partial \ell_i}{\partial u} \right)^2.\end{aligned}$$

Therefore, the signs of λ and $\lambda \frac{\partial \ell}{\partial \ell_i}$ and thus the polarities of the corresponding learning processes indicate whether we maximize or minimize those losses.

In general we can prove the following lemma.

Lemma 4.3. *If two learning process are homopolar they apply the same optimization (minimization or maximization) to homopolar losses and opposite optimizations to heteropolar losses. The opposite is true for heteropolar processes.*

Proof. Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be two learning processes with losses ℓ_1 and ℓ_2 and learning rates λ_1 and λ_2 .

Now, if the processes are homopolar it is $p^\pm(\mathcal{L}_1) = p^\pm(\mathcal{L}_2)$. Thus,

$$\text{sign}(\lambda_1) = \text{sign}(\lambda_2) \frac{p^\pm(\ell_2)}{p^\pm(\ell_1)} = \begin{cases} \text{sign}(\lambda_2) & \text{if } \ell_1 \text{ and } \ell_2 \text{ are homopolar,} \\ -\text{sign}(\lambda_2) & \text{if } \ell_1 \text{ and } \ell_2 \text{ are heteropolar.} \end{cases}$$

Consequently, $\mathcal{L}_1$ performs the same optimization as $\mathcal{L}_2$ when ℓ_1 and ℓ_2 are homopolar but opposite optimizations when the losses are heteropolar.

Similarly, when the processes are heteropolar, it is $p^\pm(\mathcal{L}_1) = -p^\pm(\mathcal{L}_2)$ and

$$\text{sign}(\lambda_1) = \begin{cases} -\text{sign}(\lambda_2) & \text{if } \ell_1 \text{ and } \ell_2 \text{ are homopolar,} \\ \text{sign}(\lambda_2) & \text{if } \ell_1 \text{ and } \ell_2 \text{ are heteropolar.} \end{cases}$$

□

In general, we can partition the losses and the learning processes of a synergetic network into two groups based on their polarity. The above lemma then implies that the positive polarity processes minimize the positive polarity losses and maximize the rest. The negative polarity processes do the opposite.

Homology or agreement in polarity imply structural and learning similarities. However, they do not imply that the updates of the learned parameters are consistent. There can always be interference among loss functions. For example, in our synergetic network, $\ell_{\omega+1}$ overlaps with $\ell_1, \dots, \ell_\omega$ since $N_{\omega+1}$ contains the networks $N_1, \dots, N_\omega$ and their learnable parameters. Consequently, if $(\Delta u)_i (\Delta u)_{\omega+1} < 0$, those two updates interfere with each other and

they may change u in either direction depending on their relative magnitude. Such interference does not necessarily derive from polarity. For example, both ℓ_i and $\ell_{\omega+1}$ may have positive polarity but they may not admit the same or even similar optimal values for the parameter u . In that case, there can be regions in the parameter space of the network where one loss will increase u and the other will decrease it.

The same applies between losses ℓ_i and ℓ_j for $i \neq j$, if the networks N_i and N_j contain some common learning parameters. This is not unlikely because, for example, N_j could be another synergetic network and it may contain N_i as a subnetwork.

In short, the learning in a synergetic neural network implies a distribution of interfering and compatible updates for each learnable parameter. Homopolar or homologous learners may result in more compatible updates on average but there will always be a distribution. It is an essential part of learning.

In current neural network literature, the term adversarial is used for heteropolar learners and learning processes. However, we opted against using the term because the ‘adversarial’ networks have a synergetic relationship during training. They cannot learn on their own. They depend on each other. They are similar to symbiotic organisms that may compete for resources but cannot survive without each other.

In general, heteropolar learners can have various forms and degrees of synergy despite their seemingly adversarial nature. We will see an example in section 4.8.

4.7 Cooperative Game Theory

In mathematics, *games* are entities that consist of four elements: the *players* of the game, the *information* available to the players at each decision point, the *actions* that each player can take at each decision point, and the *payoff* for the outcome of each action or set of actions. (Those elements are sometimes represented by the acronym “PAPT”.) *Game theory* is the branch of mathematics that studies games.

In general, players of a game use the available information to take actions that would maximize their overall payoff. Thus, in a typical game, each player acts on their own and they are primarily concerned with maximizing their payoff.*

In contrast, *cooperative game theory* studies *cooperative games*, which are games where players can form *coalitions* in order to maximize the collective payoff of their coalition as well as their own payoff. Formally, a cooperative game between a set of players P contains a *value function* $v: 2^P \rightarrow \mathbb{R}$ that maps coalitions, which are subsets of players and thus members of the power

*Sometime a cost function is used instead of a payoff and the players are concerned with minimizing their cost. However, the versions are equivalent if we consider cost to be negative payoff.

set 2^P , to real numbers and assigns no value to the empty coalition. Hence, for any coalition $C \subseteq P$, $v(C)$ is its value or payoff and $v(\emptyset) = 0$.

Cooperative games often contain a method to distribute payoffs to players based on their contributions to all possible coalitions. A common form of such a distribution is the *Shapley value* that averages the value that each new player adds to any given coalition

$$\begin{aligned}\phi_i(v) &= \frac{1}{p} \sum_{c=0}^{p-1} \frac{1}{\binom{p-1}{c}} \sum_{\substack{C \subseteq P - \{i\} \\ |C|=c}} v(C \cup \{i\}) - v(C) \\ &= \frac{1}{p} \sum_{C \subseteq P - \{i\}} \frac{v(C \cup \{i\}) - v(C)}{\binom{p-1}{|C|}},\end{aligned}\tag{4.11}$$

where ϕ_i is the estimated payoff for player i , $p = |P|$ is the total number of players, $\binom{n}{m} = \frac{n!}{m!(n-m)!}$ is the *binomial coefficient* (the number of m -combinations of n distinct items), C represents any coalition that does not contain player i , $c = |C|$ is the number of players in the coalition C , and, of course, $v(C \cup \{i\}) - v(C)$ is the value that player i adds to the coalition when inserted as a member into C .

Cooperative game theory is relevant for synergetic learning because the learners can be seen as players forming a coalition with the goal of maximizing the performance of the coalition, which corresponds to minimizing some form of a loss function. Estimated payoffs like the Shapley value can be used to estimate the importance of each learner – albeit there are various issues this approach that we will discuss in the next section and later in the theory.

4.8 Examples of Synergetic Learning

We can better understand synergetic networks and their synergy and learning with some examples. In this section, we consider synergetic generalizations for ensemble learning and generative networks, too. The latter invariably appear as generative adversarial networks, or GAN, in neural network literature.

Both ensemble and adversarial networks are minimally synergetic. With a possible exception for simple tasks or small data sets, it is unlikely that optimal (or nearly optimal) solutions are present at such low levels of synergy. On the contrary, optimality and competitive performance most likely require a greater and more elaborate form of synergy. When entities grow in an environment, and they must handle harder and more complex learning tasks, the level of required synergy can only increase.

4.8.1 Ensembles

In physics and particularly in thermodynamics and statistical mechanics, an *ensemble* is a set that usually contains a large or infinite number of instances of

a system or entity that is often represented by one or more random variables. In machine learning, a *learning ensemble* is usually a smaller and certainly finite set of learners. The outputs of those learners, which are called *ensemble members*, are then combined to generate the output of the ensemble.

There are multiple forms of ensembles. One of the simplest is averaging the output of the learners, assuming that they all answer the same question. A linear combination of the outputs is also a common scheme. The coefficients of each learner L are sometimes computed using probabilistic methods, such as the probability $P(D/L)$ that the data set D was generated by a relationship that is accurately represented or approximated by L . A better approach is to learn the coefficient along with the rest of the parameters of the learners. This is a linear version of *model* or *learner aggregation* where the outputs of a set of learners are inputs into other learners that aggregate those sub-learners.*

For our purposes, aggregating through learning is a better method than probabilistic approaches. It is more general. It is not bound to linear forms. It can potentially capture arbitrary and non-monotonic relationships between the aggregated learners. Its disadvantage is potentially increased demand for training data – but this can be addressed by using smaller or simpler learners instead of using probabilistic methods.

Such aggregation does not require the learners to have compatible outputs, operate on the same input, or even try to solve the same problem. For example, learner aggregation has been used to combine speech recognition with lip-reading in order to improve the overall accuracy of the speech system. It has also been used to aggregate handwriting recognition with *handwriting personalization*, which represents internal adaptation of a handwriting system to individual handwriting styles.

The various parts of an ensemble can be trained on the same data set, overlapping or distinct data sets, or any combination these. For example, *bagging* is an ensemble learning method that generates training data sets, one for each member of the ensemble, by sampling an original training set (with *replacement*, which means that each sample can appear multiple times in each member data sets). Consequently, each produced training set will most likely have different density in some areas of the input space, compared to the overall training set. In areas where its density is higher, the corresponding training set will effectively have a different emphasis but within the constraints of sampling and variation of the original data set. The associated learner may thus be similarly biased. As a result, bagging randomly distributes emphasis across the input space and the members of the ensemble relative to the distribution of a training set. Although such a random distribution is a good initial approach or baseline, it is probably not as effective or specialized as a distribution generated by feedback from synergetic components.

Ensembles usually have a minimal form of synergy but they may include

*The term *mixture of experts* and the less elegant term *stacked generalization* have also been used to describe this scheme, albeit the former is limited to linear aggregation only.

unidirectional feedback. For example, *boosting* is an ensemble learning method where member learners are trained in sequence and the training samples for each learner are selected or weighted based on the performance of the previous members of the ensemble so far. There are many variants of boosting based on how emphasis (weighting) is distributed on training samples and member learners. In general, samples with higher error rates get larger weights and they bias the remaining learning process to place more value on reducing their loss.

In some cases, such as *gradient boosting*, each member learner is trained on the *residual error* of the ensemble so far, which is the negative of the derivative of the loss function for the ensemble with respect to the output of the ensemble. Then the new member is added linearly to the ensemble so that the overall loss is minimized.

The overall intuition behind learning ensembles is that two learners can be better than one. In human affairs, the principle is quite common. For example, people may seek a “second opinion” before making a medical, financial or other important decision.

However, learning ensembles also include a counterintuitive and limiting aspect. They only use the output of their constituent learners. There is not a deeper or more detailed form of sharing and synergy. There is often no feedback; and when there is, it is static and unidirectional, from one learner to the next, based on the errors of the previous one. Learners cannot interact with each other in any other way. They cannot relax their performance on certain data points when subsequent learners perform better on them. Learners only operate once and they cannot change their ‘mind’ (state and output) after they have operated. They do not have a shared state. From a collective viewpoint, it is as though they have no state at all.

The equivalent in human affairs would be to ask for advice but never ask for the rationale behind the advice, never discuss anything or allow for the advisers to revise their suggestion, never backtrack or reconsider, and always move in one direction, from one adviser the next, no matter what happens with the next adviser.

Of course, synergy does not have to be so strict or limited. We can easily create synergetic versions of ensembles. For example, an ensemble that computes the average for neural networks $N_1, \dots, N_\omega$, can easily be transformed into a synergetic network $N_{\omega+1}$ if we add state to each network N_i for $i = 1, \dots, \omega$, and define the overall network and its synergy as follows

$$N_{\omega+1} = \frac{1}{\omega} \sum_{i=1}^{\omega} N_i$$

$$\prod_{i=1}^n N_i \triangleright N_{\omega+1}.$$

We implicitly assume r repetitions over the synergy with some initial incoming

state θ (which can be $\emptyset$ or any default value) but we omit them for simplicity. Such parameters should always have default values when not specified.

Equally straightforward is the case of learner aggregation. We add states to the networks $N_1, \dots, N_{\omega+1}$ and require aggregation and synergy

$$N_{\omega+1}(N_1, \dots, N_{\omega}) \\ \prod_{i=1}^n N_i \triangleright N_{\omega+1},$$

where $N_{\omega+1}$ is the aggregating network and $N_1, \dots, N_{\omega}$ are the aggregated ones.

Similarly for bagging, boosting, or any other ensemble. In all cases, sequential synergy, such as $\sum_{i=1}^n N_i \triangleright N_{\omega+1}$, or any combination of sequential and parallel synergy can be used instead of purely parallel. This may be more appropriate when the networks are trained in sequence or they depend on each other as, for example, in boosting ensembles.

Note that we can actually pick the output of each network N_i as its state, if we wish. This is probably not optimal for real applications or large data sets but there is nothing to prevent such a choice if it is somehow indicated.

4.8.2 Generative Networks

In section 2.3, we briefly described adversarial approaches, which correspond to a special case of heterologous learning in our theory. A popular method is generative adversarial networks, or GAN, that are often used to learn a distribution and generate samples from that distribution. It is often used to generate realistic images similar to the images of a given data set.

A GAN usually consists of two networks, the *generator* that learns to generate samples from latent or random input, and the *critic* that learns to distinguish between actual and generated samples. The critic usually tries to minimize a loss function that measures the error of its prediction while the generator tries to maximize it.

More specifically, let N_c and N_g be the critic and generator networks, respectively. Let Z represent the latent input to the generator and X represent the input to the critic from a data set $\mathcal{D}$. In other words, $N_g(Z)$ is the output of the generator upon latent input Z , $N_c(N_g(Z))$ is the output of the critic upon input that was generated by the generator, and $N_c(X)$ is the output of the critic upon input $X \in \mathcal{D}$. The output of the critic is supposed to be in the interval $[0, 1]$, with 0 indicating a generated sample and 1 indicating a sample from $\mathcal{D}$. When the input to the critic comes from the generator, the target value is 0. When it comes from the data set $\mathcal{D}$, the target is 1. If ℓ is a loss function for the predictions of the critic, then the critic tries to minimize ℓ while the generator tries to maximize it. For example, if we use cross-entropy, the loss for a generated sample from latent input Z is $-\ln(1 - N_c(N_g(Z)))$ while the loss for an actual sample $X \in \mathcal{D}$ is $-\ln(N_c(X))$. Thus the overall

loss is

$$\ell = - \sum_Z \ln(1 - N_c(N_g(Z))) - \sum_{X \in \mathcal{D}} \ln(N_c(X)). \quad (4.12)$$

The second sum does not depend on the parameters of N_g . Thus, N_g effectively tries to maximize the first sum only.

(Note that when $N_c(X) = 0$ or $N_c(N_g(Z)) = 1$ the loss is infinite. However, cross-entropy is often used for networks with output in the open interval $(0, 1)$, as is the case when a softmax or softmin layer are used before the loss is calculated. Limited numerical precision can still produce an output of 0 or 1 through softmax or softmin (e.g. when one input value to softmax is much larger or smaller than the rest) but there are numerically stable methods that combine the softmax/softmin layer with cross entropy. This is based on the fact that, when the output is of a softmax form $z = e^y / \sum_x e^x$, then $-\ln(z) = -y + \ln(\sum_x e^x)$. Similarly for softmin.

Despite their name, there is an obvious synergy in such networks. N_g provides its output to N_c and the output of N_c is used to generate the loss and thus training derivatives for N_g (and itself, too).

However, the synergy of the two networks is minimal and arguably suboptimal if not counterproductive. Indeed the two networks develop rich internal representations in order to be able to capture the underlying distribution of the data set $\mathcal{D}$. For example, when GAN are used for the generation of images, the networks must develop complex internal representations so that they can generate or accurately detect realistic images. Yet, the networks do not share their representations or any of their elements. They only share their output; in the case of the critic, this is a single number!

We can furthermore understand how suboptimal the situation is if we elevate it to the level of human affairs and draw an analogy with software engineering. We can imagine a software engineer that authors the code for an application and a tester that tests the functionality of the code. The tester then finds and reports a glitch but refuses to provide any information about the glitch, such as its description, effect, location, and reproducing steps. Despite the counterproductive attitude, there is still some synergy between the engineer and the tester, as there is synergy between the generator and the critic, but it is so limited and minimal that the underlying task is made unnecessarily hard.

We use the term *generative heterology* to refer to *generative heterologous networks* (GHN), which are heterologous networks (of arbitrary synergy) that are used for generative purposes.

The simplest and most straightforward improvement is to allow the critic N_c to share internal information with the generator N_g . We can achieve that goal if we add state to N_c (but not incoming state), add incoming state to N_g , and create a synergetic network so that N_g can receive s_c as incoming

state:

$$\begin{aligned} & \left\{ \begin{array}{l} N_c \circ N_g \\ N_g \triangleright N_c \end{array} \right\} && \text{for latent input } Z, \\ & \left\{ \begin{array}{l} N_c \\ \text{no synergy} \end{array} \right\} && \text{for input } X \in \mathcal{D}. \end{aligned} \quad (4.13)$$

The loss function remains the same as in (4.12).

Of course, the above synergy requires at least one repetition so that the state of the critic N_c can be passed to the generator N_g . In other words, the generator gives the critic a generated sample and receives the critic's state as feedback about its sample. Thus the generator can improve its sample during repetitions and receive updated feedback.

A typical choice for state in N_c would be a periphery of a complete input-output cut near the output.

Although the above synergetic network is an improvement over GAN, it is still limited in synergy. The generator does receive feedback from the critic but it remains a black box for the critic.

We can increase the synergy if we add state to both networks and mutual feedback. However, when the input is a real sample $X \in \mathcal{D}$, we do not have an input Z for N_g and thus we cannot run the generator and compute its state. If we provide a random or fixed input Z , then $N_g(Z)$ and s_g will be irrelevant to X . Thus we have to use synergy only when we operate the networks on latent input Z . On the other hand, when the input is $X \in \mathcal{D}$, the incoming state to N_c cannot always be $\emptyset$ or any default value because then the critic will know that a sample is generated when the incoming state is different than the default. Thus we have to use a random incoming state θ :

$$\begin{aligned} & \left\{ \begin{array}{l} N_c \circ N_g \\ N_g \triangleright N_c \end{array} \right\} && \text{for latent input } Z, \\ & \left\{ \begin{array}{l} N_c^\theta \\ \text{no synergy} \end{array} \right\} && \text{for input } X \in \mathcal{D}. \end{aligned} \quad (4.14)$$

We can increase the synergy further and also get rid of the compromise of randomness in the incoming state θ , if we introduce an auto-encoder that learns the identity mappings $(N_i \circ N_g)(Z) = Z$ for latent input Z and $(N_g \circ N_i)(X) = X$ for $X \in \mathcal{D}$. We call N_i the *inverse network* of the generator N_g because it computes the inverse mapping. We can then use $N_i(X)$, for $X \in \mathcal{D}$, to derive a latent input Z that can generate X or an approximation of it. In its simplest form, N_i has no internal or incoming state; but its state

is analogous to the state of N_g and it does not affect the synergy formulas:

$$\begin{aligned} & \left\{ \begin{array}{l} N_i \circ N_g, N_c \circ N_g \\ N_g \parallel N_i \triangleright N_c \end{array} \right\} && \text{for latent input } Z, \\ & \left\{ \begin{array}{l} N_g \circ N_i, N_c \\ N_i \parallel N_g \triangleright N_c \end{array} \right\} && \text{for input } X \in \mathcal{D}. \end{aligned} \quad (4.15)$$

The loss function is still the same and as defined in equation (4.12).

The above examples of generative heterology as well as the synergetic ensembles of the previous subsection demonstrate the importance of using an algebra or a proper formalism. For we can then describe elaborate structure and synergy with concise expressions. Algebra does unite the ‘broken parts’ as its name suggests and it does so in a concise and effective manner.

4.9 Diversity

Diversity is the condition that the elements of a set or system tend to differ from each other. It is a broad term that is used in various forms in science, technology, politics, and business.

In biology, diversity is so fundamental that it is studied at various levels and forms from molecular levels to taxonomic ranks and even entire ecosystems (thus the concepts of nucleotide diversity, genetic diversity, phylogenetic diversity, ecosystem diversity, biodiversity, etc.).

Biodiversity measures the variety and variability of life forms within an ecosystem while *genetic diversity* and *genetic variability* refer to the total number of genetic characteristics in a species and the variability of those characteristics, respectively.

Diversity is broadly evident and directly observable in nature. Even minimal ecosystems contain so many life forms that it takes considerable effort and resources to identify and list all (or even most) of them.

On an abstract level, diversity is a consequence of the fact that life is a synthesis. If an organism consists of billions or trillions of cells, the potential differentiations and combinations of those cells is a quantity beyond practical measure. Even if we limit ourselves to cells that are organized into tissues, organs, systems, and viable organisms, the combinations are still exponential and astronomical. The same applies when we exclusively constrain our estimates to life forms that are fit for their environment.

Indeed, there is so much diversity in nature that eight nested taxonomic ranks are used in biology to classify species as shown in tables 4.1 and 4.2. The total number of species is unknown but there are millions of catalogued extant species and there are probably billions in total. (Most estimates of the table have a degree of subjectivity, especially in middle ranks, and there is ongoing debate about the number of classifications.)

	Human	Red fox	E. bumble bee	Olive tree
Domain	Eukarya	Eukarya	Eukarya	Eukarya
Kingdom	Animalia	Animalia	Animalia	Plantae
Phylum	Chordata	Chordata	Arthropoda	Tracheophyta
Class	Mammalia	Mammalia	Insecta	Asteridae
Order	Primates	Carnivora	Hymenoptera	Lamiales
Family	Hominidae	Canidae	Apidae	Oleaceae
Genus	Homo	Vulpes	Bombus	Olea
Species	H. sapiens	V. vulpes	B. impatiens	O. europaea

Table 4.1: Examples of taxonomic biological classification.

However, synthesis only provides a basis for diversity. It does not explain the cause of it. In the field of *genetics*, there are several hypotheses or theories about the causes of genetic diversity. *Frequency-dependent selection* asserts that high frequency *alleles*, which are variant forms of genes, increase vulnerability given that a pathogen can spread if it can overcome such an allele. *Diversifying selection* hypothesizes that subpopulations of a species may live in different environments or locations that select for different alleles. The *neutral theory of evolution* proposes that most of the variation at a molecular level is driven not by selection or fitness but rather by genetic drift. Recent developments suggest a more elaborate stochastic scheme of *linked selection*, where beneficial alleles are linked to neighboring neutral mutations and they are all selected as a group. This is effectively stochastic selection. Neutrality can accumulate diversity over time but linked selection can generate even more variation.

The above hypotheses induce a distribution of variation and their common effect is that alleles tend not to concentrate but diversify through change, chance, and evolution.

There are other factors that affect and often increase diversity, such as sexual reproduction or breeding, *gene flow* (which is movement of genetic material by bird migration or pollen in the wind, for example), beneficial or harmful random mutation, *exaptation* and *vestigiality* (which respectively refer to functional shift of a trait through evolution and retention of traits or structures that have lost some or all of their initial functionality*), large populations, and environmental changes. Across phyla or classes, functionality can also be a factor. For example, as it can be calculated from table 4.2, the ratio of the number of phyla between animals and plants is about $35/14 = 2.5$. However, the corresponding ratios for genera and species are about 8.3 and

*The distinction between exaptational and vestigial structures can be subjective. For example, the wings of a penguin are clearly vestigial as structures for flight but they can be seen as exaptational as structures for underwater locomotion. In general, vestigiality tends to increase variability.

Rank	Size
Domain	Three domains: eukarya, bacteria, and archaea.
Kingdom	Seven kingdoms: animalia, plantae, fungi, chromista, protozoa, bacteria, and archaea.
Phylum	About 190 known phyla, about 35 phyla for animalia, 14 for plantae, 18 for fungi (plus 23 subphyla), 8 for chromista, 8 for protozoa, 92 for bacteria, and 26 for archaea.
Class	More than 300 classes, about 94 to 108 for animals, 33 for plants, 74 for fungi, and 82 for chromista.
Order	More than 900 orders, approximately 450 for animals, 177 for plantae, and 215 for fungi.
Family	At least 8000 families, approximately 5800 for animalia, 800 for plantae, 731 for fungi, 360 for chromista, and 310 for protozoa. (There are at least 327 catalogued families for archaea and bacteria but no reliable estimate for the true number. The numbers have been increasing exponentially.)
Genus	About 352,000 genera in publication and databases, approximately 239,100 for animalia, 28,700 for plantae, 10,500 for fungi, 11,100 for chromista, 3100 for protozoa, at least 60,000 for bacteria and archaea.
Species	Millions (possibly billions) extant species on earth, about 5 to 10 million animal and plant species (about 7,770,000 for animalia, 350,700 for plantae, 611,000 for fungi, 27,500 for chromista, 36,400 for protozoa, and at least 200,000 for bacteria and archaea.

Table 4.2: Taxonomic ranks of biological classification.

22.1, respectively. This is not surprising. In general, animals can perform more complex and advanced functions, such as movement and perception, have organs and more systems to support their functionality, and thus appear in more forms.

Diversity combined with natural selection affects the chance that a population can adapt and survive significant or severe changes in the environment. Diversity thus becomes a measure of survivability. The greater the genetic diversity of a population, the more likely the population is to adapt and survive. The same can apply not only to species but also to entire ecosystems.

Most of the above factors can be extended to learning and nervous systems. For example, frequency-dependent selection can be assumed as a method to make a learning system less likely to focus on fewer common components and overfit the data. Diversifying selection can apply when we train the components of a synergetic network on different or disjoint data sets. Genetic

drift is random sampling and thus *neural drift* can be defined likewise for neurons. Random mutation, the neutral theory of evolution, and linked selection can also be applied to neurons and learning. Of course, evolution must take place and some form of heredity is required, if changes are to transfer to other learners. The same is true for reproduction as a source of diversity. Although we have not yet introduced such elements to the learning processes of this theory, we consider them to be indispensable principles of learning and we will progressively address them in later stages of the theory.

The case for diversity in the foundations of life is supported by strong evidence and extensive study. The above factors and considerations may suggest that diversity is also a principle of learning. Moreover, there is a supplemental and perhaps crucial justification that we present in the next two subsections.

4.9.1 Voting and Condorcet's Theorem

A simple ensemble scheme for classification problems is the *voting ensemble*, where the output of each learner in the ensembles is used to pick the winning class for that learner and then the learner 'votes' for that class. In the end, the class with the most votes is the output of the aggregate learner.

An interesting theorem on voting, assuming independence of the voters, appeared in 1785.

Theorem 4.1 (Condorcet's Theorem). *If $X_1, \dots, X_n$ are independent individual voters in a yes/no election and each voter makes the correct decision with probability $p > \frac{1}{2}$, then the probability that a majority of the voters make the correct decision tends to 1, as $n \rightarrow \infty$.*

Intuitively, Condorcet's theorem states that if voters have a higher probability of making the right decision, then large populations of independent voters are almost guaranteed to make the right decision.

Instead of proving the above theorem, we will prove its more abstract version below.

Theorem 4.2. *If $X_1, \dots, X_n$ are independent and identically distributed (i.i.d.) binary random variables (with values 0 and 1), $\text{Prob}(X_i = 1) = p > \frac{1}{2}$ for $i = 1, \dots, n$, and $M_n = \frac{1}{n} \sum_{i=1}^n X_i$ then*

$$\lim_{n \rightarrow \infty} \text{Prob}(M_n > \frac{1}{2}) = 1.$$

Proof. Indeed, if we pick $\epsilon = p - \frac{1}{2} > 0$, and observe that $EM_n = EX_i = p$ for $i = 1, \dots, n$ we get

$$\begin{aligned} \text{Prob}(M_n > \frac{1}{2}) &\geq \text{Prob}(0 < M_n - \frac{1}{2} < 2\epsilon) \\ &= \text{Prob}(-\epsilon < M_n - p < \epsilon) \\ &= 1 - \text{Prob}(|M_n - p| \geq \epsilon) \xrightarrow{n \rightarrow \infty} 1. \end{aligned}$$

In the last step, we used the weak law of large numbers, which states that

$$(\forall \epsilon > 0) \quad \lim_{n \rightarrow \infty} \{\text{Prob}(|M_n - EM_n| \geq \epsilon)\} = 0.$$

□

We can furthermore prove (as a direct application of *Cramèr's theorem*) the following lemma.

Lemma 4.4. *If $X_1, \dots, X_n$ are independent and identically distributed binary random variables (with values 0 and 1), $\text{Prob}(X_i = 1) = p > \frac{1}{2}$ for $i = 1, \dots, n$, and $M_n = \frac{1}{n} \sum_{i=1}^n X_i$ then*

$$\text{Prob}(M_n \leq \frac{1}{2}) < 2e^{-2n\epsilon^2}, \quad (4.16)$$

where $\epsilon = p - \frac{1}{2} > 0$.

Proof. Cramer's theorem implies that, for any $\delta > 0$, it is

$$\text{Prob}(|M_n - p| \geq \delta) < 2e^{-2n\delta^2}. \quad (4.17)$$

As a result,

$$\begin{aligned} \text{Prob}(M_n \leq \frac{1}{2}) &\leq 1 - \text{Prob}(0 < M_n - \frac{1}{2} < 2\epsilon) \\ &= 1 - \text{Prob}(-\epsilon < M_n - p < \epsilon) \\ &= \text{Prob}(|M_n - p| \geq \epsilon) \\ &< 2e^{-2n\epsilon^2}. \end{aligned}$$

□

Lemma 4.4 provides a striking bound on the error of a voting ensemble. Indeed, if we assume that 1 represents a correct classification and 0 represents a misclassification, a voting ensemble would commit a misclassification when $M_n \leq \frac{1}{2}$. As a result, the error of the ensemble is $\text{Prob}(M_n \leq \frac{1}{2})$. Lemma 4.4 thus implies that the error decreases exponentially with the number of the learners in the ensemble.

For example, if we have 70 voting learners, each with a classification error of 30%, their voting ensemble will have a classification error that is less than 0.74%, given that $p = 1 - 0.3 = 0.7$ and $\epsilon = 0.7 - 0.5 = 0.2$. This corresponds to a relative reduction of at least $\frac{0.3 - 0.0074}{0.3} \simeq 97.53\%$. Furthermore, if we double the number of learners $n = 140$, then the error is reduced to less than 0.0027% and the error reduction is at least 99.99%.

Such reductions are virtually impossible in practice. Indeed, if lemma 4.4 could apply, learning would be such an easy task. We could just use a few hundreds of very weak learners in a voting ensemble and the ensemble would have error less than 1%.

Reality is not so simple or straightforward, though. There is a serious limitation in lemma 4.4 and Condorcet's theorem as well. They both assume

that the voting learners are independent. This may appear as an innocuous or reasonable assumption but it is not. It is actually so fundamentally flawed that it implies diminished error bounds as those of lemma 4.4. Statistical independence is largely incompatible with nature and reality. In the next subsection, we will attempt to describe and understand this condition better.

4.9.2 A Theoretical Justification for Diversity

In large systems, environments, or ecosystems as well as in complex learning domains and tasks, there is a great deal of dependencies that find their way into the learners.

Some of those dependencies come from the data and the environment. If learners operate in similar environments they often share data sets, samples, or experiences by necessity, norm, or coincidence. As a result, implicit, non-causal dependencies and correlations can emerge.

Furthermore, if the learners try to solve similar or identical problems, their low level features would most likely overlap. For example, edge detector features are most likely present in the early layers of all vision systems and they are probably similar or identical. Even higher level features, such as the presence of a beak or a crest, and their relationship to encompassing features, like the presence of a bird, are similarly correlated across learners. Even if the learners use somewhat different logic, there is not enough logical structure to support hundreds of independent learners.

In classification problems, classes can be defined by their *decision boundary*, which is the set of points that separate a class from other surrounding classes. (Classes usually consist of contiguous segments as we shall explain in chapter 5.) In general, samples near the decision boundary have a higher chance of misclassification. Conversely, samples further from the decision boundary are less likely to be misclassified. For those samples, the learners could be highly correlated. Independence is unlikely or impossible to hold, especially as the number of learners grows.

Furthermore, the *Bayes error rate* defines the minimum error rate for any classifier in a given domain $\mathcal{X}$. It is derived from the probability $P(C/X)$ that a given sample X belongs to class C . It corresponds to a classifier that predicts the most likely class for each sample and it is thus defined as

$$1 - \int_{\mathcal{X}} \max_c P(C = c/X = x) f_X(x) dx.$$

As a result, if $\max_c \{P(C = c/X = x)\} < 1$ in a measurable set S where $\int_S f_X(x) dx > 0$, the Bayes error is positive. However, Condorcet's theorem implies that the error rate goes to zero, if there are infinitely many independent classifiers with error rate less $p < 0.5$. Consequently, in any domain with positive Bayes error, the number of independent classifiers with error rate less or equal to $1/2 - \delta$ is finite, for all $\delta > 0$. (Lemma 4.4 implies that the

number n of independent classifiers with error probability $p < 1/2$ is bound by $\frac{\ln 2 - \ln b}{2\epsilon^2}$, where b is the Bayes error and $\epsilon = 1/2 - p$.)

There are other factors, like similarities in learning, structure, bias, or evolution that can generate correlations or dependencies. In the end, the overall effect is a potentially large number of such correlations and hidden, implicit, non-causal dependencies among the learners. Those invalidate Condorcet's theorem and lemma 4.4, too.

Yet, those theoretical results are not entirely in vain. They have practical significance because they provide useful intuitions. We can describe those intuitions better if we define a measure of dependence between random variables.

The *entropy* of a random variable X with density f_X is the expected value of $-\ln f_X$. Thus

$$H(X) = -E \ln f_X = - \int \ln(f_X) dF_X, \quad (4.18)$$

where F_X is the probability distribution of X .

The *mutual information* $I(X, Y)$ of two random variables X and Y with densities f_X and f_Y , respectively, is the relative entropy of the densities $f_{X,Y}$ and $f_X f_Y$. In other words,

$$I(X, Y) = E \ln \frac{f_{X,Y}}{f_X f_Y} = \int \ln \frac{f_{X,Y}}{f_X f_Y} dF_{X,Y}, \quad (4.19)$$

where $f_{X,Y}$ and $F_{X,Y}$ are the joint density and probability distribution of X and Y .

The strict convexity of the negative logarithm function and Jensen's inequality imply that

$$I(X, Y) = E \left(- \ln \frac{f_X f_Y}{f_{X,Y}} \right) \geq - \ln \left(E \frac{f_X f_Y}{f_{X,Y}} \right) = - \ln \int f_X f_Y = - \ln 1 = 0$$

with equality holding iff $\ln \frac{f_{X,Y}}{f_X f_Y}$ is constant, that is when $f_{X,Y} = f_X f_Y$ (since $\int dF_X = \int dF_Y = \int dF_{X,Y} = 1$).

Futhermore, it is

$$I(X, Y) - H(X) = E \ln \frac{f_{X,Y}}{f_Y} \leq E \ln(1) = 0.$$

Similarly, $I(X, Y) - H(Y) \leq 0$. Thus, $I(X, Y) \leq \min\{H(X), H(Y)\}$. And, of course,

$$I(X, X) = E \ln \frac{f_X}{f_X f_X} = H(X).$$

As a result, the mutual information $I(X, Y)$ is minimized when X and Y are independent and it is maximized when X and Y are identical. In other

words, it measures the level of dependence between X and Y .

We can furthermore define the *normalized mutual information* as

$$I_1(X, Y) = \frac{I(X, Y)}{\min(H(X), H(Y))} \quad (4.20)$$

so that its range is the interval $[0, 1]$ (the subscript 1 simply indicates that I_1 is less or equal to 1).

Note that the *covariance* $\text{cov}(X, Y) = E(X - EX)(Y - EY)$ of the variables X and Y can be zero in certain cases where X and Y are not independent. The same applies to the *correlation* $\text{corr}(X, Y) = \frac{\text{cov}(X, Y)}{\sqrt{\text{cov}(X, X) \text{cov}(Y, Y)}}$. Thus those are not as accurate measures of dependence as mutual information is.

The generalization of mutual information to more than two random variables is straightforward. The plain version is

$$I(X_1, \dots, X_\omega) = E \ln \frac{f_{X_1, \dots, X_\omega}}{f_{X_1} \cdots f_{X_\omega}} \quad (4.21)$$

and

$$I_1(X_1, \dots, X_n) = \frac{I(X_1, \dots, X_\omega)}{\min(H(X_1), \dots, H(X_\omega))}. \quad (4.22)$$

Now, we can return to our analysis and consider the various involved factors – the environment and the data, features, boundaries in feature space, relations of underlying concepts, learning processes and components, synergy, evolution, structure, etc. They all seem as though they ‘conspire’ to increase the levels of dependence – as measured by mutual information – among our learners and thus to frustrate our straightforward (and rather naive) effort for statistical independence and consequently our hopes for a learning triumph through voting.

In complex systems, independence is closer to wishful thinking than reality; however, a reduction or even some local minimization of dependence is not necessarily so. The corresponding error reductions would most likely be limited and they would certainly not be exponential, but they could be significant. If we seek a factor that can help us reduce dependence, the most obvious and perhaps most effective is diversity. The reason is simple. Similarities tend to increase statistical dependence while dissimilarities have the opposite effect.

More importantly, evolutionary processes do not operate on a basis of fixed or predefined dependencies. Instead, they apparently work with mechanisms that can manipulate distributions, including distributions of dependencies, and fit them to a given environment or population. Diversity has profound evolutionary significance because it is precisely such a mechanism that can be used to alter the amount of statistical dependence among a population of entities.

We consider the above to be an additional and theoretical justification for diversity in learning – and evolution as well.

4.10 The Synergy Conjecture

We hereby introduce the *synergy conjecture*, which implies that optimal synergy is expected to outperform independent learners or any other learner combination.

More specifically, given ω learners $N_1, \dots, N_\omega$ and an aggregation method $\mathcal{M}$ that generates learner $N_{\omega+1} = \mathcal{M}(N_1, \dots, N_\omega)$, we can consider the ratio of the error $\mathcal{E}(N_{\omega+1})$ of $N_{\omega+1}$ over the average error of $N_1, \dots, N_\omega$:

$$\mathcal{R}(N_{\omega+1}) = \frac{\omega \mathcal{E}(N_{\omega+1})}{\sum_i \mathcal{E}N_i}.$$

In one extreme case, when the learners $N_1, \dots, N_\omega$ are identical, their normalized mutual information is 1. In that case, the expected error ratio $\mathcal{R}(N_{\omega+1})$ should also be 1. Intuitively, the error rate cannot be reduced when identical models are combined. Of course, it can get worse if the aggregation method is defective. However, we assume that $\mathcal{M}$ is *effective* in the sense that the expected error ratio is always less or equal to 1 and that equality holds only when the learners are identical.

On the other hand, when the learners are statistically independent, their normalized mutual information is 0 while the expected value of the error ratio would usually be $R_0 < 1$ per Condorcet's theorem. This is certainly the case when the individual learners have constant error below 0.5, that is when $\mathcal{E}(N_i) = \mathcal{E}(N_j) < 0.5$, for all $i, j \leq \omega$.

Between those two extremes, and as the normalized mutual information $I_1(N_1, \dots, N_\omega)$ changes from 0 to 1, the error ratio should transition from R_0 to 1. The synergy conjecture states that the above transition is not monotonic but rather it is expected to contain at least two local extrema and three main segments. This postulated situation is depicted in figure 4.3.

The *detachment segment* is the leftmost segment. It corresponds to learners that do not cooperate well, often ignoring each other. At its left end, the normalized mutual information I_1 is zero, the learners are statistically independent, and they completely ignore each other or they operate without any information about each other. As the learners begin to cooperate or at least take each other into account, I_1 increases from 0 in that segment, and, we postulate, the error ratio tends to increase because the learners become more dependent. However, as I_1 and synergy increase further, the additional synergy begins to counter the negative effects of dependence. The error ratio reaches a local maximum and begins to decrease. The detachment segment ends when the error ratio reaches the value R_0 again. This is the point where synergy has the same overall effect on error as independence.

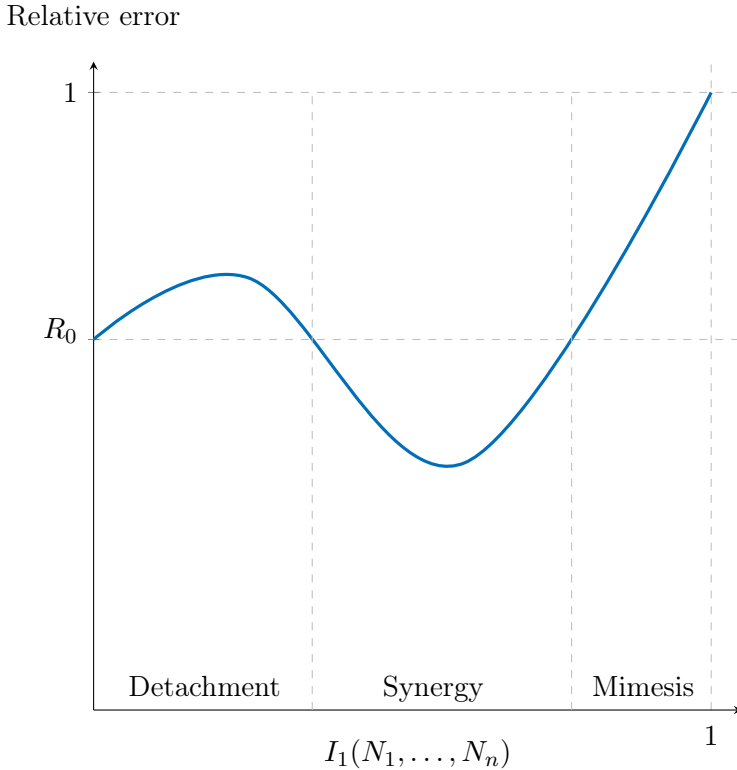


Figure 4.3: The synergy conjecture.

The *synergy segment* is the middle segment right after detachment. It represents the region where synergy outperforms independence. As I_1 and synergy increase in that segment, the error ratio tends to decrease until synergy reaches an optimal level. After that level, the learners start becoming too dependent, possibly imitating each other in some cases, and the error ratio begins to increase again. The synergy segment ends when the ratio returns to the value R_0 .

Finally, the *mimesis segment* is the third segment extending from the end of the synergy segment to the point of identical learners and $I_1 = 1$. The transition in that segment tends to be monotonic. As synergy and I_1 increase, the learners imitate each other more and the ratio tends to grow because the learners have little to add to each other. In the end, when $I_1 = 1$, the learners are identical, and the ratio reaches its maximal value.

Intuitively, the synergy conjecture derives from the premise that learners can operate better when they have knowledge about each other than when they have no such knowledge or they completely ignore each other. And if that is true, then there is an optimal level of such knowledge that defines the optimal synergy. This is a probabilistic argument. Individual exceptions are not as relevant. On average, synergy is expected to win.

In real situations, the error ratio curve may contain higher frequencies and various smaller local extrema. However, the overall trend should be as depicted in figure 4.3.

The synergy conjecture is consistent with the concept of the *price of anarchy*, which is used in game theory and economics to measure how the cost of a system increases (or its efficiency degrades) when its agents act selfishly based on their own interests only. It also relates to the broader concept of *equilibrium inefficiency* – although learning that is terminated after a number of training iterations does not necessarily correspond to an equilibrium.

Another important factor in favor of synergy is energy and resources. Independence is resource-intensive because it requires the training data and learning experiences to be fully or largely disjoint. Even if we assume that the optimal synergy has the same error ratio as independence, it would still have a significant evolutionary advantage because it requires fewer external resources. Its only cost is additional communication and processing to enable synergy. However, if neural state and the communicated information represent only a small fraction of the network, the overhead would most likely be insignificant.

4.11 Extrinsic Diversity

Definitions of diversity in machine learning literature (and particularly ensemble literature) are based on the outputs of learners. Those learners are usually classifiers. Probabilities or counts of their correct and incorrect classifications are used to create various measures of diversity. We collectively refer to such measures as *extrinsic diversity* because they are based on output instead of internal structure and constitution.*

Let us consider an example – along with related issues and complications.

When a synergetic network contains homologous learners with outputs that correspond to the same predictions, we can define a concept that measures the differences between the outputs of the learners.

Definition 4.10. Let $N_{\omega+1}$ be a synergetic network that contains learners $N_1, \dots, N_\omega$. Assume that the network has been trained on an unknown data set containing pairs (x, y) and that $N_i(x)$ can be treated as a prediction of y , for $i = 1, \dots, \omega + 1$. Then, if N_i and $N_{\omega+1}$ are homologous, the *dissent* of N_i on a data set S is defined as

$$D_{N_i}(S) = \frac{1}{|S|} \sum_{(x,y) \in S} (N_i(x) - N_{\omega+1}(x))^2.$$

Similarly, if N_i and N_j are homologous, their *disagreement* is defined as

$$D_{N_i, N_j}(S) = \frac{1}{|S|} \sum_{(x,y) \in S} (N_i(x) - N_j(x))^2.$$

*Behavioral diversity is an analogous but more elaborate concept in biology.

In other words, D_{N_i} is the mean squared error (MSE) between N_i and $N_{\omega+1}$ and D_{N_i, N_j} is the MSE between N_i and N_j . Dissent is thus disagreement of a constituent with the aggregate network as opposed to disagreement with other constituents.

Obviously, dissent and disagreement are special cases of extrinsic diversity. They are thus similarly phenomenal and a posteriori. In a sense, they measure differences in behavior. They do not require a supervised training set because they do not use the target values y . They can thus be computed over any subset of the input domain of the network.*

A great degree of dissent implies that the output of the synergetic network differs significantly from the outputs of its homologous subnetworks. In certain cases, it also implies that the synergetic network performs better than its homologous components as the following lemma demonstrates.

Lemma 4.5. *If $N_{\omega+1} = \sum_{i=1}^{\omega} c_i N_i$ and $\sum_{i=1}^{\omega} c_i = 1$ then*

$$E_{N_{\omega+1}} = \sum_{i=1}^{\omega} c_i (E_{N_i} - D_{N_i}),$$

where $E_N(S)$ indicates the MSE of the learner N on the data set S . Thus

$$E_N(S) = \frac{1}{|S|} \sum_{(x,y) \in S} (N(x) - y)^2.$$

Proof. It is

$$\begin{aligned} & \sum_{i=1}^{\omega} c_i \left((N_i(x) - y)^2 - (N_i(x) - N_{\omega+1}(x))^2 \right) \\ &= (N_{\omega+1}(x) - y) \sum_{i=1}^{\omega} c_i (2N_i(x) - y - N_{\omega+1}(x)) \\ &= (N_{\omega+1}(x) - y)^2. \end{aligned}$$

Thus

$$\begin{aligned} E_{N_{\omega+1}}(S) &= \frac{1}{|S|} \sum_{(x,y) \in S} (N_{\omega+1}(x) - y)^2 \\ &= \frac{1}{|S|} \sum_{i=1}^{\omega} c_i \sum_{(x,y) \in S} (N_i(x) - y)^2 - (N_i(x) - N_{\omega+1}(x))^2 \\ &= \sum_{i=1}^{\omega} c_i (E_{N_i} - D_{N_i})(S). \end{aligned}$$

*This property can be used to create hybrid supervised/unsupervised methods.

□

A consequence of the above lemma is that increased dissent can reduce the MSE of an ensemble or a synergetic network even when the MSEs of its homologous subnetworks remain the same. The reduction is not always linear as in the normalized weighted aggregation of the lemma. On the contrary, such reductions are rarely linear in practice. Measures have dependencies. Error rates tend to change when dissent changes. Furthermore, more complex forms of aggregation and synergy can often outperform algebraic combinations such as weighted summations. There is not a simple or clear relationship between dissent and error that applies to all or most cases.

In general, there are no guarantees in learning. Theoretical results that claim the opposite are always based on simplifying assumptions and limiting cases and they often focus on training errors only. Although the above lemma generally falls into that category, it is nevertheless a useful indicator.

Dissent can be seen as a training opportunity. It correlates with increased probability of error and it does not require the true classification of data samples to be known. It can thus help in data gathering and selection, particularly collection of data in regions of increased dissent so that additional evidence can resolve or reduce such dissent. This is particularly useful for ‘unexplored’ regions of the input space where the current training set is maximally sparse or void.

Domains contain an exponential number of data points. This is a direct consequence of combinatorics. As a result, data sets are bound to be sparse almost everywhere. A measure that can point to regions with higher error rates can be useful or critical for improving learning.

Dissent can also enable possible corrections of errors by an ensemble. Indeed, such corrections require one or more of the outputs of the models to be different; and the more different they are the greater the leverage for change and correction. Conversely, if a result is unanimous, there is no dissent and thus no basis for any change or correction on that result.

In synergetic networks, the homologous learners can have identical outputs and a correction can still be possible, if the states of the learners are sufficiently different. This often applies in borderline cases such as classification examples where the winning class has a narrow margin. Such cases can represent a significant percentage of the errors and they may require subtle information from the state that is not present in the output.

Therefore, if we intend to model dissent for synergetic networks in a more accurate way, we should operate on states instead of outputs. If we assume $s_i(x)$ to be the state of network N_i upon input x , we can attempt to use MSE again and redefine dissent as shown below.

$$D_{N_i}(S) = \frac{1}{|S|} \sum_{(x,y) \in S} (s_i(x) - s_{\omega+1}(x))^2.$$

However, such a definition implies that the states s_i and $s_{\omega+1}$ have the same size and dimensionality. It also implies that they are somehow aligned and that they compute similar or identical properties. Those are severe and potentially detrimental constraints. Instead, when dissent is defined upon states, it must use some form of distance between the states that measures their similarity. This introduces a new level of complexity in the learning process. We will examine the issue in more detail in part III of this theory.

Dissent does not necessarily imply structural or architectural diversity. As it appears in definition 4.10 it is a form of extrinsic diversity and it can sometimes be achieved by training data selection. However, increased structural diversity is more likely to result in increased dissent and potentially better overall performance. In other words, dissent is yet another indicator in support of increased diversity.

In this theory, we use extrinsic diversity for data collection purposes and for augmenting the supervised learning paradigm. In every other part of the theory, including learning, heredity, and evolution, we primarily focus on structural and functional diversity. Our emphasis is thus on form and binding (which is allocation of functionality to structure) rather than upon the outcome of functionality. It is an emphasis on cause rather than effect and it is more consistent with biology and evolution.

For the rest of this theory, we use the term *intrinsic diveristy* or simply *diveristy* to refer to structural and functional diversity.

4.12 Definitions of Diversity

In order to define diversity, we must first define a set of characteristics that provide a frame of reference for diversity.

Definition 4.11. Let T be an operator that extracts parameter values from learners. In analogy with biology, we call the parameters *characteristics* and their values *traits*.^{*} We also call T a *trait operator*. If we assume that T_L indicates a tuple containing the corresponding traits for the learner L and D is a metric on the domain of the trait tuples then the *distance* of two learners L and M with respect to T and D is defined as

$$\|L, M\|_T = D(T_L, T_M).$$

For example, the characteristics of a learner could be the total number of units or connections in its neural network, the number of layers, the locality of each layer, the average locality, the sizes of convolution kernels, the activation functions, the learning rate, the learning algorithm, etc.

^{*}The term *configuration parameters* is often used for such characteristics in machine learning and neural network literature.

If we can define trait differences through subtraction, we can then use a norm $\|\cdot\|$ to define learner distance

$$\|L, M\|_T = \|T_L - T_M\|. \quad (4.23)$$

For numeric characteristics such as total number of connections or average locality, trait difference can simply be numeric subtraction. However, if we consider the number of units or connections per layer and two learners have different numbers of layers, the corresponding traits will be tuples with different lengths. In that case, we can use a common convention and pad the smaller tuple with zeros so that both tuples have the same length. Alternatively, we could replace such variable characteristics with some of their statistics such as their length (which, in this case, will be the depth of the network), mean, median, standard deviation, etc.

A similar issue exists for any characteristic that is expressed as a tuple of numbers. For example, we can consider convolution kernels. In that case, we can treat each kernel dimension as an individual characteristic or we can use their product as a single characteristic for the kernel, given that a kernel $(k_1, \dots, k_n)$ represents a window of $\prod_{i=1}^n k_i$ elements.

For non-numeric characteristics, we must define some form of difference between them. The simplest possible form only checks for identical values. Thus if α and β are traits of a characteristic c , their difference can be defined as

$$\alpha - \beta = \begin{cases} 0 & \text{if } \alpha = \beta, \\ 1 & \text{otherwise.} \end{cases}$$

A more elaborate version can assign a rank to each value or set of values of a characteristic. The subtraction can then be defined as the difference of the corresponding ranks

$$\alpha - \beta = r(\alpha) - r(\beta).$$

For example, we can partition activation functions into classes such as sigmoids, partly linear functions, Gaussian or other symmetric functions, multidimensional functions (such as softmin or softmax), etc. and assign a rank to each class. Similarly, for learning algorithms. This choice is applicable when we can partition all the possible values of a characteristic and order the partitions in some intuitive way – for example from simpler to more complex.

Although we can define learner characteristics and trait difference in advance, it is a more appropriate design if we allow the algebra to operate on any sets of characteristics and any form of trait difference and permit the users to provide definitions for those. We thus allow custom characteristics and trait difference in the algebra (the same way we allowed custom synergy, earlier in this chapter).

Note that characteristics can have very different ranges. For example, the

total number of units in a neural network is usually orders of magnitude larger than the number of layers. As a result, similar or identical trait increments can have different significance across different characteristics. This suggests a normalization so that all traits have a common range, such as the interval $[0, 1]$. However, such a normalization can be achieved through an affine transformation, mapping trait value t to $(t - m)/(M - m)$, where m is the minimum possible value of t and M is its maximum. Given that different characteristics can still have different importance and different overall effect even after normalization of their traits, it is likely that a weighted norm will be used, such as weighed p -norm $(\sum_i w_i^p x_i^p)^{1/p}$, where each characteristic receives an individual weight. However, such a norm does not require normalization. A weight w for a characteristic c with affinely normalized traits is equivalent to a weight $\frac{w}{M-m}$ on the original trait, where m and M are the minimum and maximum possible values of the trait.

For the rest of this section, we assume an operator T for given characteristics, predefined trait difference for those characteristics, a specialized norm that accounts for the differences in range and importance among the characteristics, and learner distance as defined in equation (4.23). We also assume a learner to be a pair (N, A) where N is a neural network as specified in definition 4.2 and A is a learning algorithm like gradient descent.

Definition 4.12. The *diversity* of a set of learners $M_1, \dots, M_\omega$ with respect to a trait operator T is defined as

$$\mathcal{D}_T(M_1, \dots, M_\omega) = \frac{2}{\omega(\omega - 1)} \sum_{i < j} \|M_i, M_j\|_T,$$

where

$$\|L, M\|_T = \begin{cases} D(T_L, T_M) & \text{if metric } D \text{ is given,} \\ \|T_L - T_M\| & \text{if trait differences are specified.} \end{cases}$$

Similarly, the diversity for a synergetic learner can be derived from the diversity of its constituent learners.

Definition 4.13. The *inherent diversity* or simply *diversity* of a synergetic learner $M_{\omega+1} = (N_{\omega+1}, A_{\omega+1})$ with respect to a trait operator T is defined as

$$\mathcal{D}_T(M_{\omega+1}) = \mathcal{D}_T(M_1, \dots, M_\omega),$$

where $M_i = (N_i, A_i)$ for $i = 1, \dots, \omega$ are the learners that are contained in $M_{\omega+1}$.

If T does not use any learning characteristics, we can eliminate the learning algorithms A_i and define diversity over networks only:

$$\mathcal{D}_T(N_{\omega+1}) = \mathcal{D}_T(N_1, \dots, N_\omega).$$

Intuitively, diversity corresponds to differences between learners as measured by their average distance. Increasing the pairwise distances (and thus the differences) in a set of learners makes the set more diverse.

An alternative definition of diversity derives from the electrostatic force in physics.* We can consider each learner as a charged body with charge proportional to its error rate. Then we can compute the total repulsive force exerted to each body and assume diversity to be inversely proportional to repulsion (or, more precisely, the average magnitude of repulsion per body).

Definition 4.14. The *dynamic diversity* of a set of learners $M_1, \dots, M_\omega$ with respect to a trait operator T and a test set S is defined as

$$\mathcal{D}_{T,S}(M_1, \dots, M_\omega) = \frac{\omega}{\sum_{j=1}^{\omega} |\vec{F}_j|},$$

where

$$\vec{F}_j = \sum_{i \neq j} \frac{e_i e_j}{\|M_i, M_j\|_T^2} \vec{u}_{ij}$$

represents the total repulsive ‘force’ exerted on learner M_j , $|\vec{v}|$ indicates the length of vector $\vec{v}$, e_j is the error of the learner M_j on the test set S , and $\vec{u}_{ij}$ is the unit vector along the line from T_{M_i} and T_{M_j} (for $i, j \in \{1, \dots, \omega\}, i \neq j$).

Similarly, for synergetic learners:

Definition 4.15. The *dynamic inherent diversity* or simply *dynamic diversity* of a synergetic learner $M_{\omega+1} = (N_{\omega+1}, A_{\omega+1})$ with respect to a trait operator T and a test set S is defined as

$$\mathcal{D}_{T,S}(M_{\omega+1}) = \mathcal{D}_{T,S}(M_1, \dots, M_\omega),$$

where $M_i = (N_i, A_i)$, for $i = 1, \dots, \omega$, are the learners that are contained in $M_{\omega+1}$.

If T does not use any learning characteristics, we can eliminate the learning algorithms A_i and define dynamic diversity for networks only:

$$\mathcal{D}_{T,S}(N_{\omega+1}) = \mathcal{D}_{T,S}(N_1, \dots, N_\omega).$$

Note that the above definitions depend on error rates and thus require a test set. Learners with higher error rates generate stronger repulsive forces. This is a reasonable convention if we assume that there are contiguous regions in the space of characteristics where learners perform poorly. In that case, we would like to associate greater repulsive forces with those regions so that greater diversity is correlated with better performance.

*In physics, the *electrostatic force* is the electric force between charged bodies at rest and its magnitude is defined to be $|\vec{F}| = k_e pq/r^2$, where k_e is Coulomb’s constant, p and q are the charges of the two bodies, and r is their distance.

We can increase repulsion and relative differences between repulsive forces if we substitute each error e_j with some exponential form b^{e_j} in the formula for the repulsive force of definition 4.14:

$$\vec{F}_j = \sum_{i \neq j} \frac{b^{e_i + e_j}}{\|M_i, M_j\|_T^2} \vec{u}_{ij}.$$

The parameter b can then be used to modulate repulsion. The greater the value of b , the greater the overall repulsion and the relative differences between the repulsive forces for each pair of learners.

4.13 Diversity Algorithms

Using the definitions of diversity of the previous section, we can specify *diversity algorithms*, which are algorithms that manipulate (usually increase) diversity in a set of learners. Given that the diversity of a synergetic learner is the diversity of its constituent learners, the above algorithms cover synergetic learners too.

We assume a trait operator T . For dynamic diversity, we also assume a test set S . $\|L, M\|_T$ is assumed as in definition 4.12.

The *diversity selection algorithm* is shown in figure 4.4. It starts with $\omega + n$ randomly chosen learners and then it picks ω of them that maximize diversity.

Diversity Selection(ω, n)

randomly pick learners $M_1, \dots, M_{\omega+n}$
 return $\operatorname{argmax}_B \{\mathcal{D}_T(B) \mid B \subseteq \{M_1, \dots, M_{\omega+n}\} \text{ and } |B| = \omega\}$

Figure 4.4: The diversity selection algorithm.

The *incremental diversity algorithm* starts with a single randomly selected learner and then it keeps adding one learner at a time so that diversity is increased at each step and there are ω learners in the end. The algorithm uses a maximum number of trials n so that it can always terminate. At any given point, s is the number of currently selected learners and d is their diversity. The algorithm is shown in figure 4.5.

The *dynamic diversity algorithm* uses dynamic diversity to select candidate learners but it picks the ones that minimize the error in the end. The algorithm randomly picks ω learners at first. It trains and tests them. Then, at each step, it considers n new randomly selected candidates that move around in trait space for s steps based on repulsive forces. The candidates are subsequently trained and tested. Then, the best ω learners are selected from the entire set. This process is repeated r times. The algorithm is shown in figure 4.6. It requires knowledge of the weight W that is used in the norm or metric

Incremental Diversity(ω, n)

```

randomly pick learner  $M_1$ 
 $s, d = 1, 0$ 
repeat  $n$  times:
    if  $s \geq \omega$ 
        break
    randomly pick learner  $M_{s+1}$ 
     $\delta = \mathcal{D}_T(M_1, \dots, M_{s+1})$ 
    if  $\delta > d$ 
         $d, s = \delta, s + 1$ 
return  $M_1, \dots, M_s$ 

```

Figure 4.5: The incremental diversity algorithm.

Dynamic Diversity($\omega, n, W, s, r, \epsilon$)

```

randomly pick learners  $M_1, \dots, M_\omega$ 
train and test  $M_1, \dots, M_\omega$  to compute errors  $e_1, \dots, e_\omega$ 
repeat  $r$  times:
    randomly select learners  $M_{\omega+1}, \dots, M_{\omega+n}$ 
    for  $j = \omega + 1, \dots, \omega + n$ 
         $\vec{v}_j = \vec{0}$ 
    repeat  $s$  times:
        for  $j = \omega + 1, \dots, \omega + n$ 
             $\vec{F}_j = \sum_{i=1}^{\omega} \frac{e_i \epsilon}{\|M_i, M_j\|_T^2} \vec{u}_{ij}$ 
             $\vec{v}_j = \vec{v}_j + \vec{F}_j \oslash W$ 
             $T_{M_j} = T_{M_j} + \vec{v}_j$ 
        train and test  $M_{\omega+1}, \dots, M_{\omega+n}$  to compute errors  $e_{\omega+1}, \dots, e_{\omega+n}$ 
         $M_1, \dots, M_{\omega+n} = \text{sort}(M_1, \dots, M_{\omega+n})$ 
return  $M_1, \dots, M_\omega$ 

```

Figure 4.6: The dynamic diversity algorithm.

when computing learner distances. v_j is the motion ‘speed’ of learner M_j . ϵ is the default error of the new learners. It is a multiplicative factor of the repulsive forces. It thus affects the speed of movement. (The larger the ϵ , the greater the forces and the faster the movement.) The $\text{sort}()$ function sorts a list of learners in increasing order of their error rate. $\oslash$ indicates element-wise division between two tuples or arrays. Thus, $A \oslash B = [A_i/B_i \mid i \in A_*]$ or $A \oslash B = (A_i/B_i \mid i \in A_*)$, depending on whether A and B are arrays or tuples.

The above algorithm is, of course, a simplification. It treats all traits as unbounded real numbers. In the end, clipping should be used to force traits

back into their range and numerical rounding should be used for traits with discrete domains. Extension of a trait domain should be considered, when the range is often exceeded.

A third type of diversity methods is based on genetic algorithms. Such methods operate on a set of learners for a number of iterations. At each iteration, they create copies of the n best learners and apply random mutations to those copies, where a random mutation is a limited change of a single trait. Then they train and test the mutants and reduce the whole set to its original size by eliminating the worst learners. The learners are also sorted in increasing test error in the beginning so that mutation is preferentially applied to learners of lower error rate. (If $n < \omega$, the n learners with the least error will be selected for mutation; $n > \omega$ some learners will be selected for mutation multiple times and the ones with lower error rates will be selected first.)

Evolutionary Diversity(ω, n, r)

```

randomly pick learners  $M_1, \dots, M_\omega$ 
train and test  $M_1, \dots, M_\omega$  to compute errors  $e_1, \dots, e_\omega$ 
 $M_1, \dots, M_\omega = \text{sort}(M_1, \dots, M_\omega)$ 
repeat  $r$  times:
  for  $i = 1, \dots, n$ 
     $M_{\omega+i} = \text{mutate}(M_i)$ 
  train and test  $M_{\omega+1}, \dots, M_{\omega+n}$  to compute errors  $e_{\omega+1}, \dots, e_{\omega+n}$ 
   $M_1, \dots, M_{\omega+n} = \text{sort}(M_1, \dots, M_{\omega+n})$ 
return  $M_1, \dots, M_\omega$ 

```

Figure 4.7: The evolutionary diversity algorithm.

There are other methods for introducing diversity that are more closely related to biology and evolution and particularly sexual reproduction. The latter suggests combining the trait tuples T_{M_1} and T_{M_2} of two learners M_1 and M_2 with a probability distribution that allows to select either trait or a (linear) combination of them, for each pair of traits from the parents M_1 and M_2 . Furthermore, and unlike biology, we do not have to limit ourselves to two parents only. (Although reproduction from $n > 2$ parents can probabilistically be approximated with multiple reproductions from pairs of parents, the approximation requires additional generations.)

We will discuss and analyze diversity algorithms in more detail later in this theory and especially in the context of heredity. For synergy and synergetic networks, there is an additional complication. The intent to minimize the error of a learner $M_{\omega+1}$ is not always consistent with minimizing the errors of its constituent learners $M_1, \dots, M_\omega$. Trying all possible combinations to reduce $\omega + n$ learners to ω is computationally expensive and thus unlikely from an evolutionary viewpoint.

4.14 Conclusion

The reasoning behind this entire chapter is relatively simple. Cell theory defines life as a synthesis of cells. Similarly, neuron theory and the elementary learning theory define nervous systems and learning systems as a synthesis of neurons or units. Synthesis, however, requires structure and synergy, and synergy requires diversity so that the final outcome would be an effective process that can create not a pile but a whole that is beyond the parts, as Aristotle aptly observed. If the intent is to understand or reproduce such a synthetic process, an algebra is required so that the various steps of the process and their effects can be described in a concise and formal way.

4.15 Historical and Bibliographical Remarks

An estimate of the total number of cells in the human body is given in [26]. A concise description of the anatomy of the human brain and its various parts is available online [322].

Neural multisensory processes are examined in [244]. In particular, the role of visual and auditory cues in multisensory perception of flavor is discussed in [244, chapter 37].

Graph cuts appeared as early as 1955 due to the dual relationship of minimum cuts to maximum flows in networks [69].

The word algebra derives from the Arabic al-jabr (reunion or broken parts) from the title of a book by Persian mathematician M. al-Khwarizmi circa 820 CE [370]. However, the underlying concepts of algebra were developed much earlier. Babylonians developed arithmetic systems and formulas to do calculations while Egyptian, Hellenic, and Chinese mathematicians usually used geometric approaches. Hellenic mathematicians introduced a geometric algebra for geometric objects and their relations prior to the 5th century BCE. The paradigm culminated in the work of Euclid and the publication of his *Elements*, circa 300 BCE [153, 9]. Another Hellenic mathematician, Diophantus, extended the elementary number theory of the *Elements* and introduced an arithmetic algebra in the 3rd century CE [86]. The Indian mathematician Brahmagupta developed a similar algebra in the 7th century CE – albeit with no proofs of propositions but including solutions to linear and quadratic equations, computations with zeros, and the elements of the decimal system as well [32]. Brahmagupta’s texts were translated into Arabic and helped in establishing algebra and spreading the decimal system. Al-Khwarizmi’s main contribution is that he established algebra as a separate branch of mathematics independently of arithmetic and geometry. However, his treatment is fully rhetorical, containing no syncopation or symbols like those that appear in the work of Diophantus and Brahmagupta.

There is a current debate whether Diophantus or al-Khwarizmi is the inventor of algebra. Although they both made important contributions,

we believe that not only Diophantus but also Brahmagupta can lay more legitimate claims on such a title.

The word homology was coined circa 1656 and it is derived from the Hellenic word ὁμολογος "agreeing, consistent", from ὁμός *homos* "same" and λόγος *logos* "ratio, proportion, relation". Thus homologous learners are more consistent with each other because they operate on the same input and they have compatible learning objectives. Conversely, heterologous learners are less consistent because they either operate on different inputs or represent contrasting learning objectives.

Modern game theory started as a field with the publication of a paper by J. von Neumann in 1928 [354] (English translation in [355]). His foundational work culminated with his book on games in 1944 (co-authored with O. Morgenstern) [243]. An overview of the history of the field as well as cooperative game theory appears in the related wikipedia pages [373, 371].

The Shapley value was introduced in 1951 by L. Shapley [311, 380].

Learning ensembles have been used in supervised domains since the late 1970s. Two linear models were aggregated in [342] with a method that fitted data in the first model and residual errors in the second. In [71], the input space was partitioned using two or more classifiers while, in [146], an ensemble of similar neural networks was used.

An aggregation of two heterologous learners was used in [328]. The first was a time-delay neural network for speech recognition and the second was a template matching system for lip reading. The outputs of the two learners were aggregated by a third learner which was a neural network.

The term stacked generalization was first used in [393]. Bagging and boosting were introduced in the 1990s [34, 297, 118].

The Tablet PC group at Microsoft developed a sophisticated learning architecture for handwriting recognition in the 2000s. The system included many heterologous learners and pattern recognition methods including a time-delay neural network, a number of feed-forward neural networks, a dynamic programming subsystem, a decision tree learner, an elastic matching component, a *comparative neural network* (which is a neural network that compares and ranks alternative classifications for a given input), a personalization subsystem, a rule-based language model with a dictionary, and dynamic training based on game theory [139, 90, 2, 51, 145, 272]. At the time, it was arguably the most complex learning architecture in both academia and industry. (It was a relatively large aggregation system but the various learners and components were trained separately. Synergy was thus limited.)

Although the most common usage of ensembles is in supervised learning, there have been applications to unsupervised methods, primarily for clustering [329, 239, 84, 16].

A measure for nucleotide diversity was introduced in 1979 [250]. Estimates of the number of species and the size of taxonomic ranks for each kingdom can be found in [241, 337, 19, 75, 46, 377, 376, 367, 219].

Neutral mutations are discussed in [117]. The neutral theory of evolution was proposed in [186]. In [306, 225], the theory is furthermore discussed and contrasted with linked selection and *soft sweeps*, which correspond to selection of existing mutations in a population.

Marquis de Condorcet published his homonymous theorem in 1785 [74]. Cramer’s theorem was published in 1938 [64].

The idea of equilibrium inefficiency appeared as early as 1983 [95, 96]. The term “price of anarchy” was first used by C. Papadimitriou in 2001 [260] but the underlying concept was introduced in 1999 as *coordination ratio* [194].

Dissent was introduced in [198] as *ambiguity* in the context of ensemble learning along with a version of lemma 4.5 for ensembles. It was cited and renamed *diversity* in [258]. However, we feel that neither term describes the concept accurately.

Various forms and definitions of extrinsic diversity (based on the output of classifiers) appear in the literature. An overview can be found in [201]. A great textbook on ensembles and combining classifiers is [200].

Empirical evidence that (extrinsic) diversity tends to improve ensemble performance can be found in [197, 67, 159, 35, 5, 231, 285]. Evidence that suggests that the actual relationship is more blurred can be found in [201].

There is an extensive literature on applications of genetic algorithms to machine learning and neural networks and more recently to deep learning [129, 237, 399, 294, 167, 332].

In the context of extrinsic diversity, one of the early applications appears in [258]. A similar evolutionary approach appears in [333].

Scientific theories are not discoveries of the laws of nature but rather inventions of the human mind.

— Athanasios Papoulis, *Probability, Random Variables, and Stochastic Processes*, 1965.

5.1	Split Notation	143
5.2	Concepts from Set and Measure Theory	148
5.2.1	Measure Theory	151
5.3	Estimators	152
5.4	Generalization	154
5.5	Is Generalization Impossible?	156
5.6	Regularization	161
5.6.1	Razors and Minimum Descriptions	161
5.6.2	Dropout	162
5.6.3	Network Pruning	164
5.6.4	Metric Regularization and Weight Decay	165
5.6.5	Batch Normalization	167
5.7	Overfitting and Generalization	168
5.7.1	Functional Distinction	170
5.7.2	Non-local and Locality-sensitive Approximation	174
5.7.3	Finite Differences	183
5.8	Coherence	184
5.9	Notes on Coherence	196
5.10	Locality as Coherence	197
5.11	Historical and Bibliographical Remarks	200

5

Coherence

5.1 Split Notation

In algebra, a *congruence* or *congruence relation* is an equivalence relation on an algebraic structure (such as a group, ring, or field) that preserves equivalence over algebraic operations. Thus algebraic operations on equivalent elements will yield equivalent elements.

In number theory, *modular arithmetic* refers to arithmetic operations on the ring of integers that is formed when numbers start from zero again whenever they reach a certain value that is called the *modulus*. Furthermore, *congruence modulo n* is the congruence over the ring with modulus n . The notation for such congruences was introduced by C. F. Gauss. We write $a \equiv b \pmod n$ or $a = b \pmod n$ to indicate that $a = b + kn$ for some $k \in \mathbb{Z}$.

In neural networks, multi-indices are common. For example, the output of a neuron i in layer k is denoted by $Y_{k,i}$ in our definitions of neural networks (chapters 1 and 4). The layer k might be multidimensional and thus the index i might be a multi-index itself. Furthermore, the network changes over time. Thus we often need to add a subscript t and write $Y_{t,k,i}$ to indicate the value of $Y_{k,i}$ at time or iteration t . Layer or activation functions also have an input which usually has one or more arguments and more indexing. Parameters w of a connection between unit i of layer k and unit j of layer l require almost twice as many indices: $w_{t,k,i,l,j}$.

Such long indexing can make formulas cumbersome and hard to read. Using them in learning algorithms makes things worse. Given that learning algorithms and the rest of our theory will become more complex as we move forward, it is important that we introduce a more elegant or at least more economical notation.

The *split notation* separates symbols from their indices, exponents, or arguments in a fashion similar to the modulo congruence. Thus we write

$$a = b : i$$

to indicate $a_i = b_i$. We allow the indices to fill incomplete expressions and

index conditions to appear at the end of the formula. For example, we write

$$f_{+1} = f + f_{-1} \quad : i \geq 1$$

to indicate the *Fibonacci sequence* $f_{i+1} = f_i + f_{i-1}$. Note that we have included the condition $i \geq 1$ on the subscript i at the end of the formula. This is simply a shorthand for “ i for $i \geq 1$ ”. We will place such conditions on the split part when they are simple enough and they do not make the formula harder to read.

Furthermore, because ± 1 are common offsets in indexing, we make the offset optional with a default value of 1. Thus the Fibonacci formula becomes

$$f_+ = f + f_- \quad : i \geq 1.$$

If the extracted index does not appear elsewhere in the formula and its presence and conditions are obvious, we can drop it altogether and write

$$f_+ = f + f_-.$$

(This formula contains only four distinct symbols and seven symbols total; it is arguably the most compact form possible for the Fibonacci sequence.)

In general, the extracted index can be added to any incomplete indexing expression. When multiple extracted indices are used, only incomplete parts are filled with the corresponding indices. Thus

$$f_{3\cdot} = f_{2\cdot} + f_{\cdot k} + f \quad : i, j$$

stands for $f_{i,3j} = f_{2i,j} + f_{i,k_{i,j}} + f_{i,j}$.

If a symbol does not admit an index, the extracted index will not apply to it. Thus, if we have particles with masses m_i for $i = 1, \dots, n$, their energy will be $E_i = m_i c^2$, or, in split notation, $E = mc^2 : i$. The index i simply does not apply to the constant c that represents the speed of light in vacuum. The same applies to *bound indices*, which are indices that are not free variables but are specified by an expression such as the indices of a sum or product. Thus,

$$x + \sum_i f(x_i) \quad : t$$

stands for $x_t + \sum_i f_t(x_i)$. The expression $x_t + \sum_{i_t} f_t(x_{i_t})$ is equivalent anyway. Adding a subscript to the bound index is a redundant operation that only makes formulas harder to read.

We call the above notation *implicit* because the same indexing implicitly applies to all symbols that admit it. However, we can use a vertical bar to indicate indexing exceptions in the implicit notation. Thus we write

$$a = b + c \quad : i | b_j, c_k$$

to indicate $a_i = b_j + c_k$.

Alternatively, in the *explicit split notation*, we must explicitly indicate the indexing of each symbol. In that case, we simply omit the implicit part and add ‘exceptions’ for all symbols. Thus, we can write the above equation as

$$a = b + c \quad : |a_i, b_j, c_k.$$

In both explicit and implicit split notation, exceptions can be used to limit the applicability of indices. Thus

$$f_{,+3} = f_{+2} + f_{,k} + f \quad : i, j | k$$

stands for $f_{i,j+3} = f_{i+2,j} + f_{i,k} + f_{i,j}$.

Exceptions can also be used to handle negative signs. Expressions of the form $-k$ or $+k$ are ambiguous. They can be interpreted as either incomplete index expressions of the form $i - k$ or $i + k$ or simply as a quantity with a sign. We have opted for the former as a convention. A positive sign can always be omitted. A negative sign can be treated as an exception. Thus

$$f = g + h_{-2} \quad : i$$

stands for $f_i = g_i + h_{i-2}$ while

$$f = g + h_{-2} \quad : i | h_{-2}$$

stands for $f_i = g_i + h_{-2}$.

When the same symbol appears multiple times in the exception list with different indexing, the appearances apply in the same order to the corresponding symbols in the formula. Thus, we write

$$a = a + c \quad : |a_i, a_j, c_i$$

to indicate $a_i = a_j + c_i$.

We can extend split notation to apply to superscripts and function arguments. In that case, we distinguish the corresponding elements by positioning them as subscripts, superscripts, or arguments. Thus we write

$$f = g + h \quad :_i(x)^2 | f_i(x)$$

to indicate $f_i(x) = g_i(x)^2 + h_i(x)^2$. (Although, in this case, it is better to write the equation as $f = g^2 + h^2 \quad :_i(x)$.)

We should note that the above substitutions take place in parallel. Thus the index i is not applied to the argument x . We can achieve the latter using the *nested split notation*, where we apply the split notation multiple times, from left to right. Thus

$$f = g + h \quad : i(x)$$

stands for $f_i(x) = g_i(x) + h_i(x)$, while

$$f = g + h \quad : (x) : i$$

stands for $f_i(x_i) = g_i(x_i) + h_i(x_i)$, and

$$f = g + h \quad : i : (x)$$

stands for $f_{i(x)}(x) = g_{i(x)}(x) + h_{i(x)}(x)$.

Finally, we may wish to exclude some parts of an equation from the split notation or use them as atomic entities. For that purpose we use double square brackets or double parentheses. Thus the expression

$$((f * g)) \quad : i(x)$$

stands for $(f * g)_{i(x)}$, not for $f_i(x) * g_i(x)$. The expression

$$[[f * g]] \quad : i(x)$$

is simply $f * g$. In other words, the split notation has no effect on any sub-expression that is contained within double square brackets $[[\cdot]]$. We will refer to the above notation as *split wrapping* or simply *wrapping*.

At first sight, split notation may seem as a somewhat cumbersome or verbose formalism instead of an improvement. There are many simple cases that do not benefit from a split. Some may become longer or harder to read, especially if they include multiple instances of the same symbol with different indexing or arguments. In such cases, a split should be avoided.

For example, the equation

$$\int f(x) d(\mu * \nu)(x) = \iint f(x + y) d\mu(x) d\nu(y)$$

does not really benefit from split notation. Although we can still split the algebraic structure of the equation from its arguments, as shown below,

$$\int f d(\mu * \nu) = \iint f d\mu d\nu \quad : |f(x), (\mu * \nu)(x), f(x + y), \mu(x), \nu(y)$$

we have to use a long explicit split that makes the equation longer and perhaps harder to read in its entirety. We can improve upon the above form using wrapping but the final equation is about as long or concise as the original

$$\int f d((\mu * \nu)) = \iint [[f(x + y)]] d\mu d\nu \quad : (x) | \nu(y).$$

On the other hand, if we consider the equation

$$\int f(x) d(\mu + \nu)(x) = \int f(x) d\mu(x) + \int f(x) d\nu(x)$$

then the split form is obviously an improvement

$$\int f d(\mu + \nu) = \int f d\mu + \int f d\nu \quad : (x).$$

Split notation can produce a greater effect in a number of cases. Those include long derivations where the same indexing or arguments apply throughout the derivation and complex formulas with repeated indexing of arguments. In such cases, the split notation produces a significant improvement in form and brevity. This is important because it is usually long and complex formulas that need help, not simple ones.

As a real example for learning, let us consider equation (2.3) from chapter 2, which we show below.

$$\frac{\partial \ell}{\partial \psi_k} = \frac{\partial \ell}{\partial Y_k} F'_k(Y_{I_k}) \nabla_{\psi} F_k(Y_{I_k}). \quad (5.1)$$

We can recall that we originally omitted time in the equation in order to make the formula more concise and readable. However, with the split notation, we do not have to resort to such simplifications. We can add time back and simply extract it as a common index. Let us see the steps.

If we add time, the above equation becomes

$$\frac{\partial \ell_t}{\partial \psi_{t,k}} = \frac{\partial \ell_t}{\partial Y_{t,k}} F'_{t,k}(Y_{t,I_{t,k}}) \nabla_{\psi} F_{t,k}(Y_{t,I_{t,k}}). \quad (5.2)$$

Of course, if the network's connectivity and thus I_k are constant over time, we do not need a time subscript for I_k . However, this condition is not true under plasticity. Thus, in its most general form, the above equation must use $I_{t,k}$.

In either case, the above formula is overloaded with indices and arguments. However, if we use the split notation and extract common arguments and indices, the equation becomes

$$\frac{\partial \ell}{\partial \psi} = \frac{\partial \ell}{\partial Y} F' \nabla_{\psi} F \quad : (Y_I) : t, k | \ell_t \quad (5.3)$$

or, if we omit time,

$$\frac{\partial \ell}{\partial \psi} = \frac{\partial \ell}{\partial Y} F' \nabla_{\psi} F \quad : (Y_I) : k. \quad (5.4)$$

(If we exclude time, ℓ does not have to appear as an exception because it does not admit any subscripts or arguments.)

If we now compare equations (5.1) and (5.2) with (5.4) and (5.3), we can easily see that the split notation makes the equation more concise and readable. The structure of the underlying equation was obscured by indexing and arguments but it is now more evident.

The split notation is an important mechanism that will become more

effective and elegant as we expand and evolve the theory and its underlying mathematics into more sophisticated and complex forms.

As a more detailed example, we include theorem 2.1 and corollary 2.1 with their equations in split form.

Theorem 5.1 (Back-propagation). *Assume a neural network (G, F, Y) over $(\mathcal{F}, \mathcal{P}, \mathcal{A})$. As in definition 1.2, assume that the network has $n + 1$ layers $L_0, \dots, L_n$ and that the layers do not share learning parameters. Furthermore assume that ℓ is a loss function for the network. Then the partial derivatives of ℓ with respect to the parameters of each layer L_k are given by the following equation:*

$$\frac{\partial \ell}{\partial \psi} = \frac{\partial \ell}{\partial Y} F' \nabla_{\psi} F_k \quad : (Y_I) : k, \quad (5.5)$$

where $\psi_k = \psi F_k$. The partial derivative of ℓ with respect to Y_k can be calculated recursively (in reverse order from $k = n$ to 0):

$$\frac{\partial \ell}{\partial Y} = \nabla_Y \ell + \sum_{\substack{r > k \\ I_r \cap L \neq \emptyset}} \frac{\partial \ell}{\partial Y_r} F'_r \nabla_Y Y_{I_r} \quad : k(Y_{I_r}). \quad (5.6)$$

Corollary 5.1. *Assuming that $F_k = f_k \circ p_k \circ a_k$, $f_k \in \mathcal{F}$, $p_k \in \mathcal{P}$, and $a_k \in \mathcal{A}$, for $k = 1, 2, \dots, n$, the back-propagation theorem is equivalent to*

$$\left. \begin{aligned} \frac{\partial \ell}{\partial \psi f} &= \frac{\partial \ell}{\partial Y} \nabla_{\psi} f(P) \\ \frac{\partial \ell}{\partial \psi p} &= \frac{\partial \ell}{\partial Y} f'(P) \nabla_{\psi} p(A) \\ \frac{\partial \ell}{\partial \psi a} &= \frac{\partial \ell}{\partial Y} f'(P) p'(A) \nabla_{\psi} a(Y_I) \end{aligned} \right\} : k \quad (5.7)$$

and

$$\begin{aligned} \frac{\partial \ell}{\partial Y} &= \nabla_Y \ell + \frac{\partial \ell}{\partial Y_+} f'_+(P_+) p'_+(A_+) \frac{\partial a_+(Y_{I_+})}{\partial Y_{S_+}} \nabla_Y Y_{S_+} \\ &\quad + \sum_{\substack{r > k+1 \\ C_r \cap L \neq \emptyset}} \frac{\partial \ell}{\partial Y_r} f'_r(P_r) p'_r(A_r) \frac{\partial a_r(Y_{I_r})}{\partial Y_{C_r}} \nabla_Y Y_{C_r} \quad : k, \end{aligned} \quad (5.8)$$

where $A_k = a_k(Y_{I_k})$ and $P_k = p_k(A_k)$ for $k = 1, \dots, n$.

5.2 Concepts from Set and Measure Theory

This section introduces some basic concepts from *set theory* and *measure theory* that will be used in this chapter. Readers can safely skip the section if they are familiar with those branches of mathematics.

In set theory, two sets A and B have the same *cardinality* (or *cardinal number* or *cardinal*) iff there is a one-to-one mapping of A onto B . In that

case we write

$$|A| = |B|. \quad (5.9)$$

The above relation is an equivalence relation. In general, we assume that each set A can be assigned its cardinal number $|A|$. (Cardinal numbers are formally defined either through the *axiom of regularity* as the equivalence relation specified by (5.9) or through well-ordered sets and the *axiom of choice*, which implies that every set can be well-ordered.)

Intuitively, the cardinal of a set represents the ‘number’ of elements in the set. Hence, if there is a one-to-one mapping from a set A onto a set B then we can assume that the two sets have the same number of elements. This is obvious for finite sets and seems reasonable for infinite ones.

Finite cardinals are the cardinals of finite sets. They are denoted as the natural numbers $0, 1, 2, \dots$. *Infinite cardinals* are the cardinals of infinite sets and they are usually denoted as $\aleph_0, \aleph_1, \aleph_2, \dots$ or $\omega_0, \omega_1, \omega_2, \dots$. (The latter are often used to represent the order-types of cardinals, which are the corresponding *ordinals*, but we will use both notations, interchangeably.)

$\aleph_0 = |\mathbb{N}|$ is the cardinal of the natural numbers and the first infinite cardinal. $c = |\mathbb{R}|$ is the cardinal of real numbers (or *cardinal of the continuum*). *Ordinal arithmetic* can be easily extended to *cardinal arithmetic*. Thus if $\rho = |R|$ and $\delta = |D|$, we define

$$\begin{aligned} \rho + \delta &= |R \cup D|, \\ \rho \cdot \delta &= |R \times D|, \\ \rho^\delta &= |R^D|, \end{aligned} \quad (5.10)$$

where A^B denotes the set of all functions $f: B \rightarrow A$.

It can be easily shown that $|2^A| = 2^{|A|}$. In other words, the cardinal of the power set 2^A of the set A is $2^{|A|}$.*

The last equation in (5.10) is important for learning because it defines the cardinal of the set of functions from a domain D to a range R . Intuitively, for each element $d \in D$, there are $|R|$ possible values for $f(d)$, when $f: D \rightarrow R$. Therefore, there are $|R|^{|D|}$ total choices (and thus functions from D to R).

As an example, when the functions are n -dimensional Boolean functions, their number is $|R|^{|D|} = 2^{2^n}$, given that $|D| = |\{0, 1\}^n| = 2^n$ and $|R| = |\{0, 1\}| = 2$.

It can easily be shown that $|\mathbb{N}^2| = |\mathbb{N}|$ and that $|\mathbb{R}| = |2^{\mathbb{Q}}| = 2^{|\mathbb{N}|}$.

(For the latter, we can use the concept of a *Dedekind cut*, which maps each real number x to the set $\{q \in \mathbb{Q}/q < x\}$ and thus creates a one-to-one function from $\mathbb{R}$ to $2^{\mathbb{Q}}$. And then we can create a one-to one function $f: \{0, 2\}^{\mathbb{N}} \rightarrow \mathbb{R}$ that maps each infinite sequence $\langle a_k/k \in \mathbb{N} \rangle \in \{0, 2\}^{\mathbb{N}}$ to the real number $\sum_{k \in \mathbb{N}} 3^{-k} a_k$.)

*Hint: consider the function $f: 2^A \rightarrow \{0, 1\}^A$ that maps each $X \subseteq A$ to the function $g_X: A \rightarrow \{0, 1\}$ such that $g_X(x) = (x \in X)$.

Cantor's theorem states that $|A| < |2^A|$, for every set A .*

Consequently, for every $n \in \mathbb{N}$, it is

$$n < |\mathbb{N}| < 2^{|\mathbb{N}|} = |\mathbb{R}|. \quad (5.11)$$

Or, equivalently, in cardinal form:

$$n < \aleph_0 < 2^{\aleph_0} = c.$$

Intuitively, finite sets have finite cardinality. Those are effectively the natural numbers. Infinite sets have infinite cardinals and the smallest of them is the cardinal $\aleph_0$ (or ordinal ω_0), which represents the multitude of natural numbers $\mathbb{N}$. Sets with cardinality $\aleph_0$ are called *countable*. Sets with cardinality greater than $\aleph_0$ are called *uncountable*. The set of real numbers is much larger than $\mathbb{N}$ and precisely as large as $2^{\mathbb{N}}$.

An important property of cardinal numbers is the following:

Theorem 5.2.

$$(\forall a) \quad \aleph_a \cdot \aleph_a = \aleph_a \text{ and} \quad (5.12)$$

$$(\forall a, b) \quad \aleph_a \cdot \aleph_b = \aleph_b, \text{ if } a < b. \quad (5.13)$$

The proof of the above theorem can be found in set theory textbooks. Equation (5.13) is a direct consequence (5.12) since

$$\aleph_b \leq \aleph_a \cdot \aleph_b \leq \aleph_b \cdot \aleph_b = \aleph_b.$$

The above theorem can be used to establish the following

Theorem 5.3. *For all $n \in \mathbb{N}$, it is*

$$\begin{aligned} |\mathbb{N}^n| &= |\mathbb{N}|, \\ |n^{\mathbb{N}}| &= 2^{\aleph_0} = |\mathbb{R}|, \\ |\mathbb{N}^{\mathbb{N}}| &= 2^{\aleph_0} = |\mathbb{R}|, \\ |\mathbb{R}^{\mathbb{N}}| &= 2^{\aleph_0 \cdot \aleph_0} = 2^{\aleph_0} = |\mathbb{R}|, \\ |\mathbb{N}^{\mathbb{R}}| &= \aleph_0^c = |2^{\mathbb{R}}|, \text{ and} \\ |\mathbb{R}^{\mathbb{R}}| &= 2^{\aleph_0 \cdot c} = |2^{\mathbb{R}}|. \end{aligned} \quad (5.14)$$

The cardinal $2^c = |2^{\mathbb{R}}|$ represents an important upper bound for probabilities as we shall see below.

*The proof utilizes an important mathematical technique that is called *diagonalization*. For any function $f: A \rightarrow 2^A$, the set $B = \{a \in A / a \notin f(a)\}$ cannot be in the image of f in 2^A . (If $B = f(x)$ for some $x \in A$ then $x \in f(x) = B$ iff $x \notin f(x)$.) Thus f cannot be onto 2^A and, equivalently, $|A| \neq |2^A|$. Finally, the function $g(a) = \{a\}$ for $a \in A$ implies that $|A| \leq |2^A|$.

The equations (5.14) can be interpreted as follows. The number of functions from finite integers to integers is countable. The number of functions from integers to integers is uncountable (equal to c). The number of functions from integers to reals is also equal to c . However, the number of functions from reals to integers and the number of functions from reals to reals are both equal to the upper bound 2^c .

It should be noted that continuous real functions can be uniquely defined by their values at rational numbers. Indeed, if $\langle q_k \in \mathbb{Q}/k \in \mathbb{N} \rangle$ is a rational sequence that converges to a real number x , the value of a continuous function f at x is uniquely defined by the values of f at $\mathbb{Q}$:

$$f(x) = f(\lim_{k \rightarrow \infty} q_k) = \lim_{k \rightarrow \infty} f(q_k).$$

Consequently, the number of continuous real functions is

$$|\mathbb{R}^{\mathbb{Q}}| = 2^{\aleph_0 \cdot \aleph_0 \cdot \aleph_0} = 2^{\aleph_0} = c. \quad (5.15)$$

5.2.1 Measure Theory

Measure theory is the branch of mathematics that studies measures. The latter are defined as follows.

Definition 5.1. A pair $(\Omega, \mathcal{S})$ is a σ -algebra or σ -field on the set Ω iff

- a. $\Omega \in \mathcal{S} \subseteq 2^\Omega$
- b. $(\forall A \in 2^\Omega) \quad A \in \mathcal{S} \Rightarrow \Omega - A \in \mathcal{S}$
- c. If $(\forall i \in \mathbb{N}) A_i \in \mathcal{S}$ then $\cup_i A_i \in \mathcal{S}$

Definition 5.2. Given a σ -algebra $(\Omega, \mathcal{S})$, a *measure* in $\mathcal{S}$ is a function $\mu: \mathcal{S} \rightarrow \mathbb{R}_+ \cup \{0, +\infty\}$ such that

- a. $\mu(\emptyset) = 0$
- b. If $(\forall i, j \in \mathbb{N}) i \neq j \Rightarrow A_i \cap A_j = \emptyset$ then $\mu(\cup_i A_i) = \sum_i \mu(A_i)$

$(\Omega, \mathcal{S}, \mu)$ is then a *measure space*, $(\Omega, \mathcal{S})$ is a *measurable space*, and the sets in $\mathcal{S}$ are *measurable sets*.

Property (b) in definition 5.2 is often called *countable additivity*. It is required because measures are used to measure sets, as their name suggests. Typical measurements are lengths, areas, or volumes. And when two or more sets that represent shapes are disjoint, their total area or volume is the sum of their areas or volumes.

A *probability space* is a measure space $(\Omega, \mathcal{S}, \mu)$ such that $\mu(\Omega) = 1$. In that case Ω is the set of all possible outcomes in the probability space while $\mathcal{S}$ is the set of all events of interest that are assigned probabilities.

In general, probabilities can be defined over finite or infinite sets and their σ -algebras as long as the sets in the σ -algebra are measurable. However,

we cannot define probabilities over the entire power set of reals because it contains non-measurable sets. (Their cardinality is 2^c and they are densely distributed.)

Intuitively, the cardinal 2^c is too big for probabilities. If we try to define probabilities over such a large set, we will end up restricting our definition to a much smaller subset. (The corresponding argument for $2^{\aleph_1}$ cannot be proved in ZFC, if ZFC is consistent.*)

This is an important limitation. In theoretical machine learning, an implicit probability distribution for a target function that generated a data set is often assumed without concern about the cardinality of the set of all possible such functions. Such an assumption should be used with caution because it can potentially lead to a mathematical contradiction.

On the other hand, if we approximate real quantities with rational numbers or with some finite precision $\epsilon > 0$, our approximate functions will be in $\mathbb{N}^{\mathbb{N}}$, which has cardinality c . The same applies if we limit ourselves to continuous functions or continuous functions with a countable number of discontinuities.

5.3 Estimators

In statistics, an *estimator* is a method for calculating an estimate of an unknown quantity from observed data. The unknown quantity is the *estimand* and the result of the estimator is the *estimate*. Assuming a random variable X , data x for an estimand θ of X , and an estimator that produces an estimate $\hat{\theta}(x)$ from x , the *estimator error* on the data x is defined as

$$e(x) = \hat{\theta}(x) - \theta \quad (5.16)$$

and the *bias* of the estimator is defined as

$$B(\hat{\theta}) = E\hat{\theta}(X) - \theta, \quad (5.17)$$

which is the expected value of the error.

An estimator is called *unbiased* or *biased* when its bias is zero or non-zero, respectively. For example the *sample mean* $\hat{\mu}(x) = \sum_{i < n} x_i/n$ is an unbiased estimator of the mean EX .

A common estimation problem is to compute the expected value $E(X/Y)$ of two random variables X and Y . Given data $x = [x_i/i \in \mathbb{N}_n]$ and $y = [y_i/i \in \mathbb{N}_n]$, the *mean-of-ratios estimator* is

$$R_1(x) = \frac{1}{n} \sum_{i < n} \frac{x_i}{y_i} \quad (5.18)$$

*ZFC is the Zermelo-Fraenkel axiomatic set theory with the *axiom of choice* included.

and the *ratio-of-means estimator* is

$$R_2(x) = \frac{\sum_{i < n} x_i}{\sum_{i < n} y_i}. \quad (5.19)$$

Both R_1 and R_2 are biased estimators. For example, given that the function $f(x) = 1/x$ is convex for $x > 0$, Jensen's inequality implies that

$$\frac{EX}{EY} \leq EXE\frac{1}{Y} = E(X\frac{1}{Y}) = E\frac{X}{Y} \quad (5.20)$$

if we assume X and Y to be independent and $X, Y > 0$. However, the bias of R_1 is about n times larger than the bias of R_2 as the following theorem shows.

Theorem 5.4. *When R_1 and R_2 are applied to data samples of size n only, for some $n \in \mathbb{N}_+$, it is $B(R_1) \approx nB(R_2)$.*

Proof. Let $f: \mathbb{R}^2 \rightarrow \mathbb{R}$ be defined as $f(x, y) = x/y$. Then if we approximate f by its second degree Taylor polynomial at a point $(a, b) \in \mathbb{R}^2$, we obtain

$$\begin{aligned} \frac{x}{y} - \frac{a}{b} &= f(x, y) - f(a, b) \\ &\approx (x - a)\nabla_x f + (y - b)\nabla_y f + (x - a)(y - b)\nabla_{xy} f \\ &\quad + \frac{1}{2}(x - a)^2\nabla_{xx} f + \frac{1}{2}(y - b)^2\nabla_{yy} f \quad : (a, b) \\ &= \frac{x - a}{b} - \frac{a(y - b)}{b^2} - \frac{(x - a)(y - b)}{b^2} + \frac{a(y - b)^2}{b^3} \\ &= \frac{2(x - a)}{b} - \frac{xy - ab}{b^2} + \frac{a(y - b)^2}{b^3}. \end{aligned} \quad (5.21)$$

We will repeatedly use the above approximation in this proof.

We now assume two random variables X and Y . Given that data contain only n samples from X and Y , we define X_i and Y_i for $i \in \mathbb{N}_n$ to be the random variables representing sample i from X and Y , respectively,

We furthermore define $\mu_X = EX = EX_i$, $\mu_Y = EY = EY_i$, for all $i < n$. We also define $S = \frac{1}{n} \sum_{i < n} X_i$, and $T = \frac{1}{n} \sum_{i < n} Y_i$. Then

$$\begin{aligned} B(R_1) &= E \left(\frac{1}{n} \sum_{i < N} \frac{X_i}{Y_i} \right) - \frac{\mu_X}{\mu_Y} \\ &= \frac{1}{n} \sum_{i < N} E \left(\frac{X_i}{Y_i} - \frac{\mu_X}{\mu_Y} \right) \\ &\approx \frac{1}{n} \sum_{i < N} -\frac{\text{Cov}(X_i, Y_i)}{\mu_Y^2} + \frac{\mu_X \text{Var}(Y_i)}{\mu_Y^3} \\ &= \frac{\mu_X \text{Var}(Y) - \mu_Y \text{Cov}(X, Y)}{\mu_Y^3}, \end{aligned} \quad (5.22)$$

where the approximation derives from a repeated applications of equation (5.21) with $a = \mu_X$, $b = \mu_Y$, $x = X_i$, and $y = Y_i$, for $i < n$.*

Similarly,

$$\begin{aligned}
 B(R_2) &= E\left(\frac{S}{T}\right) - \frac{\mu_X}{\mu_Y} \\
 &= E\left(\frac{S}{T} - \frac{\mu_X}{\mu_Y}\right) \\
 &\approx -\frac{\text{Cov}(S, T)}{\mu_Y^2} + \frac{\mu_X \text{Var}(T)}{\mu_Y^3} \\
 &= \frac{\mu_X \text{Var}(Y) - \mu_Y \text{Cov}(X, Y)}{n\mu_Y^3}, \tag{5.23}
 \end{aligned}$$

where the approximation once more derives from equation (5.21) (with $a = \mu_X$, $b = \mu_Y$, $x = S$, and $y = T$) and the last step from the facts that $\text{Cov}(S, T) = \frac{1}{n} \text{Cov}(X, Y)$ and $\text{Var}(T) = \frac{1}{n} \text{Var}(Y)$.

Combining equations (5.22) and (5.23) completes the proof. $\square$

An immediate consequence of the above theorem is that we can construct an almost unbiased estimator from an obvious linear combination of the ratio-of-means and the mean-of-ratios estimators.

Corollary 5.2. *Let $R_3 = \frac{nR_2 - R_1}{n-1}$. Then $B(R_3) \approx 0$.*

Proof. $B(R_3) = B\left(\frac{nR_2 - R_1}{n-1}\right) = \frac{nB(R_2) - B(R_1)}{n-1} \approx 0$. $\square$

In general, when R is a ratio estimator applied to data x and y , we would write $R(\frac{x}{y})$ instead of $R(x, y)$ to make the notation more visibly associated with the functional intention.

Various unbiased ratio estimators are present in statistics literature. They either use common sampling schemes or modify such sampling schemes to generate unbiased estimators. However, all those methods invariably result in more complex and sophisticated estimators. In this theory, we mainly use R_2 or R_1 because their simplicity makes certain proofs easier. However, as we argued in section 2.6, provability should not take precedence over complexity. Consequently, we think that amended definitions with unbiased estimators may be part of the evolution of this theory; but we do not feel that they are essential at this stage because R_2 is often sufficiently accurate.

5.4 Generalization

In machine learning literature, *generalization* is defined as the ability of a learner to perform well on data samples that do not appear in its training

* $\text{Cov}(X, Y) = E(XY) - EXEY$ indicates the (covariance) of two random variables X and Y , while $\text{Var}(Y) = \text{Cov}(Y, Y) = E(Y^2) - (EY)^2$ indicates the (variance) of Y .

set. It is usually measured by using a test or validation set that is disjoint from the training set. (An alternative and more computationally intensive method is *cross validation* where only a training set is required, training is repeated a number of times, each time excluding a small portion of the training set, usually around 10%, training on the rest and testing/validating on the excluded parts. The average of the validation errors is then used as an estimate of the overall validation error.)

The above definition of generalization is primarily phenomenal. It does not explain how and why generalization is possible; or what are the processes and factors that enable or affect it. It is similar to describing rain as water that falls from the clouds. The latter is an accurate description. We can measure rain in that regard as the amount of precipitation per minute. However, such a description is not an explanation or a scientific definition of rain. It does not provide an origin for the water, the process that moved the water to the clouds, or the process that makes it fall from the clouds. Without the concepts of evaporation and condensation, rain is primarily a mystery. We can measure it, but we cannot understand it.

We will tentatively adopt the above phenomenal definition for generalization but we will provide a more causal one in this chapter. We will also assume that generalization is a primary and prerequisite function of learning. Without it, there can be no learning.

We should note that, in psychology and neuroscience, generalization is defined as the use of learning from prior experiences in situations that are regarded as similar. The learner uses patterns, logic, principles, or other unidentified similarities to connect prior experiences and knowledge with the current situation and make inferences that may affect behavior and modify knowledge.

This definition is certainly less phenomenal but it is not mathematical or rigorous. It is rather qualitative and behavioral. It does not explain the kind of learning that is used or how prior experiences are encoded. It does not provide a definition for the underlying patterns or principles. It does not define how similarity is measured. It does not describe the mechanisms that enable similarity detection and the associated inferences and behavior. And it focuses largely on similarity when contrast, and everything between contrast and similarity, constitute legitimate alternatives.

Let us now return to our tentative phenomenal definition. Using it, we can define generalization error as follows:

Definition 5.3. Let D and R be two sets. The *generalization error* of a function $h: D \rightarrow R$ with respect to a loss function ℓ , a target function $f: D \rightarrow R$, and a probability distribution F over D is defined to be

$$\mathcal{E}_\ell(h, f) = \int \ell(h, f) dF.$$

The function h is often called the *hypothesis* and R^D is called the *hypothesis*

esis space. If not all functions in R^D are possible or valid hypotheses, the hypothesis space is a subset of R^D .

We should note that we did not exclude the training samples from the above definition because the domain D is often infinite or combinatorially larger than any actual training set. For the same reason, we cannot use the above definition to compute the generalization error.

Instead we use a finite and often small set that is disjoint from the training set to obtain an estimate of the generalization error.

Definition 5.4. Assume two sets D and R , a hypothesis $h: D \rightarrow R$, a loss function ℓ defined over R^2 , a probability distribution F over D , and a finite test set $T \subset D \times R$ that is disjoint from the training set of h , distributed according to F , and derived from an unknown target function $f \in R^D$. Then the *test error* (or *empirical error*) of h on T with respect to ℓ is defined to be

$$E_\ell(T/h) = \frac{1}{|T|} \sum_{(d,r) \in T} \ell(h(d), r),$$

where $|T|$ is the cardinal (number of elements) of T .

Given that the hypothesis h is generated by a learning algorithm, we can similarly define the test error for the learning algorithm.

Definition 5.5. Assume two sets D and R , a learning algorithm $L: 2^{D \times R} \rightarrow R^D$, a loss function ℓ defined over R^2 , a probability distribution F over D (for the data), a probability distribution G over R^D (for the hypotheses), and a finite test set $T \subset D \times R$ that is disjoint from the training set S used by L , distributed according to F , and derived from an unknown target function $f \in R^D$. Then *test error* (or *empirical error*) of L on T with respect to ℓ and given the training on S is defined to be

$$E_\ell(T/L, S) = \int \frac{1}{|T|} \sum_{(d,r) \in T} \ell(L_S(d), r) dG.$$

In other words, the test error provides an approximation of the generalization error that is generated by a smaller test set T instead of using the entire domain D .

The above definitions raise an obvious question. Is generalization possible? If the value $f(d)$ of the unknown target function f can be arbitrary at any given point d , what hope is there for generalization? We will attempt to answer this question in the following section.

5.5 Is Generalization Impossible?

In this section we will assume a domain D and a range R for the functions. In order to simplify the discussion and the proofs and avoid logical pitfalls

or loopholes that emerge from large cardinality (such as 2^c or $2^{\aleph_1}$), we will assume that both sets D and R are finite and discrete, or, equivalently, that the cardinals $|R|$ and $|D|$ are finite.

Let us now be more formal and define data sets as relations.

In set theory, a *binary relation* over sets D and R is a subset of the Cartesian product $D \times R$. A *data set* S over D and R is thus a binary relation over D and R .*

Definition 5.6. If S is a data set (binary relation) over two sets D and R , then $\text{dom } S$ and $\text{ran } S$ indicate the domain and range of S , which are the projections of S onto D and R , respectively:

$$\begin{aligned}\text{dom } S &= \{d \in D / (\exists r \in R) (d, r) \in S\}; \\ \text{ran } S &= \{r \in R / (\exists d \in D) (d, r) \in S\}.\end{aligned}$$

When S represents a function (that is $r = t$ whenever $(d, r) \in S$ and $(d, t) \in S$), the data set is called *functional*; otherwise it is called *relational*. Given that a function is a special case of a binary relation, we will also use the symbol of a function to indicate the underlying relation. Thus, the following expression is a tautology iff f is a function

$$f = \{(d, f(d)) / d \in \text{dom } f\}. \quad (5.24)$$

In other words, a functional data set S is a function over its domain while a relational one is a relation (but not a function). If S is a functional data set then, for every $d \in \text{dom } S$, $S(d)$ denotes the unique $r \in R$ that satisfies the condition $(d, r) \in S$.

Relational data sets may indicate that the intention is to model a relation, as is the case with multi-label classification. However, relational data sets can arise because of noise in the data, human errors in the generation of the data set, or latent dimensions that are not present in the domain D and make functions on the expanded domain appear as relations over D .

We will return to this topic in volume III. For the rest of this chapter, we will primarily consider functional data sets.

Definition 5.7. Given a data set S over two sets D and R , a function $f \in R^D$ is *consistent* with the data set S iff $S \cap f = S$ (or, equivalently, $r = f(d)$ for all $(d, r) \in S$). The set of such functions is indicated as R_S^D . Hence,

$$\begin{aligned}R_S^D &= \{f \in R^D / S \cap f = S\} \\ &= \{f \in R^D / (\forall (d, r) \in S) r = f(d)\}.\end{aligned}$$

We can now prove the following theorem about the cardinality of consistent functions.

*This refers to supervised data sets. An unsupervised data set is simply a subset of D .

Theorem 5.5. *If S is a functional data set over sets D and R then*

$$|R_S^D| = |R|^{|D - \text{dom } S|}$$

Proof. Given that S is a functional data set, we can define a function B mapping functions in $R^{D - \text{dom } S}$ to functions in R_S^D such that $B(g) = S \cup g$. B is obviously a bijection. (If $g_1, g_2 \in R^{D - \text{dom } S}$ then $S \cup g_1 = S \cup g_2$ iff $g_1 = g_2$; and if $f \in R_S^D$ then $S \subseteq f$, $f - S \in R^{D - \text{dom } S}$, and $B(f - S) = f$.) As a result, $|\text{dom } B| = |\text{ran } B|$, which implies that $|R|^{|D - \text{dom } S|} = |R_S^D|$. $\square$

In the previous section, we assumed a learning algorithm L but we did not provide a definition. Formally, we can define learning algorithms as follows.

Definition 5.8. A *learner of learning algorithm* L over two sets D and R is a method that operates on a data set S over D and R and produces a function $h \in R^D$ (or a relation $h \in 2^{D \times R}$, if S is relational) as a candidate for the function or relation that was used to generate S .

$L(S)$ indicates the random variable that represents the output of L upon input S . As in the previous section, $E_\ell(T/L, S)$ indicates the expected test error of $L(S)$ on the test set T with respect to the loss function ℓ . T is usually assumed to be disjoint from the training set S ($\text{dom } T \cap \text{dom } S = \emptyset$).

The test error is often used as an estimate of the generalization error since the latter cannot be computed directly in most situations.

A common loss function simply counts the number of correct predictions by the generated function $L(S)$. In that case, the loss function is

$$\ell(z, y) = (z \neq y) = \begin{cases} 1 & \text{if } z \neq y \\ 0 & \text{otherwise} \end{cases}. \quad (5.25)$$

When the above loss is used, we will simply drop the subscript and write $E(T/L, S)$ instead of $E_\ell(T/L, S)$.

Now we can prove the following theorem.

Theorem 5.6. *Let L and S be a learning algorithm and a data set over two sets D and R of finite cardinality. If the underlying function f that is used to generate such a data set is uniformly distributed in R^D then, for all $d \in D - \text{dom } S$, it is*

$$E(T/L, S) = 1 - \frac{1}{|R|}, \quad (5.26)$$

where $T = \{(d, f(d))\}$.

Proof. Let $P(f/S)$ be the probability that the function $f \in R^D$ generated the data set S , given S ; and $P(h/L, S)$ be the probability that $L(S) = h$, given L and S . We assume that $d \in D - \text{dom } S$ and define $U = S \cup \{(d, h(d))\}$.

Then it is

$$E(T/L, S) = \sum_{f, h \in R^D} (f(d) \neq h(d)) P(h/L, S) P(f/S). \quad (5.27)$$

However, $P(f/S) = 0$ when $f \notin R_S^D$. Thus

$$\sum_{f \in R^D} (f(d) \neq h(d)) P(f/S) = \sum_{f \in R_S^D} (f(d) \neq h(d)) P(f/S). \quad (5.28)$$

The probability that a randomly chosen function $f \in R_S^D$ is not consistent with U is

$$\sum_{f \in R_S^D} (f(d) \neq h(d)) P(f/S) = \frac{|R_S^D - R_U^D|}{|R_S^D|} = \frac{|R_S^D| - |R_U^D|}{|R_S^D|} \quad (5.29)$$

since f is uniformly distributed in R^D and $R_U^D \subset R_S^D$. Applying theorem 5.5 to the above equation, we get

$$\sum_{f \in R_S^D} (f(d) \neq h(d)) P(f/S) = 1 - \frac{1}{|R|} \quad (5.30)$$

since $|D - \text{dom } U| = |D - \text{dom } S| - 1$. Combining equations (5.27), (5.28), and (5.30), we obtain

$$E(T/L, S) = \sum_{h \in R^D} P(h/L, S) \left(1 - \frac{1}{|R|}\right) = 1 - \frac{1}{|R|}$$

because $P(h/L, S)$ is a probability distribution over R^D and as such it must be $\sum_{h \in R^D} P(h/L, S) = 1$. $\square$

Now, let us compare the above error with the error of a function τ that picks a random value $r \in R$ for any given $d \in D$. The probability that τ would be correct (i.e. $\tau(d) = f(d)$) on a data point $d \in D$ is $1/|R|$, since there are $|R|$ possible choices and only one is correct. As a result, the probability of error for τ is $1 - 1/|R|$.

In other words, all learning methods L have the same expected generalization error as random guessing. This is a striking outcome. It implies that learning (and thus life and intelligence) are impossible. Given that planet earth contradicts the latter conclusion, there must be an incorrect assumption in theorem 5.6.*

Of course, we proved theorem 5.6 for sets R and D with finite cardinality. However, this is not an important limitation. For we can always use a finite

*More elaborate versions of theorem 5.6 are known as the *no free lunch theorems*. Those are discussed in section 5.11.

precision $\epsilon > 0$, quantize numbers, and operate within a bound, since the total energy of planet earth is finite. Then our domain and range will be finite, and picking ϵ to be sufficiently small, we can bring the generalization error arbitrarily close to $1 - 1/|R|$.*

The culprit is not our assumption of finite cardinality but the assumption of a uniform distribution of the function f over R^D . This is clearly a fallacy. Any universe that is governed by physical laws is bound to operate on a dramatically non-uniform distribution. In any universe and any environment that satisfies conservation laws, such as the conservation of energy, smaller constants in functions and functions that increase at a measured pace will be significantly more likely than astronomical constants and triply or quadruply exponential functions. Different environments, such as different ecosystems on earth, have diverging distributions that favor certain species over others – or species adapt and evolve according to those distinct distributions. Finally, and despite its immensity, the known universe is dwarfed by $|R^D|$ when $|R|$ and $|D|$ are of the order of 1000, which is a relatively small size for domains and ranges. Only a tiny fraction of those functions will ever appear in the lifetime of our planet and our species.

Despite its erroneous assumption, theorem 5.6 is actually important and for two reasons.

First, it demonstrates that there is a significant bias on planet earth (and perhaps in the universe as well) that makes learning and intelligence possible. This is an important philosophical realization and it may invite research for a definition or description of the bias and its distribution across various environments.

Second, and more importantly for learning, the theorem implies that a learning method that does not take the bias of the environment into account, is bound to operate as though there were no bias and thus produce greater error rates as well as compromised learning and generalization. In this case, environment indicates context. It does not refer only to broad environments such as ecosystems on earth or application domains but also to individual problems, learners, applications, localities inside a learner, and even individual neurons that have a large number of connections organized in a dendritic fashion.

For example, let us consider gradient descent as a learning algorithm. We assume a parameter that has a negative gradient upon some training iteration. If we increase the value of that parameter proportionally to its gradient without taking into account the value of the parameter, its prior history of values, updates and gradients, and the values and the histories of other parameters in the same locality, we effectively ignore the bias of the locality and place all of our trust to a single derivative value. In the end, such

*We can also extend theorem 5.6 for domains D that are infinite but countable (i.e. $|D| = \aleph_0$) but this does not affect the overall argument. Quantization and finite quantities constitute an essential condition that can apply to all real situations and applications.

narrow and myopic methods can only have a detrimental effect on learning and generalization.

We will refer to learning systems that do not use environmental or contextual information as *narrow* – for they can perform well only under a narrow (and often unrealistic) set of conditions.

5.6 Regularization

The mathematician J. Hadamard believed that mathematical models of physical systems or phenomena should admit a unique solution for the underlying problem and that the solution should change continuously with the initial conditions of the system. In modern mathematics and physics, such problems are called *well-posed problems*; all other problems are called *ill-posed*.

It should be noted that there is nothing wrong with ill-posed problems. The term is actually misleading. Such problems are either *under-constrained* (meaning that there are not enough constraints to lead to a unique solution) or discontinuous. The former is quite common in machine learning and optimization because there often exist multiple optimal or near-optimal solutions for a system, given a data set and a loss function.

The generic term *regularization* is used to describe the process of adding information to such a system so that a solution can be selected either deterministically or probabilistically. (Regularization is also used to prevent or reduce *overfitting*, which is discussed in the next section.)

Of course, if the distribution of the underlying functions were uniform, regularization would be meaningless. Conversely, when we apply any form of regularization, we implicitly assume a non-uniform distribution.

The use of regularization in machine learning has primarily been driven by empiricism. This is not the best approach to the problem but it is a starting point. It also reflects an imperative condition because current non-regularized learning systems are narrow and thus substantially suboptimal. A number of techniques have thus been developed. The most common method is to add a term to the loss function that affects the derivatives of the learned parameters. As a minor exposition, we provide a few regularization methods below.

5.6.1 Razors and Minimum Descriptions

Ockham's razor states that “entities should not be multiplied without necessity”. It is often attributed to William of Ockham but the earliest reference to the principle as well as a related discussion appeared in Aristotle’s *Physics* and *Posterior Analytics* more than fifteen centuries earlier. In a sense, Aristotle invented the principle but William of Ockham popularized it. We will thus refer to it as the *Aristotle-Ockham razor*.

The principle is effectively a bias for minimalism and simplicity. If two contending theories A and B explain the same phenomena, the simpler one

should be preferred because it is more likely to be true. This probabilistic argument is obvious if the assumptions of A are a proper subset of the assumptions of the more complex theory B . By definition, every assumption introduces a possibility of error. As a result, the more complex theory B is more likely to be erroneous.

The same applies to trained models in machine learning. When two models achieve the same training error, the smaller model is more likely to generalize better. As a result, the Aristotle-Ockham razor is a selection mechanism and thus a form of regularization that contradicts the assumption of a uniform distribution of functions. The fundamental nature of the principle and the fact that it was first discovered by Aristotle more than 23 centuries ago only demonstrates the extent of the fallacy when a uniform distribution is assumed and all learning is declared to be equal.

The actual situation is more complex than the Aristotle-Ockham razor because models of different sizes and complexities usually achieve different training errors. Thus, the Aristotle-Ockham razor cannot be applied directly. One must provide a method to compare increments in size and complexity with reductions of error.

In either case, Aristotle-Ockham razor is not a deductive argument or mechanism. It works in many situations in machine learning because learning algorithms cannot guarantee optimality. Consequently, larger and more complex models are used to account for the necessary deficiency of the learning methods. Hence, when two models achieve identical or similar results, the simpler or smaller model is more likely to generalize better.

Minimum description length is a principle of parsimony applied to formal languages that describe data. It is a formalization of the Aristotle-Ockham razor on such languages and it states that the shortest description of the data is their best model.

The concept is directly related to the *Kolmogorov complexity* of an entity, which is the length of the shortest program (in a predefined language) that produces the entity as output. Both the Aristotle-Ockham razor and the minimum description length principle represent a bias for lower Kolmogorov complexity.

The above are general principles that apply to a range of problems. For example, the Aristotle-Ockham razor has been applied in various sciences and a diverse set of problems. However, those are beyond the scope of this theory.

5.6.2 Dropout

In physics, *mean-field theory* represents a collection of methods that reduce high-dimensional stochastic systems to simpler approximate models that replace a set of parameters in the system with their mean.

Dropout is an application of mean-field theory to learning where the mean is taken over a subset of the parameters of a model during training. It is

a regularization method that attempts to reduce variance and it is usually applied to layers of neural networks.

More specifically, dropout uses a probability p that any unit in a layer will be ‘dropped’, meaning that the unit and its connections will not be used in the computation of the current training iteration. This is equivalent to training a smaller network that does not include the dropped unit. Given that units are dropped probabilistically, dropout effectively trains a mixture of models instead of a single one.

In its original publication, dropout was also described as a method that prevents units from “co-adapting”. The argument is more intuitive than formal. Dropout certainly reduces correlation between units and the corresponding redundancy but the latter is not always a good idea.

An issue with dropout is discrepancies in expectation, if dropout is applied without adjustments. More specifically, if a dropout probability p is applied to a unit u with output Y_u in a layer L_k of a neural network, the expected value of the output of u will be $EY_u = 0p + Y_u(1-p) = Y_u(1-p)$ while the expectation without dropout is obviously Y_u . Given that the entire network is used for testing and inference in the end (as if dropout were not there), discrepancies in expectation can have adverse affects on test error and generalization. As a result, the output Y_u of each unit u of a layer with dropout is multiplied by $1/(1-p)$ during training to achieve identical expected values in both cases.

The above is certainly a reasonable and valid adjustment, if the polynomial functions of layers L_m for $m > k$ contain only affine terms of Y_u . For example, this is the case when all polynomial functions are affine. However, the adjustment is incorrect when the polynomial functions contain higher-order terms.

For example, let us assume a unit u in a layer L_k of a neural network. If the random variables Z_u and Y_u represent the output u with and without dropout, respectively, then $EZ_u = p0 + (1-p)EY_u/(1-p) = EY_u$ but $EZ_u^2 = p0^2 + (1-p)EY_u^2/(1-p)^2 = EY_u^2/(1-p)$, assuming a constant probability dropout p . Consequently, if the polynomial function p_{k+1} of L_{k+1} only contains a quadratic term Y_u^2 of Y_u , the expected contribution of unit u to p_{k+1} is EY_u^2 with no dropout but $EZ_u^2 = EY_u^2/(1-p)$ with dropout probability p .

The current convention is to use dropout on individual units. This has worked so far because existing neural networks almost always use affine polynomial functions and thus they do not suffer from the above discrepancy. However, this is not the case with this theory. We will progressively move toward more complex polynomial functions.

In general, it is a design limitation to apply a dropout adjustment to individual units as is the current convention. Instead, the adjustment should be applied to the polynomial functions that use the corresponding units. In that case, the polynomial contribution of each unit u would be known and its output Y_u could be adjusted accordingly.

From a biological viewpoint, this is equivalent to moving the dropout functionality from the axon of a neuron to the dendrite of a receiving neuron; and it is more consistent with current biological evidence.

The modifications in the functionality of dropout should be clear from the above discussion. Irrespective of this issue, we use the term *probabilistic dropout* to refer to dropout of the above forms.

In volume II, we will introduce a more general form of dropout that operates on segments of the polynomial functions and is more consistent with the current biological record and models of neural activation in nervous systems. We call it *superposition* or *quantum dropout* because it allows a unit to be active for some parts of a network or its functions and dropped for others at the same iteration. Thus, in superposition dropout, training is not operating on a distribution with random selection at each iteration but rather on a superposition of neural structure in a fashion similar to the superposition of quantum states in quantum mechanics.

Typical values for the dropout probability p are usually 0.5 or less. However, the actual value of p depends on the desired ratio between the size of a model or layer (in terms of units) and the size of the corresponding mixture components. Hence, dropout values may vary across different layers or parts of a neural partition.

Dropconnect is a variant of dropout where the dropout probability p is applied to each connection in a layer instead of individual units. Similarly, in *spatial dropout*, the probability p is applied to each filter of a cross-correlation instead of each individual unit, which represents a spatial instance of the corresponding filter.

Stochastic depth is another variant in which the probability p is applied to a layer as a whole. When a layer is dropped in that sense, it is replaced with the identity function. Stochastic depth is important because it effectively trains a mixture of models that have different depths. It can also reduce the problem of *diminishing gradients* in deep networks. (The latter refers to the fact that gradients can decrease exponentially with each layer, if the product of the derivative of the activation function and the norm of the polynomial function of the layer is less than 1.0. The opposite, known as *gradient explosion*, is exponential growth of gradients when the above product is greater than 1.0.)

Spatial dropout and stochastic depth are two dropout methods that attempt to take the structure of a network into account instead of applying dropout uniformly across all parameters. *AutoDropout* is another such method that learns dropout patterns that are represented as masks applied to parameters (usually within a layer).

5.6.3 Network Pruning

Network pruning refers to techniques that reduce the size of a network by removing connections or units.

Two common methods are *optimal brain damage* and *optimal brain surgeon* and they represent a bias for smaller and simpler networks.

Both methods train a network until a local minimum is reached and then use a second-order Taylor expansion of the loss function to calculate the *saliency* of each connection, which is an estimate of the increase in the overall loss if the connection is removed and the rest of the connections are updated to minimize the effect. Given that the linear terms of Taylor expansions vanish at local minima, this step reduces to the calculation of the inverse of the *Hessian matrix*.*

Optimal brain damage uses gradient descent to obtain a diagonal approximation of the inverse of the Hessian matrix and thus avoids the relatively high computational cost of inverting the matrix. Optimal brain surgeon does not assume a diagonal matrix. It uses Lagrange multipliers to compute saliency but it does require the inverse of the Hessian matrix.

5.6.4 Metric Regularization and Weight Decay

We use the term *metric regularization* to refer to methods that augment the loss function of a model with a simple function of the norm of the model or the distance of the model from a reference point that is usually the null model. Hence, if the model can be represented as a point in a normed space[†] with norm $\|\cdot\|$, the regularized loss of a model M has the form

$$\ell + \rho(\|M\|), \quad (5.31)$$

where ρ is a simple function (usually a small constant or a monomial) that defines the relative significance of regularization with respect to loss. The higher the values of ρ , the more importance is placed on regularization.

We call the above *normed regularization*.

In case the model space is a metric space with metric $\|\cdot\|$ but not a normed space, the regularized loss can be defined as

$$\ell + \rho(\|M, 0\|), \quad (5.32)$$

where 0 represents the origin, or equivalently, a model where all parameters are zero. We call the above metric regularization.

*The Hessian matrix of a function $f: \mathbb{R}^n \rightarrow \mathbb{R}$ is an $n \times n$ matrix H such that $H_{i,j} = \frac{\partial^2 f}{\partial x_i \partial x_j}$.

[†]A *norm* on a vector space V over a field F of $\mathbb{R}$ or $\mathbb{C}$ is a function $\|\cdot\|: V \rightarrow \mathbb{R}$ that has the following properties:

- a. $(\forall v \in V) \|v\| = 0 \Rightarrow v = 0$ (positive definite)
- b. $(\forall v \in V)(\forall r \in F) \|rv\| = |r|\|v\|$ (absolute homogeneity)
- c. $(\forall u, v \in V) \|u + v\| \leq \|u\| + \|v\|$ (triangle inequality)

A *normed space* is simply a pair $(V, \|\cdot\|)$ where V is a vector space and $\|\cdot\|$ is a norm on V .

In general, every normed space is a metric space since a metric can easily be defined from a norm: $\|x, y\| = \|x - y\|$. However, the opposite is not true. Thus metric regularization is the most general of this type of regularization and we will use the term to refer to both metric and normed regularization. We will also drop the reference point for simplicity.

For regularization purposes, we usually represent a model M with the array ψM of its learned parameters. A common norm for those parameters is the L_p norm that is often indicated as $\|\cdot\|_p$. Thus, we define

$$\|M\|_p = \|\psi M\|_p = \left(\sum_i |\psi M_i|^p \right)^{1/p}. \quad (5.33)$$

We use the term L_p regularization for such regularization, irrespectively of the form of the function ρ . (Note that ψ has greater precedence than indexing. Thus ψM_i stands for $(\psi M)_i$.)

A most common form is L_2 regularization where the function ρ is defined as $\rho(x) = \gamma x^2/2$ and γ is a constant. In that case, the regularization term is proportional to the sum of the squares of the parameters of the model and the regularized loss is

$$\ell + \frac{\gamma}{2} \|\psi M\|_2^2. \quad (5.34)$$

The above regularization is often called *weight decay* because the derivative of the regularized loss with respect to a learned parameter w is now $\frac{\partial \ell}{\partial w} + \gamma w$ and the update to w is $\Delta w = -\lambda(\frac{\partial \ell}{\partial w} + \gamma w)$.^{*} In other words, the regularization term forces the learned parameters to decay proportionally with their value and with rate γ at each training iteration.

Weight decay is one of the most common regularization methods in deep neural networks. In this theory, we extend the definition to all cases where the parameter update is of the form $\Delta w = -\lambda(\frac{\partial \ell}{\partial w} + \gamma \frac{|w|^p}{w})$ and thus to all metric regularization of the form $\ell + \frac{\gamma}{p} \|M\|_p^p$ for $p > 0$. In that case, we define p to be the *exponent* of the decay.

We are not aware of any theoretical, biological, or other factors that would imply an advantage to the case of $p = 2$. Our experiments have shown that a values of $p \in [1, 2)$ are often as good or better than $p = 2$, albeit the differences tend to be limited. Even a value of $p < 1$ can temporarily be useful in late training stages as a method to prune out small parameters (but, in that case, extra conditions are required to handle small parameters that change sign).

^{*}Sometimes weight decay is defined as $\Delta w = -\lambda \frac{\partial \ell}{\partial w} - \gamma w$ but the two forms are equivalent (we can simply use a different γ that is equal to $\gamma\lambda$ and factor out λ).

5.6.5 Batch Normalization

In iterative neural network training methods, an *epoch* or *training epoch* refers to one or more training steps that span an entire training set. It represents a complete pass of the set through the training process. The process can be defined as *online* or *stochastic training* when the learning parameters of the network are updated once per training sample; or as *batch training* when the parameters are updated once per epoch. *Mini-batch training* is an alternative approach where the training set is partitioned into *mini-batches*, which are smaller subsets of the training set, each of size $b > 1$, and the learning parameters of the network are updated once for each mini-batch. (Typical values of b range from 16 or 32 to a few thousands.)

Batch normalization is a normalization method that is used with mini-batch training in neural network units or layers to make their statistics more comparable across mini-batches.

Formally, we can define batch normalization as follows.

Definition 5.9. Let $N = (G, F, Y, S)$ be a neural network as in definition 4.2. If batch normalization over mini-batches of size b is applied to a unit $u \in G$ at training iteration t , its output $Y_{u,t}$ is transformed to

$$\hat{Y} = \gamma \frac{Y - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta \quad : u, t | \epsilon$$

where β_u and γ_u are additional learning parameters while

$$\begin{aligned} \mu_{u,t} &= \frac{1}{b} \sum_{i=\lfloor \frac{t}{b} \rfloor b}^{\lfloor \frac{t}{b} \rfloor b + b - 1} Y_{u,i} \\ \text{and } \sigma_{u,t}^2 &= \frac{1}{b} \sum_{i=\lfloor \frac{t}{b} \rfloor b}^{\lfloor \frac{t}{b} \rfloor b + b - 1} (Y_{u,i} - \mu_{u,t})^2. \end{aligned}$$

Intuitively, μ_u and σ_u are the sample mean and standard deviation of the output Y_u of the unit u during each mini-batch. They are used to transform the output Y_u so that it has zero mean and unit standard deviation, if we ignore the small positive constant ϵ that has been added for numerical stability. The parameters β and γ are learning parameters that are added to ensure that the network is not constrained by the normalization.*

*Theoretically, the parameters β and γ can be chosen at each training mini-batch so that $\hat{Y} = Y : u, t$.

5.7 Overfitting and Generalization

Overfitting is an important learning phenomenon that has appeared with various simplified definitions in scientific literature. It is an integral part of generalization that has empirically been verified in numerous experiments and analyses in statistics and machine learning. Yet, similar to the larger and encompassing concept of generalization, it is not formally defined or well understood. We will attempt to understand both concepts more completely in this section.

In statistics, overfitting is often defined as a model or analysis that fits a data set closely or exactly but fails to fit additional data reliably. This definition is relatively accurate but it is phenomenal. It describes the effect and allows us to identify overfitting in many cases but it does not provide an answer or suggestion for the cause of it. Why would a model that fits a data set exactly or better than any other model fail on new data? Furthermore, the definition is often treated as though it implies that the cause of unreliable performance of the model on new data is its 'unreasonably' good fit of the training data. This is not always a sound argument or accurate interpretation. It is rather an empirical observation of the statistical correlation of two effects.

Overfitting is sometimes defined as learning the detail and noise in specific data to such an extent that it negatively impacts the accuracy of a model on new data. The model is postulated to extract variation and noise from the data as if those were real and essential parts of the underlying functions.

The above definition is probably accurate for a number of overfitting cases. However, if we indeed assume overfitting to be indicated as reduced accuracy on new data, then overfitting can occur when there is no noise in the training data and when we constantly transform data points or use each sample only once so that fitting detail and variation is hard or impossible. It can even occur when the model learns what it should learn, as we shall demonstrate shortly. Furthermore, even if we somehow assume this definition to be accurate for all cases, it still does not explain the cause of the effect. Why would a model fit variation or detail? Is such the nature of learning?

A similar definition resorts to model complexity and attempts to explain overfitting as excessive complexity in a model that results in learning specific characteristics or idiosyncrasies of a training set. However, this definition is equally problematic and only partly accurate. Overfitting can be aggravated by model size and complexity but it can occur even in cases where the size or complexity of a model is minimal and with an opposite relation to model complexity. For example, fitting a set of m points in the plane with an n degree polynomial results in error that can decrease with n if the x coordinates of the points are properly distributed towards the ends of their interval. (We will describe this fact in more detail later in this section.) Similarly, a very simple model, such as a parabola, can accurately fit the sine function in any interval of the form $[k\pi, (k+1)\pi]$ but it is highly inaccurate outside such an

approximation interval as shown in figure 5.1. In this case, we approximate

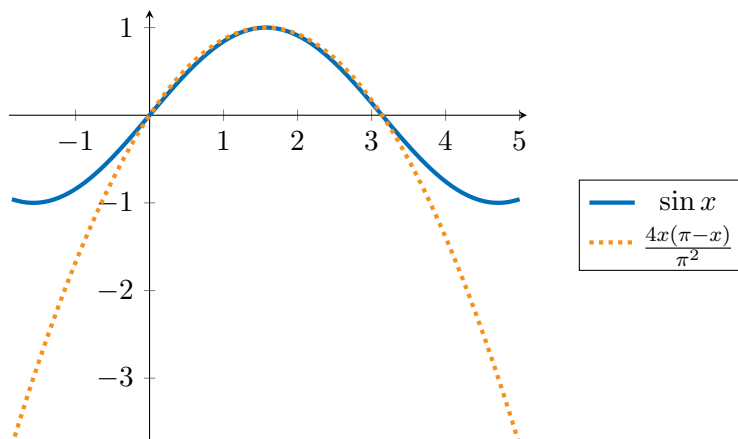


Figure 5.1: The sine function and an approximating parabola.

the function sine with the parabola $\frac{4x(\pi-x)}{\pi^2}$ in the interval $[0, \pi]$. We will examine this example in more detail because it corresponds to a constituent element that is essential for understanding overfitting and generalization.

In statistics, it is often assumed that an overfitted model contains more parameters than can be justified by the data. However, this assumption is qualitative, generic, and sometimes inaccurate even within a generous interpretation. Data do not include explicit or absolute parameter justifications of any kind. Such justifications can implicitly be derived from data using methods but they depend on the methods themselves. They also depend on the underlying functionality, the domain, its distribution, and even the model or the learning algorithm. If samples are non-uniformly distributed in a domain and data are collected with such a distribution, as is often the case, training on such data can result in overfitting the underlying function in regions with high density in the training set even when the overall model contains fewer parameters than those that are ‘justified’ by the data.

Generic qualitative definitions like the above do not really improve our understanding of our subject matter. Overfitting is a phenomenon that is closely related to generalization and as such it is considerably more complex.

In current practice, the above fact is reflected by a virtually exclusive reliance on empirical methods instead of theoretical ones to detect overfitting or estimate model complexity relative to the data. A validation set is usually used to estimate the generalization error during training. When there is statistically significant evidence that the error has started increasing, overfitting is assumed and training usually terminates. Related theoretical methods are as inaccurate or oversimplifying as to be practically useless.

Let us now try to understand overfitting and its relation to generalization in a more causal fashion.

5.7.1 Functional Distinction

We consider the sine-parabola example of figure 5.1 from the previous section. If all of our training data are in the interval $[0, 3]$, the failure to generalize in the interval $[4, 5]$ may be interpreted as overfitting in $[0, 3]$, assuming that the underlying function (sine) is unknown and thus its obvious incompatibility with the modelling function (parabola) is also unknown. Furthermore, if our validation set contains only points in the interval $[4, 5]$ and we start with a coefficient significantly lower than $4/\pi^2$, which will make the parabola flatter, the mean squared error in the training set will decrease as we increase the coefficient but the validation mean squared error will increase. The latter is consistent with the empirical process of validation to detect overfitting.

However, we argue that this case should not be deemed as overfitting. The size of the data set and the size of the model are not relevant in this example. The same applies to noise and model complexity. There is no noise in the data; and the parabola is not a more complex function than the sine. The parabola fits the data well in the interval $[0, 3]$ not because of its complexity but because of an incidental shape similarity with the function sine in that interval. It does not fit detail or noise but the actual shape of the underlying function as it should.

If anything, the above example is the opposite of overfitting, which is often called *underfitting* in the literature. The parabola is not complex enough to fit or overfit the sine function. If we are to use such an approximation, we should partition the input domain into intervals of the form $[k\pi, (k+1)\pi]$ and use a separate parabola in each interval. This would require a data set and a set of parabolas that grow linearly with the number of intervals. For multidimensional problems, which are invariably the case in real applications, such an approach would result in a number of approximation regions that grow combinatorially. Even if we ignore the combinatorial growth, we cannot really know when and where our modeling is incompatible with the underlying function, because the underlying function is unknown.

We can now consider the above approximation issue for large learning systems and multidimensional data sets that represent complex underlying functions. Shape similarities and incompatibilities between the functional forms of the learner and those of the environment will be different in various regions of the domain. The possibilities are combinatorial and so numerous that if we could visualize the potential error across the multidimensional input domain when we apply the same resources across the domain, we would most likely observe the equivalent of a multidimensional mosaic. Different clusters of the mosaic represent different degrees of similarity (or dissimilarity) between the functional forms of the learner and the environment. Depending on how data from the training, test or validation set are distributed relative to the mosaic, we can have various generalization or overfitting outcomes.

In general, learning systems tend to have significantly lower Kolmogorov complexity than the functionality in application domains or environments.

Learning systems usually have specific or generic functional forms and architectures. They admit a relatively short description that is the specification of the learner on a software framework or the specification of a nervous system in hereditary information. On the other hand, domains and environments often support a diverse range of dynamically changing functionality and its instances vary significantly across the domain or environment. Learners may develop a functional bias if functionality is relatively stationary, but they cannot possibly have advance knowledge of all or most of the functional forms of the environment at each instance.

In nature, evolution can shift functional forms and stationarity. It usually leads to energy minimization through evolutionary pressure. If a learner were considerably more complex than its environment, the learner would most likely waste energy and neural or other resources. The complexity of an organism can significantly grow over time, of course, but the change represents a double feedback: it is often an adaptation to a changing environment but it affects the environment too, increasing its complexity. Genetic information can be seen as a learned encoding of an environment produced by evolution. Conversely, an environment can be defined as the aggregate of the genetic information in it.

The same can apply to cognition and acquired skills and ideas. In 1689 John Locke argued that “nothing is in the mind without being first in the senses, except for the mind itself”. Given that our sensory input comes from our environment, the above statement implies that everything that is in the mind must have originated in the environment. Locke’s arguments against innate ideas were criticized by Gottfried Leibniz, who adopted a rather extreme view of innate cognition, assuming that all thoughts and actions are innate in a rebuttal he wrote in 1704.

In our view, there are both innate and acquired components in cognition. Evolution can actually blur the line between the two; but the environment is the ultimate source for both acquired and innate or genetic structures, faculties, skills, and ideas. All those are outcomes of learning. The main difference is the time-scale. Cognition is learning in the span of an organism’s lifetime while evolution is learning across the timelines of species. Science and recorded knowledge is learning and aggregate cognition over a prolonged and possibly indefinite time-scale.

For example, *universal grammar* is a theory in modern linguistics that postulates a genetic component in language comprehension and thus an innate ability to process and understand natural language. However, it is not clear how much of our language processing abilities are acquired and how much are innate; or when humans developed such an ability. The *origin of language* and its relationship with human evolution and cognition is a subject of study in linguistics – and an open question with contrasting approaches. Estimates of the underlying transition in humans range widely from as late as 50,000 years ago to as early as the beginning of the human genus more than 2.5 million

years ago. Language represents a form of aggregate cognition, potentially with supporting genetic components, that must have occurred gradually. It was enabled by sensations, interactions, and information from the environment but it has also enriched and altered the environment through science and civilization.

Irrespective of the nature of innate abilities and ideas, there is an inevitable expansion of information in learning processes such as cognition or evolution. The genetic information of an ecosystem is much larger than the genetic footprint of any single organism in the ecosystem. The innate abilities of any organism or species represent only a small part of all the innate abilities of all organisms and species in the ecosystem. The same applies to acquired ideas and skills. By definition, the complexity of an environment is greater than the complexity of any constituent part and usually by a factor that is proportional to the population of the environment (assuming that we do not include microorganisms or species of significantly reduced complexity in our estimate of the population).

We call the above property *functional distinction* or simply *distinction*. Environments and domains have considerably more complexity and diversity in their functionality than learners in those environments or domains. As a result, shape incompatibilities between learning and environmental functional forms will be common. The error mosaic will thus be detailed and extensive. Any disproportional allocation of resources to any region of the mosaic will most likely have an adverse effect on generalization but a positive effect on the training accuracy in the region and potentially the entire training set.

Conversely, if there is limited or zero functional distinction between an environment and its learners, overfitting will be considerably harder or impossible. For example, if both the environment and the learners contain only polynomial functions, overfitting (and generalization) will be entirely dependent on the ability of a learner to estimate the degree n of an underlying polynomial from the training data. Data size, detail, and noise can potentially affect the numerical robustness and stability of the learning process but their effect on overfitting and generalization is otherwise diminished (assuming that the training set contains at least $n + k$ distinct samples, for some small $k > 0$).

As a more concrete example, we can use the *Python* package NumPy and its function `Polynomial.fit` to fit polynomials to data. The function is an implementation of *polynomial regression* (or *approximation*). It assumes a training data set and a degree n and it uses the method of *least squares* to derive a polynomial fit to the data. Given that there is always finite numerical precision in such computations, we can limit the precision of the derived coefficients to 10 decimal digits (thus having a precision of 10^{-10} for each coefficient). Then if we consider the function $f(x) = (x - 4)^2$ and the training set $S = \{(x, (x - 4)^2) / x \in \{0, 1, \dots, 8\}\}$, we can use an eight degree polynomial to fit the nine points in S . (The polynomial is specified by nine

coefficients, one for each power x^k for $k = 0, 1, \dots, 8$, while the data points in S generate a system of nine linear equations that admits a unique solution.) The result is $p(x) = 16 - 8x + x^2$ and it is shown in figure 5.2. In other words,

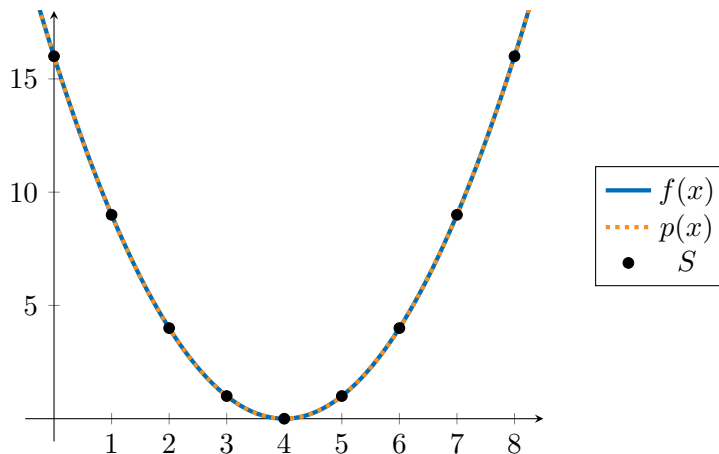


Figure 5.2: Polynomial fit detects the parabola in the data.

the polynomial fit discovered that the underlying function is a parabola and thus has degree two even though we used a polynomial of degree eight.

However, the above method is not robust under noise or functional distinction. If we introduce a small offset in the function and the data for $x = 5$, redefining $f(x) = (x - 4)^2 + (x = 5)$, the new data set N will only differ from S at single point, containing the point $(5, 2)$ instead of $(5, 1)$:

$$N = \{(0, 16), (1, 9), (2, 4), (3, 1), (4, 0), (5, 2), (6, 4), (7, 9), (8, 16)\}.$$

This introduces functional distinction because our underlying function f is no longer polynomial. The result is that the approximating polynomial p , shown in figure 5.3, is no longer a parabola but a polynomial of degree eight:

$$\begin{aligned} p(x) = & 16 + 3.2x - 27.2x^2 + 27.172222222x^3 - 13.2555555556x^4 \\ & + 3.584722222x^5 - 0.5430555556x^6 + 0.0430555556x^7 \\ & - 0.001388889x^8. \end{aligned}$$

It can be argued that the polynomial p now fits the detail or noise in the data, depending on whether we consider the offset at $x = 5$ to be part of the function f or just an error or noise in the data set N . p is still a good approximation of f in the interval $[0, 8]$ but the error increases quickly and dramatically outside that interval.

Learning systems should always assume functional distinction and use robust learning methods to handle it. Functional coincidence will be a fortunate outcome, if it happens, but learners should not be based or designed around it. Otherwise, the performance (and perhaps survivability) of a learner

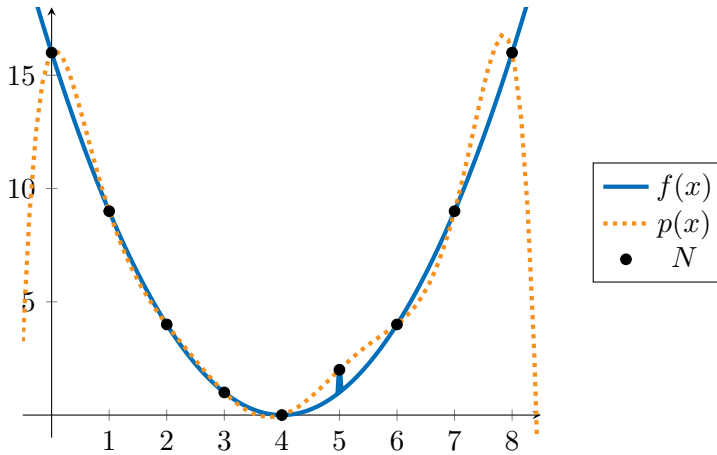


Figure 5.3: A single offset introduced functional distinction and poor approximation outside the training interval.

can degrade significantly even with minor changes in the environment as the above example with the small parabola offset demonstrates.

Functional distinction represents the simple fact that functions have distinct forms that are often incompatible (under a set of transformations). There is great diversity in functional forms and a necessary discrepancy in functional complexity and diversity in efficient or evolved environments. In this theory, we postulate that functional distinction is a primary factor and cause for overfitting. Learners attempt to model their environment with their own functional forms that are only partially compatible with the functions of the environment. This phenomenon is aggravated by the increased complexity of the environment, which forces learners to use forms that have large sets of parameters and thus larger Kolmogorov complexity. Distinction is thus a significant factor not only for overfitting but also generalization.

5.7.2 Non-local and Locality-sensitive Approximation

Functional distinction is not the only cause or factor in overfitting and generalization phenomena. The fact that a small change at a single data point in the training set N of the previous subsection created such a big difference suggests that our approximation method is overly sensitive to functional distinction or noise in the data. Consequently, factors that affect or produce such sensitivity are relevant to our subject and our intention to understand the above phenomena.

Some of the sensitivity comes from our approximation functions, which are polynomials in the examples of this section. Let us understand this fact in more detail.

Theorem 5.7 (Weierstrass approximation theorem). *If $[a, b]$ is a real interval and $f: [a, b] \rightarrow \mathbb{R}$ is a continuous function then there exists a sequence of*

polynomials $\langle p_n/n \in \mathbb{N} \rangle$, where p_n has degree at most n , that approximates f with uniform convergence in the interval $[a, b]$ as n tends to infinity:

$$\lim_{n \rightarrow \infty} \sup \{ |f(x) - p_n(x)| / x \in [a, b] \} = 0.$$

Proof is omitted. A reference that contains the proof can be found in section 5.11. $\square$

The Weierstrass approximation theorem implies that we can approximate any continuous function f in any interval $[a, b]$ within any given $\epsilon > 0$ using polynomials. However, the theorem has a number of limitations. It is an existential theorem that implies the existence of the polynomials but it does not provide a method to construct them. It does not specify any bounds for the approximation error outside the interval $[a, b]$, if f is defined outside the interval. Finally, it does not provide any rate of convergence for the error within the interval or how the convergence is affected by the length $b - a$ of the interval $[a, b]$.

We can ignore the convergence issues, given that our goal is not to use polynomials in the most effective way but rather to understand overfitting and generalization.

A polynomial $p(x)$ of degree n can be uniquely defined by its values at $n + 1$ distinct points on the x -axis.* As a result, identifying each polynomial p_n is equivalent on providing a data set s_n for it. Given that p_n has degree at most n , s_n does not need to contain more than $n + 1$ points. However, when the points in s_n are equally spaced in the x -axis, *Runge's phenomenon* can occur. The latter refers to oscillation at the edges of the interval when high degree polynomials are used. More specifically, if we try to approximate the function $f(x) = 1/(1 + x^2)$ in the interval $[-5, 5]$ with polynomials derived from equally spaced points in the interval, the approximation error increases indefinitely with the degree of the polynomial:

$$\lim_{n \rightarrow \infty} \sup \{ |f(x) - p_n(x)| / x \in [-5, 5] \} = +\infty.$$

Runge's phenomenon is shown in figure 5.4. The function $1/(1 + x^2)$ is approximated in the interval $[-5, 5]$ with two polynomials that are derived from equally spaced samples in the x -axis. The degrees of the two polynomials are six and ten, respectively.

Runge's phenomenon occurs primarily because of the growth of derivatives (the n -th order derivative in this case) and a quickly increasing *Lebesgue constant* due to equidistant interpolation points. We will not elaborate on

*In polynomial approximation, the points (x_i, y_i) for $i = 0, 1, \dots, n$ define a system of $n + 1$ linear equations $Vx = y$, where $V_{i,j} = x_i^j$. V is a *Vandermonde matrix* and its determinant is called the *Vandermonde determinant*. Considering its roots, we can easily prove that the Vandermonde determinant is $\det(V) = \prod_{i < j} (x_j - x_i)$. Thus the determinant is non-zero and V is invertible when all the x -axis coordinates are distinct.

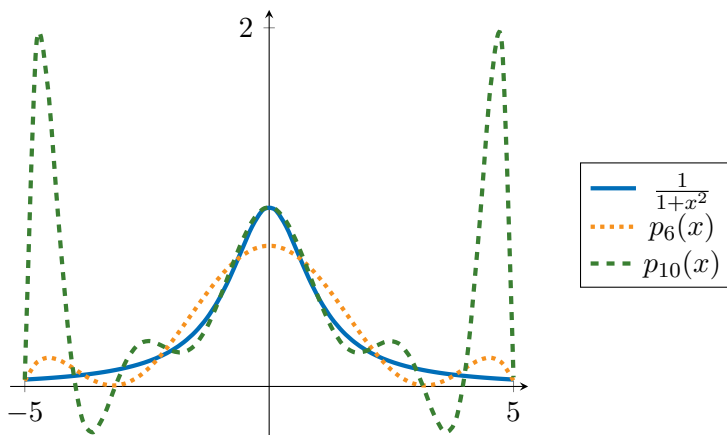


Figure 5.4: Runge's phenomenon. The function $1/(1+x^2)$ is approximated by two polynomials of degree six and ten, respectively. The polynomials are derived from equally spaced samples in $[-5, 5]$.

the causes of the phenomenon further because they are rather incidental to our goal.

There are various methods that address Runge's phenomenon. The error can be reduced by distributing the data points more densely toward the ends of the interval $[a, b]$. A common approach is *Chebyshev nodes*, which are defined as follows

$$(\forall i \in \mathbb{N}_n) \quad x_i = \frac{a+b}{2} + \frac{b-a}{2} \cos\left(\frac{2i+1}{2n}\pi\right) \quad (5.35)$$

for the interval $[a, b]$, n being the number of nodes. Using n Chebyshev nodes, we can generate polynomial approximations p_k with diminishing error for $k < n$. Figure 5.5 shows polynomials p_6 and p_{10} of degree 6 and 10, respectively, derived from 11 Chebyshev nodes for the function $1/(1+x^2)$ in the interval $[-5, 5]$. The approximation error is now considerably smaller for p_{10} than for p_6 . Chebyshev nodes actually guarantee that the approximation error converges to zero as the degree of the polynomial goes to infinity.

Constrained optimization (that limits the derivatives of the polynomial approximation) and piecewise polynomials such as *splines* can also be used to address Runge's phenomenon.

However, all of the above methods only work within the interval $[a, b]$. Chebyshev nodes and constrained optimization suffer from increased error outside of the interval $[a, b]$ while piecewise methods are not meant to be extended beyond the points they connect – their error usually grows quickly in such regions.

For example, if we expand the plot of figure 5.5 even by a mere 0.3 on either side, as we have done on figure 5.6, we begin to see how quickly the error grows outside the interval $[-5, 5]$.

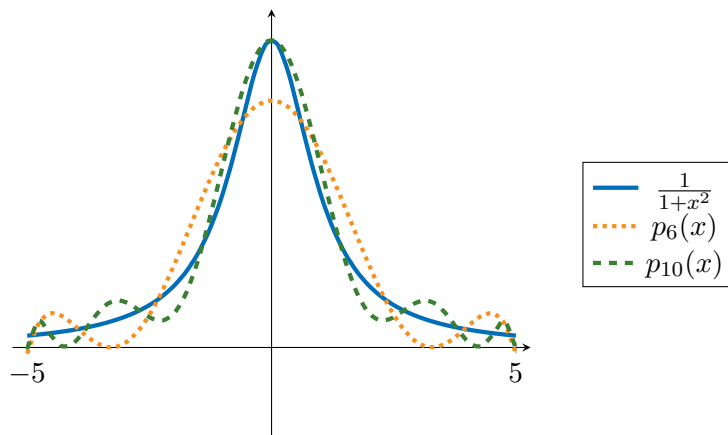


Figure 5.5: Chebyshev nodes eliminate Runge’s phenomenon. They guarantee that the approximation error converges to zero as the degree of the polynomial approximation goes to infinity.

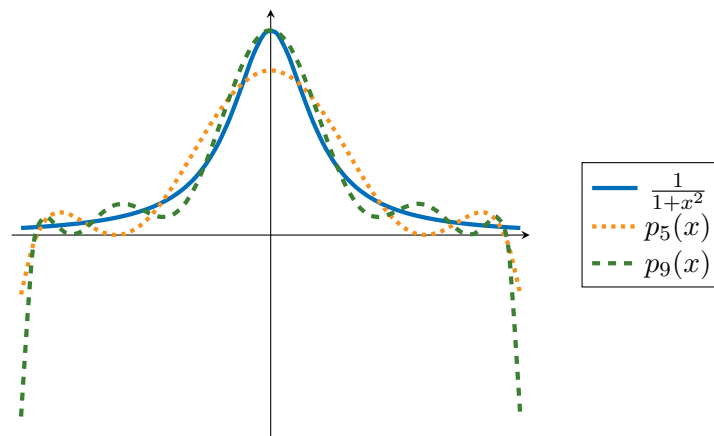


Figure 5.6: Chebyshev nodes do not address approximation error outside the interpolation interval.

In this case the underlying function is $1/(1+x^2)$, which converges to 0 when x tends to infinity. Given that non-constant polynomials always diverge to infinity when x tends to infinity, such a large error is bound to take place no matter how good the approximation is within the interval $[a, b]$.

However, the increased error occurs even with underlying functions that diverge to infinity. In its most general case, this is a problem of mismatch between rates of change. We will discuss this important issue shortly.

A known issue with polynomial approximation is the fact that it is *non-local*, meaning that the value $p(x)$ of the approximating polynomial at a point x may strongly depend on data samples (x_i, y_i) in the training set even when $|x - x_i|$ is large. This is widely considered a drawback for interpolation and approximation purposes. However, the same does not necessarily apply to

learning.

The *convex hull* or *convex closure* of a set of data points is simply the smallest convex set that contains those data points. When the data points represent a training set, we use the term *training convex hull* for their convex hull.

In general, *interpolation* refers to estimates of an unknown function within the ‘range’ of a training set while *extrapolation* refers to such estimates outside the above range. In multidimensional domains, the concept of such a range is ambiguous. It can refer to the training convex hull or the *training hyperrectangle*, which is the minimum *hyperrectangle* that contains the training data points.*

In spaces of low dimensionality, it is not clear whether one definition should be preferred over the other. For applications in two or three dimensions (such as in computer graphics), the minimum hyperrectangle is often used. However, the situation changes with higher dimensionality. The approximation accuracy of multidimensional interpolation methods usually starts degrading in a visible manner outside the training convex hull. For multidimensional domains and learning purposes, we thus consider it more appropriate to define interpolation and extrapolation based on the training convex hull instead of the training hyperrectangle.

Using the above definition, we can argue that learning is primarily about extrapolation; and that interpolation, when and if it occurs, tends to involve data points that are far from the new sample x under consideration or inference. (Such distances can increase significantly if we switch to the hyperrectangle definition, which is another argument against its use that effectively blurs the line between interpolation and extrapolation.)

The above arguments derive from the fact that learning is invariably applied to multidimensional domains. Such domains have combinatorially large magnitudes and volumes. Even if we consider only the vertices of the domain’s boundary, they correspond to data points where each coordinate has either its minimum or maximum value. Thus their number is 2^n , where n is the dimensionality of the domain. However, n can be large for real applications. For example, when the input is color images of $k \times k$ pixels, n is $3k^2$ since there are 3 values for the red, green, and blue color components in each of the k^2 pixels of each image. Thus there are 2^{3k^2} vertices. This is a prohibitive magnitude even when k is as small as 20 or even 10.

Moreover, most of the volume of a domain is near its vertices rather than its center or middle.† Hence, the expected ratio of the volume of a training

*A *hyperrectangle* or *n-orthotope* is a generalization of a rectangle for higher dimensions and it is simply the Cartesian product of n intervals. The *minimum hyperrectangle* of a set of points is also known as their *minimum bounding box*.

†This is demonstrated by the fact that the ratio of the volumes of an n -dimensional *hypersphere* with diameter d and an n -dimensional hypercube with edges of length d is $\frac{\pi^n}{2^n \Gamma(1+n/2)}$, where Γ is the *gamma function*. Given that, for $n \in \mathbb{N}$, $\Gamma(n + 1/2) = \pi^{1/2} \prod_{k=1}^n (k - \frac{1}{2})$ and $\Gamma(n) = (n-1)!$, the above ratio converges to 0 as n tends to infinity.

convex hull over the volume of its domain is usually a small or trivial number that furthermore converges to zero exponentially as the dimensionality n of the domain tends to infinity.

Consequently, if we randomly pick a point in a multidimensional domain to infer the value of an underlying function, the point will most likely be outside the training convex hull and it will thus require extrapolation. Of course, samples are not uniformly distributed in domains; but there are often heavy tails in such distributions and thus a non-trivial probability of occurrence of samples in relatively non-trivial volumes across the entire domain.

For example, the *MNIST data set* contains 60,000 training images of handwritten digits and 10,000 testing images. Each image contains 28×28 gray-scale pixels but the minimum bounding box is at most 20×20 pixels. We can thus ignore the rest of the image and treat each picture as a point in a 400-dimensional space. We can now scale each image down to $n \times n$ pixels and compute how many test samples are inside the training convex hull. Table 5.1 shows the percentage of the test set samples that fall inside the training convex hull for image width and height $n = 2, 4, 6, 8, 10$, and 12 .*

Image Size	Test Set Coverage
2×2	99.43%
4×4	17.02%
6×6	0.43%
8×8	0.04%
10×10	0.00%
12×12	0.00%

Table 5.1: Percentage of the MNIST test set in its training convex hull with reduced images.

If we assume the above percentage to be an estimate of the probability that a random sample will be inside the training convex hull, we can see that the probability diminishes quickly with dimensionality. It is trivial for 36 dimensions while it is so small for 100 dimensions that cannot be measured with a test set of only 10,000 and thus appears as zero.

For more realistic cases, such as images from data sets like Imagenet that

*Computing the convex hull of a set of points $A = \{a_1, \dots, a_n\}$ is computationally expensive and it can be numerically challenging sometimes. On the other hand, inclusion of a point in a convex hull can be determined simply by solving a system of linear equations. Indeed, a point b is in the convex hull of the set of points A iff

$$(\exists x_1, \dots, x_n \in [0, 1]) \quad \sum_{i=1}^n x_i a_i = b \quad \text{and} \quad \sum_{i=1}^n x_i = 1. \quad (5.36)$$

are reduced for neural processing, n is of the order of a few hundred and the number of pixels and thus dimensions in each image is of the order 100,000. Actual images, such as those produced by the camera of the iPhone X are 12 megapixels. In biological vision systems, the number of dimensions is substantially larger. For example, the resolution of the human eye has been estimated to be about 576 megapixels.

For such dimensionality, the probability that a sample will fall in the convex hull of any realistic training set is practically zero.

Inclusion in the convex hull may be an appropriate definition of interpolation in multidimensional spaces but it is also a binary metric. A point is either inside or outside any given convex hull. A more informative metric would be the distance of a point to the training convex hull. Indeed, a point can be outside but near the convex hull and thus be approximated with decent accuracy (e.g. using the *k-nearest neighbor rule*). The distance of a point to a convex hull can be estimated using the *Gilbert–Johnson–Keerthi distance algorithm*, which computes the distance between two convex sets. (Alternatively, the sum $\sum_{i=1}^n x_i$ from equation (5.36) is a quantity that tends to increase with the distance from the convex hull. Hence, we can associate each point b with the value $\beta_b = \min \sum_{i=1}^n x_i$ under the constraint $\sum_{i=1}^n x_i a_i = b$. We call this the *convex hull expansion method* because each point b belong to a multiplicative expansion of the convex hull by β_b .)

In the end, the overall argument remains the same. Distances within a domain, which is often a hyperrectangle, grow exponentially with the number of dimensions of the domain or hyperrectangle. Discrete domains are usually multidimensional grids; and the grid points are exponential with the dimensionality of the grid. As a result, training sets represent very small or trivial fractions of their domains. They are necessarily sparse and if they are not concentrated within a small volume, the distances between their data points will tend to be large. Therefore, not only interpolation represents a small minority of the inferences in multidimensional domains, but also a large fraction of the interpolations rely on data points that are far from each other.

Consequently, non-local approximation is not actually a drawback for learning in multidimensional domains but rather an important and necessary property. It is correctly considered a drawback for applications in low-dimensional spaces (such as two-dimensional or three-dimensional modeling in computer graphics or engineering). This is because interpolation can be considerably more common than extrapolation in such domains and distances are smaller. Consequently, non-local dependencies are not as important per the principle of locality (as defined in 1.5). However, the situation reverses quickly with higher dimensionality; and non-local approximation thus becomes an important factor and property for learning.

Moreover, we would like our approximation or learning methods to be *locality-sensitive** in the sense that small changes in the training data, such

*This is analogous to *locality-sensitive hashing*, which is a partitioning process that

as adding a modest amount of noise, should only effect limited changes in the learned model (with high probability). We will explain and quantify this concept further in volume II. We mention it here in order to emphasize the differences between approximation locality, general locality, and sensitivity to sample locality. Locality in a neural structure refers to locality of the computation during the processing of a single sample or data point. It is thus irrelevant to approximation locality, which, by definition, is locality across multiple samples within a training set. Sensitivity to sample locality does refer to the locality of training samples but it does not imply approximation locality or derive from it.

In short, locality in the computation, sensitivity to sample locality, and non-local approximation are important and desirable properties in learning systems. There is value in local approximation but there is also an inherent high cost. For example, local piecewise methods such as splines are probably among the most robust and accurate for interpolation purposes but they are usually among the least fortunate for extrapolation.

Sensitivity to sample locality can provide protection or robustness against noise and functional distinction, reducing overfitting pressure. It is thus an important factor for learning and generalization overall. Conversely, its absence can enable or aggravate overfitting and reduce generalization.

Polynomial approximation suffers from reduced sensitivity to sample locality under functional distinction as shown in the example of figure 5.3 where a small change in a single data point changed the polynomial approximation from a parabola to an eight degree polynomial. Without specific constraints in the optimization, polynomial approximation is particularly subject to overfitting, too. Yet, the non-locality of polynomial approximation is not the true cause and certainly not the only cause of those issues.

There is another and arguably more important factor for overfitting and generalization. We can best understand it with an example.

Let us consider the diagonal function $d(x) = x$ and a ‘noisy’ version of it defined as $s(x) = x + \sin(x\pi/2)$. In order to avoid Runge’s phenomenon, we will approximate the function s in the interval $[-5, 5]$ using 11 Chebyshev nodes and two polynomials p_5 and p_9 of degree 5 and 9, respectively. The Chebyshev nodes are thus at coordinates

$$x_i = 5 \cos\left(\frac{2i+1}{22}\pi\right), \text{ for } i \in \mathbb{N}_{11}. \quad (5.37)$$

The function s , the data set S that contains the Chebyshev nodes, and the approximating polynomials p_5 and p_9 are shown in figure 5.7

Despite the fact that polynomials are not periodic functions and they do not have periodic terms, p_9 approximates the underlying periodicity of s so accurately in the interpolation interval that its plot can barely be distinguished from the plot of s . In general, the error is expected to diminish with the

hashes similar items into the same value with high probability.

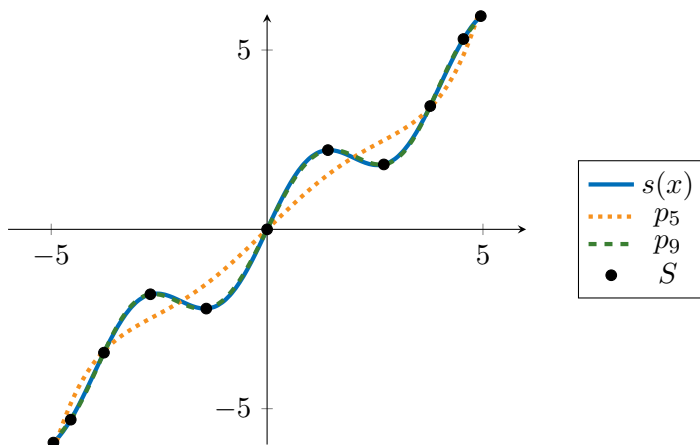


Figure 5.7: Approximating diagonal data with sinusoid displacement or 'noise' using 11 Chebyshev nodes in the interval $[-5, 5]$ and polynomials p_5 and p_9 of degree 5 and 9, respectively.

degree of the polynomial and it is already trivial for p_9 but clearly not as small for p_5 .

On the other hand, if we look at extrapolation space and thus outside the interval $[-5, 5]$, p_5 is a better approximation of s than p_9 although both polynomials are so poor that they can hardly be called approximations anymore. This is shown in figure 5.8 where we have just extended the plot range to $[-6, 6]$ and the polynomials are already two or three times larger in magnitude than the function s .

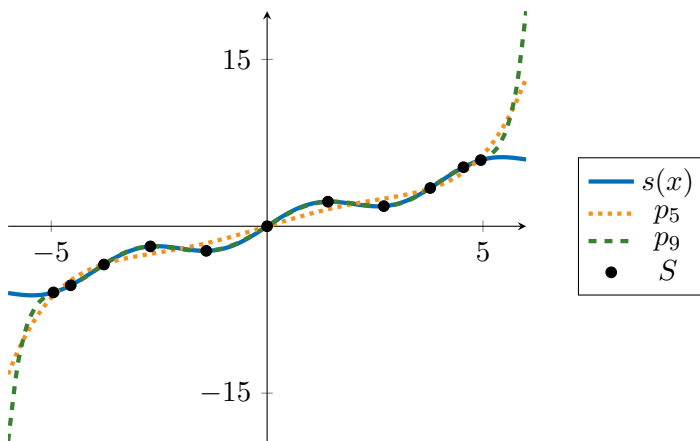


Figure 5.8: Extending the plot of the approximating polynomials p_5 and p_9 (of degree 5 and 9, respectively) to the range $[-6, 6]$.

We will attempt to understand the source of such large errors in the next subsection.

5.7.3 Finite Differences

It can be argued that great errors are inherent in the nature of polynomials since they diverge to infinity as x goes to infinity. However, in the examples of the previous subsection, both the diagonal function d and the function s likewise diverge to infinity. The reason for the large extrapolation error is not divergence but rather a great discrepancy in the *rate of change* (a.k.a. *growth rate*) of the associated functions. Using the Θ -notation* (pronounced *big-theta notation*) we can see that $s(x) = \Theta(x)$ but $p_5(x) = \Theta(x^5)$ and $p_9(x) = \Theta(x^9)$. This is the reason that p_9 deteriorates much faster than p_5 in extrapolation space, while d and s remain good approximations of each other (they both grow linearly). The ratio $s(x)/d(x)$ converges to 1 as x goes to $\pm\infty$, while $|p_n(x)/d(x)| = \Theta(x^{n-1})$ for any polynomial of degree n .

The above is not an incidental discrepancy or a limitation that is particular to polynomials. Functions represent change. In differentiable functions, the rate of change is effectively represented by the derivatives of the function. The *Taylor series expansion* furthermore implies that the derivatives are sufficient to define a function uniquely when they exist and the Taylor series converges. Intuitively, change defines functions because functions represent change.

For non-differentiable, discontinuous, or discrete functions, the *finite difference*

$$\Delta_h f(x) = f(x+h) - f(x) \quad (5.38)$$

between points $x+h$ and x (or its differential version $\Delta_h f(x)/h$ or some expectation over a distribution of such differences) can be used instead of the derivative. Higher orders of such differences can be used in place of higher order derivatives. Expectation can also be used to identify trends within the domain or specific subsets of it. We will collectively refer to all of these as *finite differences*.

Approximation and optimization methods cannot generalize well if they do not use inductive methods over the finite differences of a function across its domain. For example, our polynomial approximations p_5 and p_9 never really detected or weighted the fact that there is a linear trend in s and that it eclipses the sinusoid variance as x tends to infinity. If we do take that fact into account, the best polynomial approximation of s is the diagonal function $d(x) = x$. All other polynomial functions will diverge from s sooner or later.

In conclusion, there are four main factors that affect overfitting and generalization: functional distinction, sensitivity to sample locality, non-locality in the approximation, and finite difference induction. Functional distinction is simply a fact for real world environments and efficient learning systems. Sensitivity to sample locality and non-local approximation are desirable properties for learning. However, inductive methods over finite

* Given a normed vector space V with norm $\|\cdot\|$ and two functions $f: \mathbb{R} \rightarrow V$ and $g: \mathbb{R} \rightarrow \mathbb{R}$, we write that $f(x) = \Theta(g(x))$ iff there exist $c_0, c_1 \in \mathbb{R}_+$ and $x_0 \in \mathbb{R}$ such that $c_0 g(x) \leq \|f(x)\| \leq c_1 g(x)$ for all $x \geq x_0$.

differences represent an essential and indispensable part of this theory and a postulated foundation for all learning. This is a main axiom in Mathesis. We will formally define it in the next section and apply it throughout the theory.

5.8 Coherence

In 1704, Gottfried W. Leibniz famously stated that “nature does not make jumps” (“la nature ne fait jamais des sauts”). About half a century later, Carolus Linnaeus, who worked on biological *taxonomy*,* made the same statement in Latin, “natura non facit saltus”. The Latin version is commonly used in natural science and philosophy as a reference to the principle of gradual change in nature. The principle was also an essential component of Charles Darwin’s theory of the evolution and it was reinforced by it. Gradual change is an indispensable component of learning and evolution because it is a primary operational mode of nature – and vice versa.† Consequently, it is a crucial part of this theory, too.

Leibniz’s principle is sometimes interpreted to imply *continuity*, which means that arbitrary small changes in the output of a system can be produced by sufficiently small changes in its input. However, this interpretation is not always accurate or applicable. Domains can be discrete. Nature may be quantized at its minimal scale. The perception of a “jump” is always relative to an observer. Small discrete changes are jumps by definition, but if they are sufficiently small, they may be perceived as continuity. We thus interpret Leibniz’s principle as an argument for relatively limited changes in nature and a gradual progression of processes and entities, which is essential for learning and the overall structure and form of the system. We define the above as the *principle of coherence*. We will provide a formal definition of coherence later in this chapter.

The term *coherence* is actually used in various scientific fields and areas including physics, computer science, signal processing, statistics, and linguistics. There are many different definitions and applications but there is always

**Taxonomy* is a branch of biology that describes, names, and classifies organisms primarily based on their shared characteristics or evolutionary and genetic relationships.

†In evolutionary biology, *phyletic gradualism* is a model of evolution that assumes that *speciation*, which is evolution of populations into distinct species, is gradual, relatively slow, and uniform. On the other hand, *punctuated equilibrium* states that populations demonstrate *stasis*, which is little evolutionary change, across most of their geological history after their appearance in the fossil record.

The above two models may seem contrasting but they are merely variations and simplifications of gradualism. Evolution is gradual but not uniform; and even if it were uniform, it could still generate spurts and periods of relative stability. Furthermore, changes in the fossil record may seem sudden sometimes but only because of the magnitude of geological scale. Nature progresses in a gradual fashion and its record contains variation not only as an inherent property of a non-stationary statistical process but also as an outcome of various environmental, geological, and even extraterrestrial factors and events that can extensively affect life and evolution.

an underlying concept or property of gradual change or similarity (that often derives from gradual change) that is measured by coherence.

Let us examine some cases in more detail.

In physics, coherence is a property that measures waveform similarity and results in stationary *interference*.*. It occurs when two (or more) wave sources have the same frequency and waveform and constant phase difference. It is one of the most important concepts in optics because it relates to the ability of light to exhibit interference effects; and it has various applications including photolithography, laser printers, optical disc drives, fiber-optics, laser cutting and welding, DNA sequencing, and laser surgery.

In signal processing, coherence is defined in terms of spectral density and it measures the accuracy in predicting a signal from another (using an optimal linear least squares function). In statistics, it is used in time-series and spectral analyses as an extension of correlation.

In computer science, coherence is usually defined in a qualitative way as similarity between data, properties, or elements of computations. Coherence in data is used for data compression. Locality of reference (which is a form of coherence in memory access) and *memory coherence* are concepts that affect system design in computer architectures. Coherence usually affects or drives algorithm design, too. It often leads to *incremental processing*. If entities change in a limited manner most of the time, storing or processing the change will generally require fewer computing and storage resources. *Subdivision methods* (such as *divide-and-conquer methods*) sometimes divide a problem into sub-problems that have increased coherence or coherence that can be exploited in a more direct way. A typical example is the *scan-line algorithm*, which is used in computer graphics to fill a shape with color or determine the visible surface of an object.

Coherence is used heavily in computer graphics. There are more than thirty different types of coherence in related literature and a comparable number of applications. We provide a partial list in table 5.2.

We can understand the cause of such common occurrence and the many forms and applications of coherence in computer graphics, if we regard its various instances as manifestations of the principle of coherence in nature and thus the visual field (which is merely a visual representation of nature). For example, if the surfaces of object somehow had extraordinary displacements that looked like a porcupine spines that were many meters long, spatial and object coherence would be exceedingly reduced and the computational costs of various computer graphics methods would increase significantly. If the spines were kilometers long, there would virtually be no spatial or object coherence. Objects would be spatially so entangled that it would be hard or impossible to tell which spine belonged to which object or where one object began and another ended.

Coherence is not an incidental property of nature. It is the principle

* *Interference* occurs when two waves superpose to form a composite wave.

Coherence	Description
<i>Area</i>	Successive or neighboring pixels in an image usually have the same or similar values. Thus values can be computed for a subset of the pixels and interpolation used for the rest.
<i>Depth</i>	The depth or distance between points on a surface of an object and the viewer change gradually resulting in similar depth orderings of objects in adjacent pixels.
<i>Object</i>	Object relationships and the likely presence of the same objects in spatial neighborhoods allow computations like intersection, clipping, or ordering to be performed on objects first to reduce the complexity and computational load on low-level geometry.
<i>Spatial</i>	There is limited spatial variation of volumes in the visual field (because they usually represent parts of contiguous objects).
<i>Temporal</i>	There is limited temporal variation in the visual field due to motion or small viewpoint changes. This allows for incremental processing of animation or video by using results obtained from previous time steps.
<i>Frame</i>	There are similarities in successive frames of an animation or video sequence when the time difference is small. This is a result of temporal, spatial, and object coherence and improves efficiency, compression and storage of videos.
<i>Scan-line</i>	There are limited differences between successive scan lines in visible-surface determination, which usually allow for incremental processing of scan lines.
<i>Span</i>	There are constant properties within segments of a scan line that are often used by scan-line methods such as polygon filling or hidden surface removal.
<i>Ray</i>	There are similarities between rays with similar origin and direction in <i>ray tracing</i> methods. This is a result of spatial and object coherence and it is used to reduce the computational cost of ray tracing.
<i>Radiosity</i>	Various types of coherence (including frustrum, intervisibility, patch-to-patch and surface coherence) are used to speed up radiosity methods.

Table 5.2: Various forms of coherence in computer graphics.

that separates order from chaos. In this theory, it is also assumed to be the principle and bias that underlies all learning and makes it possible. The highly non-uniform distribution of functionality in nature (which is necessary

for learning as we discussed in section 5.5) is largely an effect of coherence. Moreover, if coherence is indeed so important, it is necessary that we provide a formal definition of it and methods to measure it.

A first and rather straightforward attempt to define coherence is to use variance as defined in statistics. However, such a definition breaks down immediately because sample variance is permutation invariant. If we permute the data points of a data set or the pixels of an image, the variance of the data set and the image remain the same but the result would most likely be incomprehensible and look like noise as shown in figure 5.9. The original

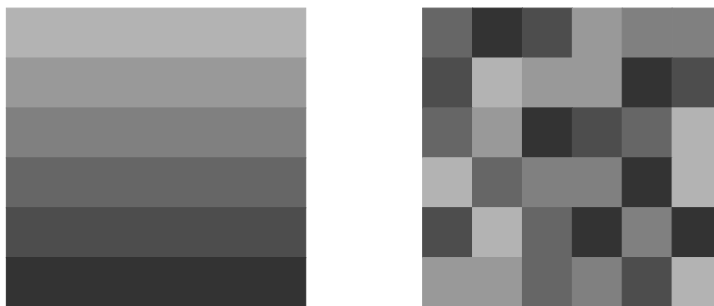


Figure 5.9: An image of gray scale color gradient on the left and a random permutation of pixels blocks in it.

image is a square that contains a vertical gradient of six shades of grey. It is then divided into 36 square pixel blocks (six on each side) and the blocks are randomly permuted to generate the second image. The mean and variance of the two images are the same (since the pixels have only changed position but not value) but the second image is clearly harder to comprehend and remember. Furthermore, if we increase the number of blocks to n^2 for some $n > 6$, the disorder in the second image will only increase.

In general, the order and subdivision of the elements of natural representations are important. Different natural entities and their representations will most likely imply or admit different order and divisions; and this suggests that ordering or division should be treated as a parameter in the definition of coherence. Standard statistical measures were designed to measure uncertainty, not to describe nature, order, division, or coherence. Uncertainty is part of learning and nature of course but it is not the only or most significant part.

A second and equally straightforward attempt to define coherence comes from calculus. If coherence is about gradual change perhaps we might represent it with continuity (or smoothness). However, this definition fails just as quickly. A single minor discontinuity in an otherwise smooth and bound function will make it discontinuous and thus incoherent. (We have tentatively assumed incoherence to indicate reduced or diminished coherence.) More importantly, continuity cannot distinguish functions based on their rate of change. For example, a triply exponential function is just as continuous as a linear or a

constant one! Yet, the number of continuous real functions is dramatically smaller than the number of arbitrary real functions as shown in equation (5.15). This is a poor ‘trade’. It reduces the cardinality from 2^c to c without achieving any broad distinction on rates of change. In short, continuity is a coarse property that imposes relatively severe constraints on functions without sufficiently discriminating among them. It is neither sufficient to define coherence nor a necessary consequence of any such definition.

The above considerations lead us to differential calculus. It is not a coincidence that Gottfried W. Leibniz, who developed differential calculus, was among the first of the post-antiquity scientists and philosophers who understood the fundamental role and form of change in nature and the importance of measuring it in a formal manner in science. Differential calculus was invented for that purpose. Thus our third attempt at a definition for coherence is the the derivative. Perhaps, we do not even need to define coherence as a separate or additional concept. After all, derivatives capture change in an accurate and formal manner.

There are problems with such a definition that arise from the nature and limitations of derivatives. Not all functions are continuous or differentiable. There are discrete domains too. A single discontinuity would make a function non-continuous and of course non-differentiable (at the discontinuity). Noise and inaccuracy can profoundly affect derivatives. And the actual value of the derivative at a point x is not as important if the derivative changes significantly around x . Nature is not as much concerned about individual points or infinitesimal changes but rather about intervals of time and change that are common or important for a given environment, its entities, and their interactions.

On the other hand, we can resolve or reduce most of the above problems. We can handle discontinuities using Dirac delta functions (as we explained in section 2.5). We can use finite differences instead of the derivative for non-continuous or discrete functions. If we use proper estimators, we may be able to handle or reduce sensitivity to noise. Finally, if we constrain change within specific and finite localities, we can avoid issues with variability and trivial or infinitesimal quantities. These ideas lead us to a more promising definition. Let us introduce some related notation.

Definition 5.10. If $(D, \|\cdot\|)$ is a metric space ($\|x, y\|$ indicates the distance between $x, y \in D$), S is a tuple or array containing n elements of D , and f is a function in D , then we define

$$fS = (f(s)/s \in S) = (f(S_i)/i \in \mathbb{N}_n)$$

and, for all $x \in D$, we define

$$\|S, x\| = (\|s, x\|/s \in S) = (\|S_i, x\|/i \in \mathbb{N}_n).$$

Similarly, when $\|\cdot\|$ is a norm, we define

$$\|S - x\| = (\|s - x\|/s \in S) = (\|S_i - x\|/i \in \mathbb{N}_n).$$

For any set X , a function $\alpha: X \rightarrow X^*$ is called an *association function* in X . Intuitively, α maps each element $x \in X$ to a tuple of elements of X and thus ‘associates’ x with those elements. Such functions can be used to map each point of a domain to a locality around the point. Note that association is different from the concept of *neighborhood* in topology. A neighborhood of a point x must include an open set containing x and it is not ordered in any specific way. An association $\alpha(x)$, on the other hand, is an arbitrary tuple in X^* and it is thus finite and ordered.

Now we can introduce a generalization of the concept of finite difference defined for functions in metric spaces and calculated using metrics.

Definition 5.11. Assume two metric spaces $(D, \|\cdot\|)$ and $(R, |\cdot|)$, a ratio estimator ρ , and an association function $\alpha: D \rightarrow D^*$. Then the *finite metric difference* or simply *metric difference* of a function $f: D \rightarrow R$ at $x \in D$ with respect to estimator ρ and association α is defined as

$$\Delta_{\rho\alpha}f(x) = \rho\left(\frac{|f\alpha(x), f(x)|}{\|\alpha(x), x\|}\right). \quad (5.39)$$

When α is not a single association function but an infinite sequence of such functions, that is when $\alpha = \langle \alpha_n : D \rightarrow D^*/n \in \mathbb{N} \rangle$, then the finite metric difference is defined as the limit when n tends to infinity

$$\Delta_{\rho\alpha}f(x) = \lim_{n \rightarrow \infty} \Delta_{\rho\alpha_n}f(x). \quad (5.40)$$

For example, if ρ is the ratio-of-means estimator R_2 (from section 5.3), equation (5.39) becomes

$$\Delta_{R_2\alpha}f(x) = R_2\left(\frac{|f\alpha(x), f(x)|}{\|\alpha(x), x\|}\right) = \frac{\sum_{z \in \alpha(x)} |f(z), f(x)|}{\sum_{z \in \alpha(x)} \|z, x\|}.$$

Furthermore, if $D = \mathbb{R}^2$, the metrics $|\cdot|$ and $\|\cdot\|$ are *induced** by p -norms, and $\alpha(x, y) = ((x + 1, y), (x, y + 1))$, the above equation becomes

$$\Delta_{R_2\alpha}f(x, y) = \frac{\|f(x + 1, y) - f(x, y)\|_p + \|f(x, y + 1) - f(x, y)\|_p}{2}.$$

The finite metric difference of a function is effectively a generalization of its finite difference if we ignore the fact that distance is used and thus the direction or sign of the difference is ignored. As a result, the finite metric

*A metric $\|\cdot\|$ is said to be *induced* by the norm $\|\cdot\|$ of normed vector space $(V, \|\cdot\|)$ iff $\|x, y\| = \|x - y\|$ for all $x, y \in V$.

difference is also a generalization of the magnitude of the derivative of f , if the derivative exists. This is demonstrated by the the following theorem.

Theorem 5.8. *We assume that $(D, \|\cdot\|)$ is a normed vector space with dimensionality δ , p -norm as its norm (for some $p > 0$), and the induced metric (that is $\|x\| = (\sum_i |x_i|^p)^{1/p}$ and $\|x, z\| = \|x - z\| = (\sum_i |x_i - z_i|^p)^{1/p}$, for all $x, z \in D$). For $i \in \{1, \dots, \delta\}$, we define $e_i = (0, \dots, 0, 1, 0, \dots, 0)$, where the non-zero element is at the i -th position, and assume that the sequences of association functions $\alpha_i = \langle \alpha_{i,n} : D \rightarrow D^* / n \in \mathbb{N} \rangle$ are such that $\alpha_{i,n}(x) = (x + e_i/n)$ for all $x \in D$. (We also assume that there exists $n_0 \in \mathbb{N}$ such that $x + e_i/n \in D$ for all $n > n_0$.) Then, for all differentiable functions $f : D \rightarrow \mathbb{R}$ and for all $k > 0$ and $i_1, \dots, i_k \in \{1, \dots, \delta\}$, it is*

$$\Delta_{\rho\alpha_{i_1}} \Delta_{\rho\alpha_{i_2}} \dots \Delta_{\rho\alpha_{i_k}} f(x) = \left| \frac{\partial^k f(x)}{\partial x_{i_1} \partial x_{i_2} \dots \partial x_{i_k}} \right| \quad (5.41)$$

if the derivatives exist and the estimator ρ satisfies the property $\rho(\frac{s}{t}) = \frac{s_0}{t_0}$ when s and t are tuples with only one element.

Proof. For any differentiable function $g : D \rightarrow \mathbb{R}$ and $j \in \mathbb{N}_\delta$, it is

$$\left| \frac{\partial |g(x)|}{\partial x_j} \right| = \left| \frac{\partial g(x)}{\partial x_j} \right|. \quad (5.42)$$

Indeed, if $g(x) \neq 0$,

$$\left| \frac{\partial |g(x)|}{\partial x_j} \right| = \left| \frac{g(x)}{|g(x)|} \frac{\partial g(x)}{\partial x_j} \right| = \left| \frac{\partial g(x)}{\partial x_j} \right|.$$

Similarly, if $g(x) = 0$,

$$\left| \frac{\partial |g(x)|}{\partial x_j} \right| = \left| \lim_{h \rightarrow 0} \frac{|g(x + he_j)| - |g(x)|}{h \|e_j\|} \right| = \left| \lim_{h \rightarrow 0} \frac{g(x + he_j)}{h \|e_j\|} \right| = \left| \frac{\partial g(x)}{\partial x_j} \right|.$$

Furthermore, we can combine equations (5.39) and (5.40) and the assumed property of the estimator ρ to obtain

$$\Delta_{\rho\alpha_j} g(x) = \lim_{n \rightarrow \infty} \frac{|g(x + e_j/n) - g(x)|}{\|e_j/n\|} = \left| \frac{\partial g(x)}{\partial x_j} \right|. \quad (5.43)$$

We can now prove equation (5.41) by induction on k . For $k = 1$ we simply apply equation (5.43) for $g = f$ and $j = i_1$. For the induction step, we would like to prove equation (5.41) for k assuming it to be true for $k - 1 > 0$. This is straightforward,

$$\Delta_{\rho\alpha_{i_1}} \dots \Delta_{\rho\alpha_{i_k}} f(x) = \left| \frac{\partial \Delta_{\rho\alpha_{i_2}} \dots \Delta_{\rho\alpha_{i_k}} f(x)}{\partial x_{i_1}} \right| \quad \begin{array}{l} \text{(by (5.43) for} \\ g = \Delta_{\rho\alpha_{i_2}} \dots \Delta_{\rho\alpha_{i_k}} f) \end{array}$$

$$\begin{aligned}
&= \left| \frac{\partial \left| \frac{\partial^{k-1} f(x)}{\partial x_{i_2} \dots \partial x_{i_k}} \right|}{\partial x_{i_1}} \right| && \text{(by the induction hypothesis)} \\
&= \left| \frac{\partial^k f(x)}{\partial x_{i_1} \partial x_{i_2} \dots \partial x_{i_k}} \right| && \text{(by (5.42) for } g = \frac{\partial^{k-1} f}{\partial x_{i_2} \dots \partial x_{i_k}} \text{).}
\end{aligned}$$

□

Intuitively, the association function α defines a locality around each point x and the estimator ρ defines a method that is used to calculate the metric difference $\Delta_{\rho\alpha} f(x)$ of f at x . Those two elements can reduce the effect of noise on the result. The association function is also defined to be a finite tuple at each point x for efficiency (and for enabling numerical estimations). The estimator will often be R_2 or R_1 ; but the choice of a custom estimator allows for greater emphasis on certain parts of the associated locality. For example, if one of the input dimensions is time, more recent data, relative to a data point x , may be weighted more when computing the metric difference at x .

The association locality also addresses an underlying issue of granularity. For example, we can consider the two images of figure 5.9 again. If we compute differences at the level of individual pixels with a narrow locality and the image is large enough, metric differences will be diminished.

We can make the example more concrete by assuming the image to be 300 x 300 pixels and the association function to be $\alpha(x, y) = ((x - 1, y), (x, y - 1))$. Then each of the 36 pixel blocks will be 50 x 50 pixels in size and only the pixels on the left and bottom side of each block will potentially have different metric difference in the permuted image. Given that there are 99 such border pixels (50 for the left side plus 50 for the bottom side minus one for the lower left pixel that is counted twice), the percentage of pixels with the same metric difference in the two images is at least $1 - 99/50^2 = 96.04\%$. On the other hand, if we decrease granularity and treat each 50 x 50 block as an atomic entity, averaging its pixel values, we can compute the metric difference for each block. In that case, the percentage of blocks with the same metric difference across the two images is only $3/36 = 8.33\%$ (the two leftmost blocks in the bottom row and the leftmost block in the top row are the only ones that happen to have the same metric difference).

In other words, coherence is affected by granularity because it is closely related to perception. As we explained earlier in this section, the concept of ‘jump’ in Leibniz’s principle is relative to the observer. It also depends on the environment and the relative scales that are common or important to it. As a result, any viable definition for coherence must include a mechanism for such relativity. In our case, the mechanism is the combination of the association function and the ratio estimator.

The above definition seems to address a number of issues. It can capture the rate of growth and change for a broad range of functions (for most

functions, there is an appropriate locality and ratio estimator). However, there are two remaining issues that must be addressed.

Coherence is not only about the relative (or absolute) magnitude of change but also about the form or 'shape' of change. The latter is particularly important for learning. For example, linear (or affine) change is probably the easiest to measure and predict. It has received great attention in statistical and early machine learning literature because it can be studied and analyzed theoretically. There can be considerably greater certainty in a learning system when change is linear or nearly linear. As a result, definitions of coherence must measure some form of distance or proximity to linearity.

For the same reason, we would like coherence to be invariant to affine transformations. This would additionally protect the system from false inferences about change in coherence when a function is merely multiplied by a constant.

Metric difference is invariant under translation when the metrics are induced by a norm. Indeed, the metric difference of the function $g(x) = f(x) + c$, where c is a constant, is then

$$\Delta_{\rho\alpha}g(x) = \rho\left(\frac{|g\alpha(x) - g(x)|}{\|\alpha(x), x\|}\right) = \rho\left(\frac{|f\alpha(x) - f(x)|}{\|\alpha(x), x\|}\right) = \Delta_{\rho\alpha}f(x).$$

However, scale invariance is a more demanding condition and the combination of the two conditions is too restrictive. Let us understand the reason.

A ratio estimator cannot possibly be scale invariant. It should be linear or close to linear ($\rho(\lambda y/x) \approx \lambda \rho(y/x)$). As a result, the only component that could potentially make metric difference invariant to affine transformations is the metric $|\cdot|$. Although affine invariant metrics are possible (e.g. the *affine invariant Riemannian metric*), the condition effectively implies angular metrics for inner product spaces.* We will examine such metrics for neural structure later in this theory. However, we do not consider them an appealing option for defining coherence. Given that coherence must measure change, it cannot possibly ignore magnitude and focus on angular metrics only. Moreover, such metrics are not appropriate for measuring and detecting proximity to linearity either.

As a result, we opted for a definition of coherence that uses multiple orders in a fashion similar to the orders of derivatives. For that purpose, we should first define orders of metric differences.

*If $(D, \langle \cdot, \cdot \rangle)$ is an inner product space over $\mathbb{R}$ with a metric $|\cdot|$ that satisfies $|\lambda x + t, \lambda y + t| = |\lambda x, \lambda y|$ for all $x, y, t \in D$ and all $\lambda \in \mathbb{R}$, then using $\lambda = 1, t = -x$, we obtain $|0, y - x| = |x, y|$; and using $\lambda = 1, t = -y$, we obtain $|x - y, 0| = |x, y|$. The symmetry of the metric $|\cdot|$ then implies that there is an even function f such that

$$(\forall x, y \in D) \quad |x, y| = f(x - y).$$

Consequently, $f(\lambda x) = |\lambda x, 0| = |x, 0| = f(x)$, for all $\lambda \in \mathbb{R}$ and $x \in D$. In other words, the magnitude of the vector x is irrelevant to the value $f(x)$; only the direction of x matters.

Definition 5.12. Assume two metric spaces $(D, \|\cdot\|)$ and $(R, |\cdot|)$, a ratio estimator ρ , and an association function $\alpha: D \rightarrow D^*$. Then the *finite metric difference of order k* or simply *metric difference of order k* of a function $f: D \rightarrow R$ with respect to estimator ρ and association α is defined recursively as

$$\begin{aligned}\Delta_{\rho\alpha}^1 f &= \Delta_{\rho\alpha} f; \\ \Delta_{\rho\alpha}^{k+1} f &= \Delta_{\rho\alpha} \Delta_{\rho\alpha}^k f.\end{aligned}\tag{5.44}$$

Furhtermore, if P is a distribution over D and S is a measurable subset of D then the *finite metric difference of order k* of f in S with respect to ρ and α is the expectation of the metric difference over S according to distribution P

$$\Delta_{\rho\alpha}^k f(S) = E_{X/S} \Delta_{\rho\alpha}^k f(x) = \frac{\int_S \Delta_{\rho\alpha}^k f(x) dP(x)}{\int_S dP(x)},\tag{5.45}$$

where X is a random variable distributed in D according to P .

Now, we can define coherence (and incoherence).

Definition 5.13. Assume two metric spaces $(D, \|\cdot\|)$ and $(R, |\cdot|)$, a distribution P over D , a ratio estimator ρ , and an association function $\alpha: D \rightarrow D^*$. Then the *first-order incoherence* or simply *incoherence* of a function $f: D \rightarrow R$ in a measurable subset $S \subseteq D$ with respect to estimator ρ and association α is the metric difference of f in S with respect to ρ and α and according to P ; and the *first-order coherence* (or simply *coherence*) of f in S is the inverse of its incoherence.

$$\mathbf{C}_{\rho\alpha} f(S) = \frac{1}{\mathbf{I}_{\rho\alpha} f(S)} = \frac{1}{\Delta_{\rho\alpha} f(S)}.\tag{5.46}$$

For $k \geq 2$, the *coherence of order k* of f in S with respect to ρ and α is again the inverse of its *incoherence of order k* and it is defined as follows

$$\mathbf{C}_{\rho\alpha}^k f(S) = \frac{1}{\mathbf{I}_{\rho\alpha}^k f(S)} = \frac{\Delta_{\rho\alpha}^{k-1} f(S)}{\Delta_{\rho\alpha}^k f(S)}.\tag{5.47}$$

The two formulas can be combined into one, if we define $\Delta_{\rho\alpha}^0 = 1$ (for all ρ and α). Using split notation, we can furthermore write the equation in a more concise form. Hence, for $k \geq 1$,

$$\mathbf{C} = \frac{1}{\mathbf{I}} = \frac{\Delta^-}{\Delta} \cdot_{\rho\alpha}^k (f(S)).\tag{5.48}$$

In general, we assume that the operator Δ has higher precedence than multiplication. Thus $\Delta f g$ stands for $(\Delta f)g$.

When the domain D or the distribution P are discrete, with possibly non-zero probability P_i at point x_i , for $i \in \mathbb{N}$, the integrals in equation (5.45) simply turn into summations. Similarly, the coherence on a single point x

implicitly assumes a discrete distribution and it is

$$\mathbf{C} = \frac{1}{\mathbf{I}} = \frac{\Delta^-}{\Delta} \cdot_{\rho\alpha}^k(f(x)).$$

Higher orders of coherence (and incoherence) have a desirable property. As the following theorem demonstrates, they are invariant to affine transformations in normed vector spaces for linear ratio estimators.

Theorem 5.9. *Assume two normed vector spaces $(D, \|\cdot\|)$ and $(R, |\cdot|)$ over a field F and the metrics that are induced by their norms; that is $\|z, x\| = \|z - x\|$ for all $x, z \in D$ and $|r, s| = |r - s|$ for all $r, s \in R$. Furthermore, assume a distribution P over D , a ratio estimator ρ , and an association function $\alpha: D \rightarrow D^*$. If the estimator is linear in the sense that it satisfies the property $\rho(\lambda y/x) = \lambda \rho(y/x)$, for all $\lambda \in F$ and $x, y \in D^*$, then the coherence and incoherence of order $k \geq 2$ of any function $f: D \rightarrow R$ in a measurable subset $S \subseteq D$ with respect to estimator ρ and association α is invariant to affine transformations.*

Proof. Let function $g: D \rightarrow R$ be an affine transformation of f . Then there must be constants $\lambda \in F$ and $t \in D$ such that $(\forall x \in D) g(x) = \lambda f(x) + t$. We would like to prove that

$$\mathbf{C}_{\rho\alpha}^k g = \mathbf{C}_{\rho\alpha}^k f$$

for all $k \geq 2$ (which implies that $\mathbf{I}_{\rho\alpha}^k g = \mathbf{I}_{\rho\alpha}^k f$).

However, for all $x \in D$ and $k \geq 2$

$$\left. \begin{aligned} \Delta g &= \rho \left(\frac{|g\alpha - g|}{\|\alpha - x\|} \right) = \rho \left(\frac{|\lambda(f\alpha - f)|}{\|\alpha - x\|} \right) = |\lambda| \Delta f \\ \Delta^k g &= \Delta^{k-1} \Delta g = \Delta^{k-1} |\lambda| \Delta f = |\lambda| \Delta^k f \end{aligned} \right\} : (x) | \Delta_{\rho\alpha}. \quad (5.49)$$

Therefore, for all measurable $S \subseteq D$

$$\mathbf{C}_{\rho\alpha}^k g(S) = \frac{\int_S \Delta^{k-1} g dP}{\int_S \Delta^k g dP} = \frac{\int_S |\lambda| \Delta^{k-1} f dP}{\int_S |\lambda| \Delta^k f dP} = \mathbf{C}_{\rho\alpha}^k f(S) \quad : (x) | \Delta_{\rho\alpha}, \mathbf{C}_{\rho\alpha}.$$

□

For most learning applications, it is reasonable to assume normed vector spaces and thus metrics induced by norms. It is also reasonable to assume linearity for the estimator ρ and unit estimate for identical samples ($\rho(\lambda y/x) = \lambda \rho(y/x)$ and $\rho(x/x) = 1$). For example, both R_1 and R_2 satisfy those properties. Under such assumptions, if f is affine, that is if there are constants λ and t such that $f(x) = \lambda x + t$ for all $x \in D$ then $\Delta_{\rho\alpha} f(x) = \rho \left(\frac{(|\lambda| \|\alpha(x) - x\|)}{\|\alpha(x) - x\|} \right) = |\lambda|$ and $\Delta_{\rho\alpha}^2 f(x) = \Delta_{\rho\alpha} |\lambda| = 0$. The latter implies that $\mathbf{C}_{\rho\alpha}^2 f(x)$ is infinite. Consequently, proximity to an affine form is

indicated by large values in second-order coherence (or, equivalently, small values in second-order incoherence).

For the remaining of this volume, we will exclusively use first-order coherence because it is a natural starting point and allows us to develop the theory in a progressive and evolutionary manner. We will start using second-order coherence in volume II. Although it is possible that applications of higher-order coherence may emerge in the future, we have not yet found such a use case in this theory. Intuitively, first-order incoherence measures the magnitude of change while second-order coherence measures the 'shape' of change using affine forms as a baseline and measuring proximity to them. This seems to be sufficient for all the applications we have considered, including learning methods, neural structure and functionality, plasticity, and evolution.

Moreover, in this theory, we postulate that learning is a non-monotonic transition in coherence. Learners start in a relatively incoherent state. Their coherence starts increasing until it reaches a maximum. Then it starts decreasing again and learners slowly descend into incoherence. In that last stage, various undesirable phenomena such as overfitting or reduced sensitivity to locality take place.

Good generalization is assumed to occur when a learner matches the coherence of its environment (or the coherence of an unknown underlying function when the learner is dedicated to that function). This is almost a tautology for various situations. If a learner approximates not only the values of a function f in certain data points within a hyperrectangle but also the magnitude and the 'shape' of the change of f across the hyperrectangle, the learner will most likely be an accurate approximation of f and generalize well in the hyperrectangle.

On the other hand, the coherence of an environment or underlying function are usually unknown and hard to compute. For many classification problems, estimating the coherence is a subjective problem and at least as hard as the classification problem. For example, it is as hard as it is subjective to determine the exact point where a deformed object, such as an apple, does not look like an apple anymore. Learners cannot assume or expect to be given a target coherence. Instead they should explore the space of coherence in an unsupervised manner and infer their state relative to the coherence of their environment. The target coherence is largely irrelevant. All that matters is the journey and a sense of direction: is the learner still moving toward the target, as it does at the beginning of its training, or is it moving away from it, as it does during overfitting?

In the next chapter, we will define operators that allow us to modify the coherence of a learner and thus guide or affect its learning. A large part of Mathesis is a transition from the current empirical state-of-the-art in machine learning into a paradigm of coherence and coherent entities – such as coherent functionality, coherent learning methods, coherent structure, coherent plasticity and growth, and coherent evolution. Coherence also marks

the beginning of a transition to an evolutionary process and approach that is assumed to be necessary for intelligence.

5.9 Notes on Coherence

If the estimator ρ and the association function α are as defined in theorem 5.8 and P is a discrete probability distribution then, for all $x \in D$ and any differentiable function F , it is

$$\mathbf{I}_{\rho\alpha}^2 F(x) = \left| \frac{\nabla_{xx} F(x)}{\nabla_x F(x)} \right| = |\nabla_x \ln |\nabla_x F(x)||.$$

Furthermore, if F is a distribution with probability density function f , its second-order incoherence becomes

$$\mathbf{I}_{\rho\alpha}^2 F = |\nabla_x \ln |f||.$$

In other words, when we use the above association function, the second-order incoherence of a distribution F is the magnitude of the derivative of the information $\ln |f|$ of F . This derivative measures change of information within the distribution. When such information changes frequently and quickly or abruptly across the domain of the distribution, the distribution is less coherent. Consequently, we sometimes use the alternative terms *information coherence* and *information incoherence* for $\mathbf{C}_{\rho\alpha}^2$ and $\mathbf{I}_{\rho\alpha}^2$, respectively.

Information coherence is important for generalization purposes. If a learning system is trained to approximate an unknown distribution F and the learned distribution G is such that $|\mathbf{C}_{\rho\alpha}^2 G / \mathbf{C}_{\rho\alpha}^2 F - 1|$ is modest or large, G would most likely not generalize well.

An alternative definition for second-order coherence can be derived using curvature. We call this the *geometric coherence/incoherence*. Given that curvature is translation invariant but not scale invariant, we should normalize such a definition by dividing by the volume of the minimum bounding hyperrectangle. Although there are some interpretations of this latter definition that relate to physics, we did not find any other arguments in support of it. Hence we opted for the information-related definition. However, geometric coherence is a potential research area within the theory and learning in general.

The relationship between coherence and generalization is slowly and independently emerging in neural network literature. More limited terms, such as *smoothness* or *robustness* are often used. For example, in a study that was published in 2021, it was shown that *smooth interpolation* requires d times more parameters than mere interpolation, where d is the dimensionality of the data. The result was then used to explain why deep neural networks with large number of learning parameters represent a class of learning models that generalize better.

This is clearly an important result and a theoretical step in understanding and establishing the relationship between coherence and generalization. However, and despite its mathematical complexity and sophistication, the approach makes some simplifying assumptions. The most significant simplification is the use of a *Lipschitz constant** as a measure of smoothness or robustness. Hence, the approach operates on a single number instead of a set of numbers; it uses a reduction and a bound on smoothness rather than coherence and its actual distribution. There are additional issues such as noise in the data and the effective dimensionality of the data manifold which is often less than d . The authors also assume *isoperimetry* of the data distribution, which implies a tight concentration around their mean, or a mixture of such distributions. Those may be necessary simplifications to make the approach mathematically tractable. Working with the distribution of coherence and arbitrary noise and data distributions is much harder.

As we explained in section 2.6, neural networks represent a universal learning paradigm where complexity is essentially unbounded but provability is hard or intractable. It is not clear whether the relationship between coherence and generalization can be accurately quantified. However, we expect more results like the above in the coming years and a convergence to some approximation of the true relationship. The above study is only a first step.

Finally, we should note that many machine learning methods are effectively methods that manipulate coherence. For example, both metric regularization and batch normalization are methods that change the coherence of the learning parameters of a network (they tend to increase it). Metric regularization is a coarser method that treats all parameters as equal while batch normalization is a more localized spatio-temporal method that modifies coherence across layers and training mini-batches. Of course, those are not the only methods for manipulating coherence, they are simple and straightforward, and they more coarse than granular.

5.10 Locality as Coherence

In section 1.5, we discussed two different types of locality. The principle of locality limits influence to the immediate surroundings of an object while locality of reference implies spatial and temporal proximity in memory access during computations. Those are two distinct forms of locality but they are special cases of coherence.

In general, measures of proximity and similarity as well as measures of distance and diversity that derive from metrics can be represented through

*If $(X, \|\cdot\|)$ and $(Y, |\cdot|)$ are two metric spaces, a function $f: X \rightarrow Y$ is called *Lipschitz continuous* iff $(\exists K \in \mathbb{R}_+)(\forall x_1, x_2 \in X) |f(x_1) - f(x_2)| \leq K\|x_1 - x_2\|$. Any such constant K is called a *Lipschitz constant* for the function f . The smallest such constant is often called the *best Lipschitz constant*.

metric differences (with appropriate ratio estimators and association functions). They thus measure incoherence over a domain.

In this section, we focus on locality in the connectivity of a neural network. We interchangeably refer to it as *synaptic locality* or *synaptic coherence*, although, later in the theory, we will expand the definition of synaptic coherence to include the overall coherence in synaptic placement, plasticity, growth, functionality, and learning.

In subsection 1.5.1, we defined layer locality ratios as a measure of locality (see equation (1.11)). However, such ratios measure locality within a layer only and they are invariant to unit permutations in the layer. In general, we would like to have a synaptic locality definition that applies to any set of units and takes into account the ‘distance’ between units or neurons that synapse with each other.

Definition 5.14. We assume a neural network $N = (G, F, Y, S)$ (as in definition 4.2) where $G = (V, E)$. We also assume a metric $|\cdot|$ over V (or, equivalently, over the set of indices Y_*) and define $\|\cdot\|$ to be the *identity metric* in V , that is

$$\|u, v\| = (u \neq v) = \begin{cases} 0 & \text{if } u = v \\ 1 & \text{else} \end{cases}$$

for all $u, v \in V$. We furthermore assume that ρ is either the R_1 or R_2 ratio estimator and we define an association function α so that*

$$\alpha(v) = \text{tuple}(G^{-1}(v)) = (u \in V / (u, v) \in E)$$

for all $v \in V$. Then the *synaptic incoherence* of unit v in the network N is

$$\mathbf{I}_N v = \Delta_{\rho\alpha} v = \rho \left(\frac{|\alpha(v), v|}{\|\alpha(v), v\|} \right) = \frac{\sum_{u \in G^{-1}(v)} |u, v|}{|G^{-1}(v)|} = E_{u \in G^{-1}(v)} |u, v| \quad (5.50)$$

while the *synaptic incoherence* of a set of units $U \subseteq V$ with respect to a distribution P over V is†

$$\mathbf{I}_N U = \sum_{v \in U} \mathbf{I}_N v P(v/v \in U) = \frac{\sum_{v \in U} \Delta_{\rho\alpha} v P(v)}{\sum_{v \in U} P(v)} \quad (5.51)$$

The *synaptic coherence* or *synaptic locality* of a unit v or set of units U in the network N is simply the inverse of their synaptic incoherence

$$\mathbf{C}_N U = \frac{1}{\mathbf{I}_N U}. \quad (5.52)$$

*We use the notation $\text{tuple}(A)$ to indicate the tuple that contains the elements of the set A in lexicographic order.

†We assume that the synaptic incoherence operator $\mathbf{I}_N$ has greater precedence than multiplication. Thus $\mathbf{I}_N v P(v/v \in U)$ stands for $(\mathbf{I}_N v) P(v/v \in U)$.

The distribution P is usually assumed to be uniform. In that case equation (5.52) becomes

$$\mathbf{C}_{NU} = \frac{1}{\mathbf{I}_{NU}} = \frac{|U|}{\sum_{u \in U} \Delta_{\rho\alpha} u}, \quad (5.53)$$

where $|U|$ indicates the size of the set U .

We should note that the above definition depends on the metric $|\cdot|$ in V . For example, a *topological sort* of the acyclic graph G is any bijection $r: V \rightarrow \mathbb{N}_{|V|}$ such that

$$(\forall u, v \in V) \quad (u, v) \in E \Rightarrow r(u) < r(v)$$

The existence of the *rank* function r is implied by the fact that G is acyclic (the same way that layerings of G are implied by the same fact). The rank function can then induce a metric in V as follows

$$(\forall u, v \in V) \quad |u, v| = |r(u) - r(v)|,$$

where $|x|$ indicates the absolute value of $x \in \mathbb{Z}$.

The above metric treats V as a one-dimensional set. This results in coarse distances that are also affected by the size of the layers even when the network contains successive connections only.

We can create a better metric if we represent the vertices by the corresponding indices of Y_* . In that case, each vertex $v \in V$ is represented by the index (l_v, i_v) , where L_{l_v} is the layer of N that contains v and i_v is the index of v in L_{l_v} . For example, if each layer L_k of N is a one-dimensional array then the indices i_v form an initial segment of $\mathbb{N}$; or, more precisely, $\{i_v / v \in L_k\} = \mathbb{N}_{|L_k|}$ for all $k \in \mathbb{N}$. We can then define a metric $|\cdot|$ in V such that

$$|u, v| = |(l_u, i_u), (l_v, i_v)| = \left| \frac{i_u}{|L_{l_u}|} - \frac{i_v}{|L_{l_v}|} \right| + \epsilon \sum_{k \in l_{u,v}} |L_k| \quad (5.54)$$

for all $u, v \in V$. We assume that ϵ is a constant to weight intermediate layers and $l_{u,v}$ denotes the set of the indices of the intermediate layers between the units u and v (that is $l_{u,v} = \{i \in \mathbb{N} / \min(l_u, l_v) < i < \max(l_u, l_v)\}$).

We can understand the above formula better if we distinguish between successive connections and cross-connections.

In the former case, there are no intermediate layers ($l_{u,v}$ is an empty set), only the first term of equation (5.54) applies, and it is simply the absolute value of the difference of the relative positions of the units u and v in their layers. Of course, such a difference depends on the size of the layers but it is meant to capture synaptic topologies where fractions of the units of layer L_{l_u} connect to similar fractions of the units of layer L_{l_v} . Alternatively, we could use the absolute positions and thus the quantity $|i_u - i_v|$, which is less consistent with the topology of the connections but also less dependent on the size of the layers.

For cross-connections, there are similar issues but there are also intermediate layers that are bypassed by those connections. Given that the Kolmogorov complexity of a layer is probably linear with the number of units in the layer (assuming no shared parameters), it is reasonable that the ‘functional’ distance between the units u and v of a cross-connection (u, v) should contain a term that is linear with $|L_k|$, for each bypassed layer L_k . This is the rationale for the summation in equation (5.54).

Of course, if the layers are multidimensional arrays, the first term of equation (5.54) should be amended to account for the dimensionality of the arrays – possibly with a metric upon the relative multidimensional positions of the corresponding units in those arrays. Physical distance or wire-length can also be used, if we assume a two-dimensional or three-dimensional layout of the units or neurons on an actual surface or physical space. Moreover, locality metrics should also capture the synaptic topology of the network and the amount of computation in various partitions of the network.

The above are potential properties of locality metrics that indicate the central role of structure in neural systems but also relate to structural adaptation through neural plasticity, growth, and evolution.

In general, coherence will only increase in complexity and sophistication as we develop the theory further. Synaptic coherence consists of multiple components, processes, and metrics and each of them contains multiple elements that often have various alternative forms. We will examine such elements in chapters 7 and 8 and elaborate upon the rest in volumes II and III.

5.11 Historical and Bibliographical Remarks

Modular arithmetic was introduced in 1801 by C. F. Gauss in his book *Disquisitiones Arithmeticae* [122].

Set theory effectively started with a seminal paper by G. Cantor in 1874 which studied infinity and introduced cardinals [44]. B. Bolzano had earlier studied infinity too but his approach was considerably more elementary.

Measure theory was formulated and progressively developed in the second half of the 19th century and early 20th century by a number of mathematicians, including E. Borel, H. Lebesgue, J. Radon, and M. Frechet. It played an important role in the axiomatization of probability theory by A. Kolmogorov, which was published in 1933 [191]. In 1950, P. Halmos published a textbook on measure theory [144].

ZF stands for the *Zermelo-Fraenkel axiomatic set theory* while ZFC is ZF with the axiom of choice included [176]. The non-provability of the existence of measurable cardinals and thus of a measure or probability space with cardinality $2^{\aleph_1}$ in ZFC was partly proved in 1930 and completed in 1967 [347, 321].

Information about estimators can be found in standard probability or

statistics textbooks [262, 182]. A more detailed treatment of *point estimators* appears in [210].

Ratio estimators have a long history. Their first known use was on the ratio of population over registered births by J. Graunt in 1662. K. Pearson was aware of their bias and suggested caution in 1897 [267]. Unbiased estimators with common sampling schemes appeared in [148, 131, 233] and with specialized sampling in [204, 234, 307, 276, 247]. Almost unbiased ratio estimators are discussed in [245] along with a proof of the relationship between the bias of R_1 and R_2 . Our proof in section 5.3 follows [349].

A discussion about generalization in the context of psychology and neuroscience can be found in [128].

Different versions of the “no free lunch theorem” first appeared in [392, 394]. However, they all make similarly erroneous assumptions of uniformity as theorem 5.6. They either assume a uniform distribution for the underlying function f or they uniformly average over all possible priors $P(f)$ for the function f . Neither of those conditions is true in real applications or environments. On the contrary, the corresponding distributions are strictly non-uniform with a majority of zero or trivial probabilities and impossible priors in virtually all actual cases.

In 2005, the authors of the above theorems argued that all “optimization algorithms are equivalent when their performance is averaged across all possible problems” [395]. Of course, this is a mathematical tautology but it is rather meaningless. Averaging across all possible problems represents a statistical fallacy. The vast majority of those ‘problems’ are superficial, extreme, or irrelevant. Most of them will never emerge in real environments or applications. There is obviously a threshold (of energy or complexity) that produces a bijection between functions across the threshold (e.g. we can use the number of logical gates to partition Boolean functions into two such sets). Yet, probability density is not evenly split between such sets. Above certain energy or complexity levels, probabilities will be zero in real environments (because the corresponding functions are impossible or too costly to construct). The same applies to Kolmogorov complexity and virtually all aspects of learning.

A similar erroneous interpretation of the theorem has appeared in various publications including some essential textbooks (e.g. [99]). In current machine learning literature, it is a widely accepted fallacy that the no free lunch theorem implies that no learning system is ‘superior’ to any other learning system, where ‘superiority’ is indicated by better generalization. This is ironic because the aforementioned theorem actually implies the opposite, given the strictly non-uniform distribution of functionality, relation, and their priors. Different environments with such divergent statistics can range from entire planets or their ecosystems down to small localities within a neural network. Even if we limit ourselves to narrow learning systems, the premise of uniformity does not apply and the presumed equivalence of learning systems is still false. In

other words, even narrow learning systems are not equivalent.

Nevertheless, as explained in section 5.5, theorem 5.6 is important. The same applies to the various versions of the no free lunch theorem. Those theorems emphasize the importance of the environment (and eventually coherence and locality) for learning. They imply that learning must take the environment into account in some form. This is a fundamental constraint that has been widely neglected in machine learning literature.

Hadamard’s description of well-posed problems appeared in [142].

The true origin of Aristotle-Ockham razor is debatable but William of Ockham historically gets credit largely because of the writings of W. Hamilton in 1852. L. Froidmont (1587–1653) had earlier used the term “*novacula occami*” without referencing Aristotle. William of Ockham is believed to have said “*entia non sunt mutlipicanda praeter necessitate*” (entities should not be multiplied without necessity) or “*pluralitas non est ponenda sine necessitate*” (plurality should not be posited without necessity).

The principle had previously been stated by Aristotle c. 350 BCE. Related statements appear in *Physics*, book I, chapter 6 (“for the more limited, if adequate, is always preferable”), book VIII, chapter 6, (“for if the consequences are the same it is always better to assume the more limited antecedent”) as well as in *Posterior Analytics* (“we may assume the superiority of the demonstration which derives from fewer postulates or hypotheses”) [228, p. 150], [13, 14, 48, 115]. A few centuries later, the principle was also stated by Ptolemy in book III, section 1 (“we consider it a good principle to explain the phenomena by the simplest hypothesis possible”) [115, ch. 9, p. 241].

Ockham’s assumed wording cannot be found in any of his existing works but he is believed to have made extensive use of the principle [65].

Minimum description length was introduced in 1978 [282].

Dropout was proposed in [158] while dropconnect was introduced a year later in [358]. Spatial dropout appeared in [341], stochastic depth in [163], and autodropout in [269].

Optimal brain damage and optimal brain surgeon were introduced in [66] and [151], respectively.

Metric regularization is a general process that has many forms (depending on the metric $\|\cdot\|$ and the function ρ of equation (5.32)). A number of such forms have been independently discovered and used in various fields of science. However, the term metric regularization is introduced in this theory. To the best of our knowledge, it has not been used before.

One of the earliest cases of metric regularization is *Tikhonov regularization*. It goes back to the 1940s and it was applied to integral equations [338, 339]. In this case, the metric term is of the form $\|Mx\|_2^2$ where M is a chosen matrix called *Tikhonov matrix* and x is the vector of the underlying variables. A generalized version uses different weighting for each dimension in x and a translation from a reference point x_0 . Thus the metric term is of the form $(z^T M z)^2$, where $z = x - x_0$.

Batch normalization was introduced in 2015 [174].

The Weierstrass approximation theorem was proved by by Karl Weierstrass in 1885 [360]. Runge’s phenomenon was discovered in 1901 by C. Runge [293]. For any given n , the Chebyshev nodes are the roots of the *Chebyshev polynomial* of the first kind of degree n . Chebyshev polynomials were named after Russian mathematician P. Chebyshev who first described them in 1854 [50].

The first concise and formal description of the method of least squares was published by A.-M. Legendre in 1805 [209]. Polynomial regression (using least squares) appeared ten years later in 1815 [123].

The idea of splines originated in shipbuilding and aircraft industries; the term appeared in the British aircraft industry during World War II [21]. However, the first reference to splines in the context of smooth and piecewise polynomial approximation seems to be [304].

The concept of the convex hull first appeared in a letter from Isaac Newton to Henry Oldenburg in 1676 [251]. (The *Newton polygon* in the letter is a special case of the *lower convex hull* of a set of points on the plane.) However, the term convex hull did not appear in the literature or become standard until the first half of the 20th century [27, 203, 85].

Estimates of the volumes of the convex hull of random points appear in [42].

The MNIST data set was released in 1999 [208]. It is a small but popular and standard data set for classification problems in computer vision. (MNIST stands for “Modified National Institute of Standards and Technology”.) The Gilbert–Johnson–Keerthi distance algorithm was published in [124]. An estimate of the resolution of the human eye appears in [59].

The Vandemonde matrix and determinant were named after the French mathematician A.-T. Vandermonde, who was the first to write about basic properties of determinants. However, the specific determinant that bears his name did not appear in any of his publications.

The idea of locality-sensitive hashing seems to have originated in data mining [214].

Nouveaux Essais Sur L’Entendement Humain (*New Essays on Human Understanding*) is a book by Gottfried W. Leibniz that contains his famed apothegm that “nature does not make jumps” [212, IV-16] (“la nature ne fait jamais des sauts”). The book was written in a dialogue form in 1704 (like many philosophical works at time, following a tradition that traces back to Plato) and it is a chapter-by-chapter rebuttal of John Locke’s *An Essay Concerning Human Understanding* [220]. However, Leibniz allegedly withheld publication for 60 years because of Locke’s death in 1704. The book was finally published in 1765.

The Latin version of the above assertion (“natura non facit saltus”) comes from C. Linnaeus’ *Philosophia Botanica* [217]. Charles Darwin introduced natural selection in his *Origin of Species*, which was published in 1859 [70].

John Locke’s arguments against innate ideas also appear in [220]. His quote that “nothing is in the mind without being first in the senses, except for the mind itself” is from the same publication [220, Book II, Ch. 1] (“nihil est in intellectu quod non fuerit in sensu excipe: nisi ipse intellectus”).

The term speciation was coined by O. Cook in 1906 [63]. Punctuated equilibrium was introduced by N. Eldredge and S. Gould in 1972 [135]. The relationship between phyletic gradualism and punctuated equilibria as instances of gradualism (interpreted through rates of change) was presented by R. Dawkins in his book *The Blind Watchmaker* in 1986 [73, ch. 9]. This view was later corroborated by S. J. Gould [134].

Universal grammar (a.k.a. *linguistic nativism*) is a theory that postulates a genetic component in the faculty of the language, and it is usually credited to Noam Chomsky [52, 53]. Arguments and evidence for early language development to facilitate tool-making appear in [162, 242, 213]. Estimates about the origin of the human genus appear in [100].

The history of coherence in physics is related to the history of interference. The concept originated from studies of Thomas Young’s *double slit experiment* and led to the *theory of coherence* in optics [391].

Coherence has a more limited presence in signal processing and it is always defined in terms of spectral densities [315, 22].

A survey of coherence types and applications in computer graphics appears in [137]. A theoretical result that connects the number of learning parameters and smooth interpolation under isoperimetry appears in [40].

Mathematics takes us still further from what is human into the region of absolute necessity, to which not only the actual world, but every possible world, must conform.

— Bertrand Russell, *The Study of Mathematics*, 1907.

6.1	Of Correlations and Convolutions	207
6.2	Finite Recursion	211
6.3	Recursion versus Correlation and Convolution	214
6.4	Correlation and Convolution Examples	221
6.5	Recursion and Coherence	228
6.6	Transformation Bias	231
6.7	Historical and Bibliographical Remarks	232

6

Operators for Coherence

6.1 Of Correlations and Convolutions

In signal processing and functional or Fourier analysis, *cross-correlation* and *convolution* are two mathematical operations that map a pair of functions to a third function that measures similarity between the two functions. Formally, those operations are defined as follows.

Definition 6.1. Assume that D is a measurable subset of $\mathbb{R}^\delta$ (or $\mathbb{C}^\delta$) for some $\delta \in \mathbb{N}_+$ and that $\bar{z}$ denotes the *complex conjugate* of $z \in \mathbb{C}$. Then the *cross-correlation* of two complex functions $f: D \rightarrow \mathbb{C}$ and $g: D \rightarrow \mathbb{C}$ over D is

$$(\forall x \in D) \quad (f \star g)(x) = \int_D \overline{f(s)} g(x+s) ds, \quad (6.1)$$

and their *convolution* over D is

$$(\forall x \in D) \quad (f * g)(x) = \int_D f(s) g(x-s) ds. \quad (6.2)$$

When D is a discrete subset, the integrals reduce to summations in the above formulas. Hence, discrete correlation has the form

$$(f \star g)(x) = \sum_{s \in D} \overline{f(s)} g(x+s) \quad (6.3)$$

and discrete convolution is

$$(f * g)(x) = \sum_{s \in D} f(s) g(x-s). \quad (6.4)$$

The two operations are considered equivalent in the sense that one can represent the other. Indeed, if we assume D to be symmetric around 0 (that is $x \in D$ iff $-x \in D$) and define $h(x) = \overline{f(-x)}$ then, for all $x \in D$,

$$(h \star g)(x) = \int_D f(-s) g(x+s) ds = (f * g)(x).$$

However, as we shall see later in this chapter, this equivalence breaks down when we restrict the integration or the summation to a subset of D or when storage, computational complexity, or coherence are taken into account.

In general, cross-correlation and convolution operate on the entire domain D . This is appropriate in signal processing and Fourier analysis where the functions f and g usually represent signals. In learning, however, systems learn and evolve over time and there is always a beginning of their existence. Time is thus an inevitable component of their domain D ; and the future is unknown, at any given time. Consequently, we cannot apply the above definition and compute correlations and convolutions for the entirety of time. We must truncate the integrals and the summations to the current time t . Given that learning also occurs in discrete learning events, we treat time as discrete.

For the remaining of this volume, we assume real or complex functions with domain $D = \mathbb{N}$. Zero represents the beginning of time for the system. The functions f and g are now sequences $\langle f_t/t \in \mathbb{N} \rangle$ and $\langle g_t/t \in \mathbb{N} \rangle$. We also redefine correlation and convolution to be relative to the current time t .

Definition 6.2. The *cross-correlation* (or simply *correlation*) of two sequences $\langle f_t/t \in \mathbb{N} \rangle$ and $\langle g_t/t \in \mathbb{N} \rangle$ is defined as

$$(\forall t \in \mathbb{N}) \quad (f \star g)_t = \sum_{k=0}^t f_k g_k. \quad (6.5)$$

Their *convolution* is similarly defined as

$$(\forall t \in \mathbb{N}) \quad (f * g)_t = \sum_{k=0}^t f_k g_{t-k}. \quad (6.6)$$

The *normalized correlation* and *normalized convolution* with respect to a function $h: \mathbb{C}^* \rightarrow \mathbb{C}$ is

$$(\forall t \in \mathbb{N}) \quad (f \diamond_h g)_t = \frac{(f \diamond g)_t}{h(f_0, \dots, f_t)}, \quad (6.7)$$

where $\diamond$ is either $\star$ or $*$ for correlation or convolution, respectively.

Note that we do not follow the standard definition of cross-correlation. We do not conjugate the first function f in the correlation formulas of definition 6.2. Although there are arguments in favor of conjugation that emerge from the view of cross-correlation as a dot-product in $\mathbb{C}^n$ and the fact that, for every $x, y \in \mathbb{C}$, $\bar{x}y$ is maximized when the phase of x is the same as the phase of y , we have opted for a different convention because it simplifies some relationships between correlation and convolution (e.g. for geometric progressions). Furthermore, our convention does not make a difference in most cases because the first function in our correlations is usually a real function.

In either case, the convention is not essential. Applying our definition to functions $\bar{f}$ and g is equivalent to applying the standard definition to f and g .

A second notational observation is that the plain correlation and convolution are special cases of normalized correlation and convolution, respectively, with respect to the constant function $h = 1$. For that reason we will often drop the term “normalized” and refer to those operations simply as correlation or convolution when the function h is implied.

When the function h is p -norm, we would only use the subscript p to indicate it. Hence

$$(f \diamond_p g)_t = \frac{(f \diamond g)_t}{(\sum_{k=0}^t |f_k|^p)^{1/p}}. \quad (6.8)$$

We will also use the terms *cumulative correlation* and *cumulative convolution* and the following notation when h is summation

$$(f \diamond_\Sigma g)_t = \frac{(f \diamond g)_t}{\sum_{k=0}^t f_k}. \quad (6.9)$$

In this volume, the emphasis is largely on cumulative correlations and convolutions.

In most cases, we will treat f as a sequence of coefficients that assign different weight to the elements of the sequence g ; and we will often use symbol c for it.

Sequence notation is used frequently in this chapter and the rest of the theory as well. The convention is that when a sequence appears in an expression, the expression applies to all the elements of the sequence. For example, if $\langle u_t/t \in \mathbb{N} \rangle$ and $\langle v_t/t \in \mathbb{N} \rangle$ are two sequences, $u \neq 0$ stands for $(\forall t \in \mathbb{N}) u_t \neq 0$. Similarly $u + v$ and u^2 stand for the sequences $\langle u_t + v_t/t \in \mathbb{N} \rangle$ and $\langle u_t^2/t \in \mathbb{N} \rangle$, respectively.

An important property of correlation and convolution operations on sequences is the relative weight they place on two successive elements g_k and g_{k+1} of the sequence g in $(c \diamond_h g)_t$.

Definition 6.3. The *growth rate* of a correlation transformation $c \star_h g$ is the sequence

$$\rho_c = \left\langle \frac{c_{k+1}}{c_k} / k \in \mathbb{N} \right\rangle.$$

Similarly, the *growth rate* of a convolution $c *_{h_1} g$ is the sequence

$$r_c = 1/\rho_c = \left\langle \frac{c_k}{c_{k+1}} / k \in \mathbb{N} \right\rangle.$$

If $c_k = 0$ or $c_{k+1} = 0$, $\rho_{c,k}$ or $r_{c,k}$ is undefined.

Correlation and convolution aging is the condition that $\rho_c \geq 1$ and $r_c \geq 1$, respectively. (Intuitively, an aging transformation respects the ‘age’ of the elements of a sequence and monotonically discounts early ones.)

We should note that when $c_t \geq 0$ for all $t \geq 0$, there is a probabilistic interpretation for cumulative correlations and convolutions. For all $t \in \mathbb{N}$ and $k \in \mathbb{N}_{t+1}$, we can define $p_{t,k} = c_k / \sum_{j=0}^t c_j$. Then $p_{t,k} \geq 0$, $\sum_{k=0}^t p_{t,k} = 1$, and $(c \star_{\Sigma} g)_t = \sum_{k=0}^t p_{t,k} g_k$. Therefore, the cumulative correlation is the sequence of the expectation of g over the distributions $p_0, \dots, p_t$. The same, of course, applies to the cumulative convolution (but the expectation is for the inverted sequence, from t to 0). The growth rate of the operation then measures the increase of successive probabilities.

Unfortunately, the probabilistic representation requires to store t parameters for each t , resulting in $\Theta(t^2)$ space complexity while the direct representation of the sequence c requires $\Theta(t)$ storage and less indexing. The probabilistic representation helps us understand the meaning of the coefficients c_k but the direct representation is a better computational and analytic option. (It can also apply when some of the coefficients are negative but we will not explore such cases in this volume.)

In general, convolution operations are symmetric. For example $c * g = g * c$. This is not the case with their normalized versions for sequences. The reason for the asymmetry is that the first sequence is treated as a resource for weighting the values of the second sequence. In the cumulative cases, the operation also retains the mean (but not the variance) of the sequence g .

On the other hand, if we also divide the definition of normalized correlation by $h(g_0, \dots, g_t)$ to make the formula more symmetric, we effectively divide the dot product of c and g with $h(c)$ and $h(g)$. If c and g are real sequences and h is 2-norm, the result would be the cosine of the angle between the vectors $[c_k/k \leq t]$ and $[g_k/k \leq t]$ for each t . Consequently, if we wish to create a formula that depends on the magnitude of g and it is monotonic or linear with the sequence g , our best option is an asymmetric definition. This is precisely our choice in this volume.

For simplicity and more concise formulas, we will also use the following notation.

Definition 6.4. Given a field R , a sequence $c \in R^{\mathbb{N}}$, and a function $h: R^* \rightarrow R$ such that $h(c_0, \dots, c_t) \neq 0$ for all $t \in \mathbb{N}$, we define the sequence operators $C_{ch}: R^{\mathbb{N}} \rightarrow R^{\mathbb{N}}$ and $K_{ch}: R^{\mathbb{N}} \rightarrow R^{\mathbb{N}}$ such that

$$\begin{aligned} C_{ch}g &= c \star_h g \\ K_{ch}g &= c * _h g \end{aligned} \tag{6.10}$$

for all $g \in R^{\mathbb{N}}$. Cumulative correlations and convolutions are denoted as $C_{c\Sigma}$ and $K_{c\Sigma}$ or simply C_c and K_c , respectively.

In the rest of this chapter, we will examine the relationship of the above operators, their relationship with recursion, and their effects on coherence.

6.2 Finite Recursion

The word recursion derives from the late Latin *recursio* (a backward run, return), from *recurrere* (to run back, return). In mathematics, it refers to a definition of a function that references previous values of that function. Such a definition is mathematically valid on its own but it is often associated with induction because its uniqueness derives from induction or transfinite induction for non-countable sets. Let us now be more formal.

In set theory, *transfinite recursion* uniquely defines a function $f: D \rightarrow R$ from a partially ordered set* $(D, <)$ to any set R using a function G over arbitrary sets such that f satisfies the property

$$f(z) = G(\{(x, f(x)) \in D \times R / x < z\})$$

for all $z \in D$.

We can see now that when the domain of f is not countable, transfinite recursion translates to transfinite induction. However, given the enormity of uncountable sets, we may constrain ourselves to countable sets only.

If we restrict the domain to be the set of natural numbers $\mathbb{N}$, we obtain *finite recursion* or simply *recursion*. In that case, the set $\{(x, f(x)) / x < z\}$ can be represented by the tuple $(f(x) / x < z) = (f(0), \dots, f(z-1))$ and G is effectively a function over tuples or finite sequences. (In computer science terminology, G can be seen as a function with a *variable number of arguments*.)

Definition 6.5. Assume a set R and a function $G: R^* \rightarrow R$. Then the unique sequence f that has the property[†]

$$(\forall t \in \mathbb{N}) \quad f_t = G(f_0, \dots, f_{t-1}) \quad (6.11)$$

is said to be defined through *finite recursion* (or *recursion*). If there is $k \in \mathbb{N}_+$ such that[‡]

$$(\forall t \in \mathbb{N}) \quad f_t = G(f_{\max(0, t-k)}, \dots, f_{t-1}, t) \quad (6.12)$$

then f is defined through *k-order recursion*.

For example the Fibonacci sequence is defined through a second-order

*A pair $(D, <)$ is a partially ordered set iff D is a set and $<$ is a binary relation on D that satisfies the following two properties:

- a. $(\forall x \in D) \quad x \not< x$
- b. $(\forall x, y, z \in D) \quad x < y \text{ and } y < z \Rightarrow x < z$

[†]The uniqueness of f can easily be established by induction. $f_0 = G(())$ is unique because G is a function; and if f_s is unique for $s < t$ so is the tuple $(f_0, \dots, f_{t-1})$ and thus $f_t = G(f_0, \dots, f_{t-1})$.

[‡]In this case, t must be included in the tuple because it can no longer be determined by the length of the tuple when $t > k$.

recursion

$$\begin{aligned} f_0 &= 0; \\ f_1 &= 1; \\ f_t &= f_{t-1} + f_{t-2}, \quad \text{for } t \geq 2. \end{aligned}$$

In this case, the generating function G is defined so that $G(0) = f_0 = 0$, $G(0, 1) = f_1 = 1$, and $G(a, b, t) = a + b$ for all $a, b, t \in \mathbb{N}$. (As a result, $f_t = G(f_{t-2}, f_{t-1}, t) = f_{t-2} + f_{t-1}$, for $t > 1$.)

Of course, k -order recursion can be transformed into first-order recursion if we operate on tuples of size k . For example, if we define $F_t = (f_t, f_{t+1})$ for $t \in \mathbb{N}$, it would then be $F = \mathcal{R}_1^G$, where $G(0) = (f_0, f_1) = (0, 1)$ and $G((x, y), t) = (y, x + y)$. (Indeed, $(f_t, f_{t+1}) = F_t = G(F_{t-1}, t) = G((f_{t-1}, f_t), t) = (f_t, f_{t-1} + f_t)$.) Similarly, when $a = \mathcal{R}_k^H g$, we can define $A_t = (a_t, \dots, a_{t+k-1})$. It would then be $A = \mathcal{R}_1^G g$, where $G(x, 0) = x$ and $G(x, t) = (x_1, \dots, x_{k-1}, H(x_0, \dots, x_{k-1}, g_t, t))$ for $t > 0$.

However, the above transformation is only a notational trick. It hides the order of the recursion into the length of its tuples. Consequently, in this theory, we would define order based on sequences of numbers only. Complex numbers can be seen as pairs or tuples of size two, but that is the only exception we allow. In other words, recursion order will be defined over real or complex numbers and not interchangeably.

A common situation in learning and coherence is to transform a given sequence $\langle g_t/t \in \mathbb{N} \rangle$ so that the new sequence has some specific properties or different coherence.

Definition 6.6. Given a field R , a *recursive transformation* of a sequence $g \in R^{\mathbb{N}}$ is any sequence $a \in R^{\mathbb{N}}$ that is defined as

$$\begin{aligned} (\forall t \in \mathbb{N}) \quad a_t &= H(a_0, \dots, a_{t-1}, g_t, t) \\ &= H_t(a_0, \dots, a_{t-1}, g_t), \end{aligned} \tag{6.13}$$

where $H: R^* \rightarrow R$ is a generator function* that defines the recursion and H_t is simply H indexed by t as shown below

$$(\forall x) \quad H_t(x) = H(x, t).$$

We then write $a = \mathcal{R}^H g$, where $\mathcal{R}^H: R^{\mathbb{N}} \rightarrow R^{\mathbb{N}}$ is a sequence-to-sequence function such that

$$(\mathcal{R}^H g)_t = H_t((\mathcal{R}^H g)_0, \dots, (\mathcal{R}^H g)_{t-1}, g_t)$$

for all $g \in R^{\mathbb{N}}$ and $t \in \mathbb{N}$.

Similarly, a sequence $a \in R^{\mathbb{N}}$ is a *k-order recursive transformation* of g iff

*In this case, the generator G is defined as $G(a_0 \dots, a_{t-1}, t) = H(a_0, \dots, a_{t-1}, g_t, t)$.

it is defined through k -order recursion as shown below

$$(\forall t \in \mathbb{N}) \quad a_t = H_t(a_{\max(0, t-k)}, \dots, a_{t-1}, g_t). \quad (6.14)$$

In that case, we write $a = \mathcal{R}_k^H g$ and define the function $\mathcal{R}_k^H$ such that

$$(\mathcal{R}_k^H g)_t = H_t((\mathcal{R}_k^H g)_{\max(0, t-k)}, \dots, (\mathcal{R}_k^H g)_{t-1}, g_t)$$

for all $g \in R^{\mathbb{N}}$ and $t \in \mathbb{N}$.

A special case is when H is affine and particularly when $k = 1$. We define it below using split notation.

Definition 6.7. The *first-order affine transformation* of a sequence $g \in R^{\mathbb{N}}$ with respect to sequences $\lambda, \mu \in R^{\mathbb{N}}$ is the unique sequence $a \in R^{\mathbb{N}}$ that satisfies

$$\begin{aligned} a &= g & : t = 0; \\ a &= \lambda g + \mu a_- & : t > 0. \end{aligned} \quad (6.15)$$

In that case we write $a = \mathcal{R}_1^{\lambda\mu} g$ and define

$$\left. \begin{aligned} H(x, y) &= \lambda y + \mu x \\ ((\mathcal{R}_1^{\lambda\mu} g)) &= H((\mathcal{R}_1^{\lambda\mu} g)_-, g) = \lambda g + \mu((\mathcal{R}_1^{\lambda\mu} g))_- \end{aligned} \right\} : t | x, y$$

for all $g \in R^{\mathbb{N}}$ and $t \in \mathbb{N}$.

A k -order affine transformation of g is similarly indicated as

$$a = \mathcal{R}_k^{\lambda\mu} g$$

(now λ is a sequence and μ is a tuple containing k sequences) and defined as

$$\begin{aligned} a &= g & : t = 0; \\ a &= \lambda g + \sum_{i=1}^{\llbracket \min(k, t) \rrbracket} ((\mu_{i-1})) a_{-i} & : t > 0. \end{aligned} \quad (6.16)$$

Intuitively, λ_t is the ‘learning’ or currency rate that determines how much a_t is affected by the current value g_t , while $(\mu_i)_t$ is a ‘memory’ or ‘momentum’ rate that determines how much a_t is affected by its previous value a_{t-i-1} for $i < k$.

Note that the values of the sequences λ and μ at zero are not used in the formulas. Consequently, we can assume that $\lambda_0 = 1$ and $(\mu_i)_0 = 0$ to simplify the above formulas. The result is shown below:

$$\left. \begin{aligned} ((\mathcal{R}_1^{\lambda\mu} g)) &= \lambda g + \mu((\mathcal{R}_1^{\lambda\mu} g))_- \\ ((\mathcal{R}_k^{\lambda\mu} g)) &= \lambda g + \sum_{i=1}^{\llbracket \min(k, t) \rrbracket} ((\mu_{i-1}))((\mathcal{R}_k^{\lambda\mu} g))_{-i} \end{aligned} \right\} : t. \quad (6.17)$$

for all $g \in R^{\mathbb{N}}$ and $t \in \mathbb{N}$.

k -order recursion usually requires us to store the k previous values of the transformation or the sequence in order to compute the next one. Hence, first-order recursion is a unique form of recursion that minimizes storage requirements. This is particularly important when we apply such transformations at large scale (for example at every connection of a neural network.)

In the next section we will examine the relationship between first-order recursion and correlation and convolution.

6.3 Recursion versus Correlation and Convolution

In general, recursion is a powerful mathematical and computing mechanism. For example, the μ -recursive functions are equivalent to *Turing machines* (they are precisely the functions that can be computed by Turing machines; we will define them formally and discuss this topic in more detail in the next chapter). If the *Church-Turing thesis*^{*} is correct, the above implies that all computable functions can be represented as μ -recursive functions. Consequently, any transformation of a sequence can be expressed as a μ -recursive function. The same is not true for correlation and convolution transformations. Such transformations represent a proper subset of finite recursion transformations, which are a proper subset of μ -recursive or computable transformations.

There is a clear relationship between correlation and finite recursion. First-order affine transformations are equivalent to correlations as shown in theorem 6.1 below. We will first prove a lemma that transforms affine first-order recursion to summation.

Lemma 6.1. *If $\beta \neq 0 : t > 0$ and $\alpha = \beta\alpha_- + \gamma : t > 0$ then*

$$(\forall t \in \mathbb{N}) \quad \alpha_t = \frac{1}{s_t} \left(\alpha_0 + \sum_{i=1}^t s_i \gamma_i \right),$$

where $s_0 = 1$ and $s_t = 1 / \prod_{j=1}^t \beta_j$ for $t > 0$.

Proof. Given that $s_- = s\beta : i > 0$, we can use the sequence s to transform the recursion into a telescopic formula:

$$s\alpha = s_- \alpha_- + s\gamma : i > 0$$

Summing up the above equation for $i = 1, 2, \dots, t$ and eliminating common terms, we obtain

$$s_t \alpha_t = \alpha_0 + \sum_{i=1}^t s_i \gamma_i$$

^{*}The Church-Turing thesis states that any function of natural numbers can be calculated by a formal method iff it can be computed by a Turing machine.

□

We can now proceed with our theorem.

Theorem 6.1. *For all $c \in \mathbb{C}^{\mathbb{N}}$ and $h \in \mathbb{C}^{\mathbb{C}^*}$, if $h(c_0, \dots, c_t) \neq 0$ for all $t \in \mathbb{N}$ then*

$$(\exists \lambda, \mu \in \mathbb{C}^{\mathbb{N}}) \quad \mathcal{R}_1^{\lambda\mu} = C_{ch}. \quad (6.18)$$

Conversely, for all $\lambda \in \mathbb{C}^{\mathbb{N}}$ and $\mu \in (\mathbb{C} - \{0\})^{\mathbb{N}}$

$$(\exists c \in \mathbb{C}^{\mathbb{N}})(\exists h \in \mathbb{C}^{\mathbb{C}^*}) \quad \mathcal{R}_1^{\lambda\mu} = C_{ch}. \quad (6.19)$$

Proof. For the first part of the theorem, we assume an arbitrary sequence $c \in \mathbb{C}^{\mathbb{N}}$ and an arbitrary function $h: \mathbb{C}^* \rightarrow \mathbb{C}$ such that $h_t = h(c_0, \dots, c_t) \neq 0$ for all $t \in \mathbb{N}$. We would like to construct sequences $\lambda, \mu \in \mathbb{C}^{\mathbb{N}}$ from the sequence c and the function h such that equation (6.18) is true. Given a sequence $g \in \mathbb{C}^{\mathbb{N}}$, we define $a = C_{ch}g$. Hence,

$$a = (c \star_h g) = \frac{1}{h} \left[\sum_{k=0}^t c_k g_k \right] \quad : t \in \mathbb{N}. \quad (6.20)$$

We would like to prove that $a = \mathcal{R}_1^{\lambda\mu}g$. However, equation (6.20) implies that,

$$ah - a_-h_- = cg \quad : t > 0.$$

Rearranging terms, we obtain

$$a = \frac{c}{h}g + \frac{h_-}{h}a_- \quad : t > 0.$$

Consequently, if we define

$$\left. \begin{aligned} \lambda &= \frac{c}{h} \\ \mu &= \frac{h_-}{h} \end{aligned} \right\} \quad : t > 0 \quad (6.21)$$

we get

$$a = \lambda g + \mu a_- = (\mathcal{R}_1^{\lambda\mu}g) \quad : t > 0.$$

The above completes the first part of the theorem.

For the second part, we assume sequences $\lambda \in \mathbb{C}^{\mathbb{N}}$ and $\mu \in (\mathbb{C} - \{0\})^{\mathbb{N}}$. We would like to construct a sequence $c \in \mathbb{C}^{\mathbb{N}}$ and a function $h: \mathbb{C}^* \rightarrow \mathbb{C}$ such that equation (6.19) is true. Given any sequence $g \in \mathbb{C}^{\mathbb{N}}$, we can define $b = \mathcal{R}_1^{\lambda\mu}g$. We would like to prove that $b = C_{ch}g$. However, by definition, it is

$$b = \lambda g + \mu b_- \quad : t.$$

Applying lemma 6.1 to the above equation (for $\alpha = b$, $\beta = \mu$ and $\gamma = \lambda g$),

we obtain that

$$b_t = \frac{1}{s_t} \left(b_0 + \sum_{k=1}^t s_k \lambda_k g_k \right), \quad (6.22)$$

where

$$s_t = \begin{cases} 1 & \text{if } t = 0, \\ \frac{1}{\mu_1 \cdots \mu_t} & \text{if } t > 0. \end{cases}$$

Now if we pick $c_0 \in \mathbb{C} - \{0\}$ and define

$$\begin{aligned} c_t &= c_0 s_t \lambda_t \quad (\text{for } t > 0) \\ h(c_0, \dots, c_t) &= h_t = c_0 s_t \end{aligned} \quad (6.23)$$

equation (6.22) becomes

$$b_t = \frac{1}{h_t} \left(c_0 g_0 + \sum_{k=1}^t c_k g_k \right) = (C_{ch}g)_t,$$

which is well-defined and implies that $b = C_{ch}g$ since $h_t = c_0 s_t \neq 0$ for all $t \in \mathbb{N}$. $\square$

For cumulative correlations, the condition that $h(c_0, \dots, c_t) \neq 0$ becomes $\sum_{k=0}^t c_k \neq 0$, which implies $c_0 \neq 0$ if we pick $t = 0$. Given that $c \star_{\Sigma} g = (\gamma c) \star_{\Sigma} g$ for all constants $\gamma \in \mathbb{C} - \{0\}$ and all sequences $g \in \mathbb{C}^{\mathbb{N}}$, we can pick $\gamma = 1/c_0$ or equivalently we can assume that $c_0 = 1$. Equation (6.23) then becomes

$$\left. \begin{aligned} c &= s\lambda \\ h &= s \end{aligned} \right\} : t$$

while equation (6.21) becomes

$$\left. \begin{aligned} \lambda &= \frac{c}{h} \\ \mu &= 1 - \lambda \end{aligned} \right\} : t > 0.$$

Combining the above equations, we obtain the following formula that connects cumulative correlations and first-order affine transformations

$$\lambda = 1 - \mu = c/s \quad : t \in \mathbb{N}, \quad (6.24)$$

where $s_t = h_t = \sum_{k=0}^t c_k = 1 / \prod_{i=1}^t \mu_i$ and $s_0 = c_0 = 1$.

Theorem 6.1 is important because it allows us to compute correlations through first-order recursion and thus usually with $\Theta(1)$ space and time complexity (at each step t). Of course, the sequences λ and μ have their own memory requirements and computational complexity; but they are often expressed in closed form that results in $\Theta(1)$ space and time complexity. Hence, the overall complexity is usually $\Theta(1)$.

Convolution, on the other hand, is not as efficient. Its relationship with recursion is more complicated, too. Convolution does not admit a first-order recursive form except in special cases with constant or geometric progression coefficients. The following theorem demonstrates the incompatibility between first-order recursion and convolution in a more formal manner.

Theorem 6.2. *For all sequences $c \in \mathbb{C}^{\mathbb{N}}$ and functions $h, H \in \mathbb{C}^{\mathbb{C}^*}$ it is*

$$K_{ch} = \mathcal{R}_1^H \quad (6.25)$$

iff there exists $\gamma \in \mathbb{C}$ such that

$$\mathcal{R}_1^H = \mathcal{R}_1^{\lambda\mu} \quad \text{and} \quad (\forall t \in \mathbb{N}) \quad c_t = c_0 \gamma^t, \quad (6.26)$$

where $\lambda_t = c_0/h_t$, $\mu_t = \gamma h_{t-1}/h_t$, and $h_t = h(c_0, \dots, c_t)$.

Proof. We first assume (6.26) and an arbitrary sequence $g \in \mathbb{C}^{\mathbb{N}}$. Then

$$\begin{aligned} \langle\langle K_{ch}g \rangle\rangle &= \frac{1}{h} \sum_{k=0}^t c_{t-k} g_k \\ &= \frac{c_0 g}{h} + \frac{\gamma}{h} \sum_{k=0}^{t-1} c_0 \gamma^{t-1-k} g_k \\ &= \lambda g + \frac{\gamma}{h} h_- \langle\langle K_{ch}g \rangle\rangle_- \\ &= \lambda g + \mu \langle\langle K_{ch}g \rangle\rangle_- \\ &= \langle\langle \mathcal{R}_1^{\lambda\mu} g \rangle\rangle \quad :t|t, \gamma \end{aligned}$$

The last step derives from the fact that equation $a_t = \lambda_t g_t + \mu_t a_{t-1}$ defines a unique sequence a when a_0 is given (in this case it is $a_0 = g_0$).

Now, the above derivation implies $K_{ch} = \mathcal{R}_1^{\lambda\mu}$ because it applies to all sequences $g \in \mathbb{C}^{\mathbb{N}}$. The latter implies $K_{ch} = \mathcal{R}_1^H$ per our assumption of equation (6.26).

For the opposite direction, we assume equation (6.25). We would like to prove (6.26). However, if $c = 0$ the convolution is always 0 and equation (6.26) is trivially true ($c_0 = 0$ and $\mathcal{R}_1^H = \mathcal{R}_1^{\lambda\mu} = 0$). Similarly, if $c_0 \neq 0$ and $c_t = 0$ for all $t > 0$, $K_{ch}g = c_0 g$ and equation (6.26) is again trivially true ($\gamma = 0$). Thus we assume that

$$(\exists i \in \mathbb{N}_+) \quad c_i \neq 0. \quad (6.27)$$

Now, if $g \in \mathbb{C}^{\mathbb{N}}$ is an arbitrary sequence and we define $a = \mathcal{R}_1^H g$ then (6.25) implies

$$a = H(a_-, g) = \frac{1}{h} \sum_{k=0}^t c_{t-k} g_k \quad :t|t. \quad (6.28)$$

We shall first show that $c \neq 0$. For that purpose we define a sequence g so that $g_k = 0$ for all $k \in \mathbb{N}_t - \{0\}$. Equation (6.28) then implies

$$c_0 g_t + c_t g_0 = h_t a_t = h_t H_t \left(\frac{c_{t-1} g_0}{h_{t-1}}, g_t \right) \quad (6.29)$$

for all $t \in \mathbb{N}_+$ and $g_0, g_t \in \mathbb{C}$. Now, if $c_{t-1} = 0$ and we pick $g_t = 0$, the above equation becomes $c_t g_0 = h_t H_t(0, 0)$, which cannot be true for all $g_0 \in \mathbb{C}$ except if $c_t = H_t(0, 0) = 0$. Thus we proved that

$$(\forall t \in \mathbb{N}_+) \quad c_{t-1} = 0 \Rightarrow c_t = 0. \quad (6.30)$$

Conversely, if $c_t = 0$ and $c_{t-1} \neq 0$ and we pick $g_0 = h_{t-1}x/c_{t-1}$ and $g_t = y$, where x and y are arbitrary complex numbers, equation (6.29) becomes

$$c_0 y = h_t H_t(x, y) \quad (6.31)$$

for all $x, y \in \mathbb{C}$. Combining the above equation with equation (6.28), we obtain $c_0 g_t = \sum_{k=0}^t c_{t-k} g_k$ for all sequences $g \in \mathbb{C}^{\mathbb{N}}$. This is obviously a contradiction, given equation (6.27) (if we pick $i > 0$ such that $c_i \neq 0$ and also pick $t > i$, $g_{t-i} \neq 0$, and $g_k = 0$ for $k \in \mathbb{N} - \{t-i\}$, the above equation implies $0 = c_0 g_t = c_i g_{t-i} \neq 0$). As a result, we have proved that

$$(\forall t \in \mathbb{N}_+) \quad c_t = 0 \Rightarrow c_{t-1} = 0. \quad (6.32)$$

Combining equations (6.30) and (6.32) we obtain the following equivalence

$$(\forall t \in \mathbb{N}_+) \quad c_t = 0 \Leftrightarrow c_{t-1} = 0. \quad (6.33)$$

In other words, either $c = 0$ or $c \neq 0$. Given that equation (6.27) contradicts the assumption $c = 0$, it must be $c \neq 0$.

Now, we shall prove that c is a geometric progression. If we pick an arbitrary index $j \in \mathbb{N}$ and define the sequence g so that $g_k = 0$ for all $k \in \mathbb{N}_t - \{j\}$ equation (6.28) implies

$$c_0 g_t + c_{t-j} g_j = h_t a_t = h_t H_t \left(\frac{c_{t-1-j} g_j}{h_{t-1}}, g_t \right) \quad (6.34)$$

for all $j \in \mathbb{N}$, $t > j$, and $g_t, g_j \in \mathbb{C}$.

Given that $c \neq 0$, we can pick $g_j = h_{t-1}x/c_{t-1-j}$, where x is an arbitrary number. If we also pick $g_t = 0$, equation (6.34) becomes

$$\frac{c_{t-j}}{c_{t-1-j}} \frac{h_{t-1}}{h_t} x = H_t(x, 0) \quad (6.35)$$

for all $x \in \mathbb{C}$, $j \in \mathbb{N}$, and $t > j$. Now, for any given $k > 0$, we can pick $t > k$. Then if we combine two versions of the above equation where we pick $j = t-1$

and $j = t - k - 1$, respectively, we obtain

$$\frac{c_1}{c_0} = \frac{c_{k+1}}{c_k}$$

for all $k > 0$. In other words, the sequence c is a geometric progression. If we define $\gamma = c_1/c_0$ then

$$c_t = c_0 \gamma^t \quad (6.36)$$

for all $t \in \mathbb{N}$.

Finally, for any two numbers $x, y \in \mathbb{C}$, we can pick a sequence g such that $g_0 = xh_{t-1}/c_{t-1}$, $g_t = y$, and $g_k = 0$ for all $k \notin \{0, t\}$. Then it will be $a_{t-1} = c_{t-1}g_0/h_{t-1} = x$. Combining the above results with equation (6.28), we obtain

$$\begin{aligned} H_t(x, y) &= \frac{c_0 g_t + g_t c_0}{h_t} \\ &= \frac{c_0}{h_t} y + \frac{\gamma h_{t-1}}{h_t} x \\ &= \lambda_t y + \mu_t x, \end{aligned}$$

where $\lambda_t = c_0/h_t$ and $\mu_t = \gamma h_{t-1}/h_t$ for $t \in \mathbb{N}$. In other words

$$\mathcal{R}_1^H = \mathcal{R}_1^{\lambda\mu} \quad (6.37)$$

Equations (6.36) and (6.37), complete the proof. $\square$

The above theorem demonstrates that convolution has a first-order recursive form if and only if the sequence of coefficients c is constant or a geometric progression. This is not a coincidence. When c is constant or geometric, the convolution is effectively a correlation. (Indeed, if $c_t = c_0 \gamma^t$ then $(c *_h g)_t h_t = \sum_{k=0}^t c_0 \gamma^{t-k} g_k = c_0 \gamma^t (d \star g)_t$, where $d_k = \gamma^{-k}$. The case of a constant sequence c corresponds to $\gamma = 1$.) In those cases, convolution shares the computation and space efficiency of correlation because it is a correlation in ‘disguise’.

Theorem 6.2 can be extended to k -order recursion but the conditions that permit a k -order recursive formulas become considerably more complicated with larger k . The complexity increases further if we allow extra memory to be used.

For example, convolution with coefficients that form an arithmetic progression $c_t = qt + r$, where q and r are constants, admit a first-order recursive form that is augmented with computing and storing the partials sums $\sigma_t = \sum_{k=0}^t g_k$ of the the sequence g . Indeed, if $a = C_{ch}g$, then

$$a_t = \frac{1}{h_t} \sum_{k=0}^t (q(t-k) + r) g_k$$

$$\begin{aligned}
&= \frac{1}{h_t} \left\{ r g_t + \sum_{k=0}^{t-1} (q(t-1-k) + r) g_k + q \sum_{k=0}^{t-1} g_k \right\} \\
&= \frac{1}{h_t} (r - q) g_t + \frac{h_{t-1}}{h_t} a_{t-1} + \frac{q}{h_t} \sigma_t
\end{aligned}$$

Any given sequence convolution can be represented by finite recursion but not by k -order recursion, if k is fixed. In order to explain this in more detail, we need to distinguish between finite and infinite sequences.

Definition 6.8. A sequence $c \in \mathbb{C}^{\mathbb{N}}$ is *infinite* iff it contains an infinite number of non-zero elements

$$(\forall t_0 \in \mathbb{N})(\exists t > t_0) \quad c_t \neq 0.$$

Conversely, c is *finite* iff

$$(\exists t_0 \in \mathbb{N})(\forall t > t_0) \quad c_t = 0.$$

In that case, the *length* of c is defined to be

$$l(c) = 1 + \max\{t \in \mathbb{N} / c_t \neq 0\}.$$

In general, convolution with a finite sequence c usually results in finite recursion (of arbitrary order). Intuitively, we may think that convolution with a sequence c of length $s + 1$ should be representable by a recursion of order $k \leq s + 1$. However, this is not the case. For example, if $c_0 = c_1 = 1$ and $c_t = 0$ for $t > 1$, the sequence c is finite and its length is 2. The sequence $a = c * g$ is then as follows

$$\begin{aligned}
a_0 &= g_0; \\
a_t &= g_t + g_{t-1}, \quad \text{for } t \geq 1.
\end{aligned}$$

However, its recursive form is

$$a_t = H(a_0, \dots, a_{t-1}, g_t) = g_t - \sum_{k=0}^t (-1)^k a_{t-k}$$

This function is not representable by k -order recursion for any fixed k . This may seem counterintuitive but it is a consequence of the fact that we did not include the previous values of g in the definition of recursive sequence transformations (equation (6.13)). In general, if we use a ‘memory’ sequence $\langle m_t / t \in \mathbb{N} \rangle$ to store intermediate results that are used in the recursion, we can redefine k -order recursion as follows

$$(\forall t \in \mathbb{N}) \quad a_t = H_t(m_{\max(0, t-k)}, \dots, m_{t-1}, g_t). \quad (6.38)$$

With the above definition, convolution with a sequence of length $s + 1$ can

always be computed with recursion of order $k = s$ if we merely set $m_t = g_t$. (Indeed, if $n = \max(0, t - s)$ then $a_t = \sum_{i=n}^t c_{t-k} g_k = \sum_{i=n}^{t-1} c_{t-k} m_k + c_0 g_t = H_t(m_n, \dots, m_{t-1}, g_t)$ with an obvious definition for H_t .)

In general, the above definition allows for more flexibility and recursive forms of lower order. However, there is a notable exception. If the stored element m_t is different than a_t and a_t must be retained then we need to store both of them and thus double the space requirement for first-order recursion. This is an important drawback because when we apply recursion at large scale we would like to have a minimal memory overhead.

We will return to this more general form of recursion later in the theory. For the rest of this volume, we will use the original definition – which also makes theoretical results easier to prove.

The situation with convolution transformations that use an infinite sequence c is considerably more complicated, if the minimum recursion order should be used. It is clearly beyond the scope of this volume. In the worst case, convolution can have $\Theta(t)$ space and time complexity. Despite this obvious disadvantage, there is an overall trade-off between correlations and convolutions. There are sequence transformations that can be represented by correlations but not convolutions and vice versa. There are also transformations that cannot be expressed by either of them.*

On the other hand, arbitrary recursion does not have such limitations but it can have arbitrary memory overhead and time complexity. There is a more fundamental and underlying trade-off. In this volume, we have opted for minimum overhead and thus for first-order affine recursion and, equivalently, correlation.

6.4 Correlation and Convolution Examples

In statistics and signal or image processing, *smoothing* refers to processes that approximate data with functions that tend to capture the trends or patterns in the data and ignore noise and details that often lead to short, rapid, and irregular changes. The term derives from the fact that such processes tend to decrease data points that are relatively large in their neighborhood and increase those that are small leading to functions that are ‘smoother’ than the data.

There is a considerable number of smoothing methods in scientific literature. In the end, all such methods can be represented as recursive processes but they are usually derived from different paradigms. A brief summary appears in section 6.7. In this section, we will only consider smoothing that derives from correlation or convolution.

In the following we assume a sequence $g \in \mathbb{C}^{\mathbb{N}}$ that is to be transformed by smoothing operations.

*Hint: consider the sequence transformation $a_t = \sum_{[t/2]}^t g_k$

Definition 6.9. *Exponential smoothing* or *exponential moving average* is a first-order affine recursion $\mathcal{R}_1^{\lambda\mu}$ where $\lambda = 1 - \mu$ and $\mu = \gamma$ for $t > 0$ (μ_0 is always assumed to be 0; γ is a constant, usually a real number in the interval $(0, 1)$). Exponential smoothing is thus defined as

$$\begin{aligned} a_0 &= g_0; \\ a &= (1 - \gamma)g + \gamma a_- \quad : t|\gamma, \quad \text{for } t \geq 1. \end{aligned}$$

The parameter γ is sometimes called the *momentum* of the transformation while the quantity $1 - \gamma$ is called its *smoothing factor*.

Per theorem 6.1, exponential smoothing is equivalent to cumulative correlation C_c with $c_0 = 1$ and $c_t = (1 - \gamma)\gamma^{-t}$ for $t > 0$, which is cumulative correlation with exponential coefficients. (This is the source of the operation's name, too.)

The asymmetry in the correlation coefficients for $t = 0$ and $t > 0$ has the undesirable effect that g_0 has a disproportionate contribution to a_t compared to g_k for small k . Indeed the growth sequence of exponential smoothing is

$$\rho_c = \left\langle \frac{1-\gamma}{\gamma}, \frac{1}{\gamma}, \frac{1}{\gamma}, \dots \right\rangle$$

When $\gamma > 0.5$, which will often be the case, it is $\rho_{c,0} < 1$ but $\rho_{c,k} > 1$ for $k \in \mathbb{N}_+$. The result is that c_0 will be greater than c_k for small k and thus g_0 will have a greater effect on a_t than g_k even though g_k is a more recent element of g relative to time $t > k$. This is demonstrated in the left plot of figure 6.1 where we plot the coefficients c_t that correspond to exponential smoothing for $\gamma \in \{0.80, 0.85, 0.90\}$. As the plot shows, c_7 is still less than c_0

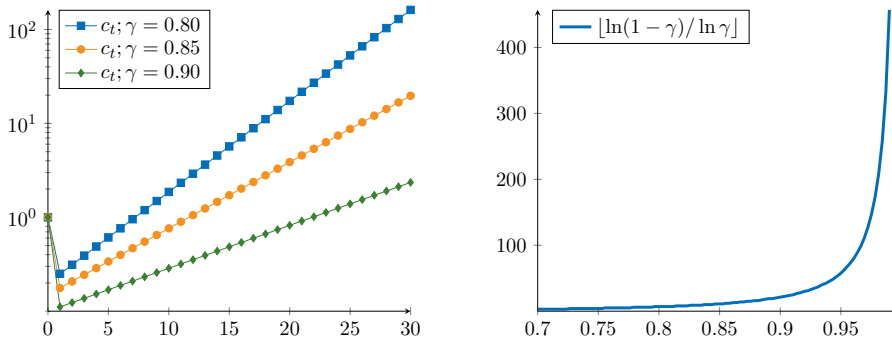


Figure 6.1: The coefficient c_0 is disproportionately large in exponential smoothing (left plot). The number of coefficients that are less or equal to c_0 increases quickly as γ tends to 1 (right plot).

when $\gamma = 0.80$. The same applies to c_{11} for $\gamma = 0.85$ and to c_{21} for $\gamma = 0.90$. The number of coefficients (other than c_0) that are less or equal to c_0 is $\lfloor \ln(1 - \gamma) / \ln \gamma \rfloor$ and it is shown on the right plot of figure 6.1. (This quantity is $\Theta(n \ln n)$, if we define $n = 1/(1 - \gamma)$ and let n go to positive infinity.)

Given that γ is effectively a momentum rate, it is usually close to 1.0 in machine learning applications. It is usually 0.9 or greater for smoothing data. Consequently, the bias of exponential smoothing in favor of initial states and values can be non-trivial and can have a detrimental effect on learning or predictions based on such smoothed data.

On the other hand, we can easily amend the method if we define $c_t = \gamma^{-t}$ for all $t \in \mathbb{N}$. In that case the growth sequence is constant and equal to $1/\gamma$. The corresponding first-order affine recursion $\mathcal{R}_1^{\lambda\mu} = C_c$ is implied by equation (6.24): $\lambda_t = 1 - \mu_t = (1 - \gamma)/(1 - \gamma^{t+1})$ for $t \in \mathbb{N}$. Therefore, we can define an unbiased form of exponential smoothing as follows.

Definition 6.10. *Unbiased exponential smoothing* is a first-order affine recursion $\mathcal{R}_1^{\lambda\mu}$ where

$$\lambda_t = 1 - \mu_t = \frac{1 - \gamma}{1 - \gamma^{t+1}} \quad (6.39)$$

for $t \in \mathbb{N}$. γ is the *momentum* of the transformation and $1 - \gamma$ is its *smoothing factor*.

The actual momentum rate of unbiased exponential smoothing is $\mu_t = 1 - (1 - \gamma)/(1 - \gamma^{t+1})$ but it converges to γ as t tends to infinity, if $|\gamma| < 1$. Similarly, the actual smoothing factor is $\lambda_t = (1 - \gamma)/(1 - \gamma^{t+1})$ but it converges to $1 - \gamma$. (The convergence is exponential in both cases, assuming that $|\gamma| < 1$.) The momentum and the smoothing factor are no longer constant as they are in exponential smoothing; but that is a small price to pay to remove the bias for the initial value g_0 .

Furthermore, we can create a more concise and efficient version of the unbiased exponential smoothing if we observe that

$$\begin{aligned} \lambda_t &= \frac{1 - \gamma}{1 - \gamma^{t+1}} \\ &= \frac{1 - \gamma}{1 - \gamma + \gamma(1 - \gamma^t)} \\ &= \frac{\lambda_{t-1}}{\lambda_{t-1} + \gamma} \end{aligned}$$

We thus obtain the algorithm for the unbiased exponential smoothing that is shown in figure 6.2. The almost trivial update $\lambda = \frac{\lambda_-}{\lambda_- + \gamma} : t|\gamma$ for the learning rate (or smoothing factor) is a minimal overhead and indeed a small price to pay for removing the bias. It is actually surprising that exponential smoothing (often under the name exponential moving average) has been used in statistics and economics for so long and this simple correction was not discovered prior to this theory. The reason is probably the fact that, under exponential smoothing, the contribution of g_0 to a_t is $\mu^t g_0$, which decreases exponentially with t . For large t , as is the case with large data sets, the effect of the bias is negligible in the long term. And if g_0 is close to the average

Unbiased Exponential Smoothing(g, γ)	
$\lambda_0 = 1$	
$a_0 = g_0$	
for $t = 1, 2, \dots$	
λ	$= \frac{\lambda_-}{\lambda_- + \gamma}$
a	$= \lambda g + (1 - \lambda)a_-$
$\left. \vphantom{\begin{matrix} \lambda \\ a \end{matrix}} \right\} : t \gamma$	

Figure 6.2: The unbiased exponential smoothing algorithm.

$(g_1 + \dots + g_k)/k$, for some small k , the effect of the bias would be limited for small t , too.

However, the above conditions are generally not true for learning. A learner should not (or cannot) delay its learning until its data set or its learning experiences are as numerous as to overwhelm the bias. A learning sequence $\langle g_t/t \in \mathbb{N} \rangle$ can have significantly different initial values, too. For example, if g_t is the gradient of a loss function with respect to a parameter of a neural network for training iteration t (where each iteration is over a small subset of the training set, which is usually called a *batch* or *mini-batch*) then g_0 will tend to be larger than the average $(g_1 + \dots + g_k)/k$. As a result, exponential smoothing may introduce a significant bias early in training and make the learning unnecessarily sensitive to initial conditions. Unbiased exponential smoothing is clearly a better option.

Given that unbiased exponential smoothing implies a correlation with a sequence c that is a geometric progression, theorem 6.2 applies. As a result,

$$\mathcal{R}_1^{\lambda\mu} = C_c = K_{1/c}.$$

The latter equation is merely a more concise version of the tautology

$$\frac{\sum_{k=0}^t g_k \gamma^{-k}}{\sum_{k=0}^t \gamma^{-k}} = \frac{\sum_{k=0}^t g_k \gamma^{t-k}}{\sum_{k=0}^t \gamma^{t-k}}$$

In other words, unbiased exponential smoothing is cumulative correlation with the sequence $\langle \gamma^{-t}/t \in \mathbb{N} \rangle$, which is also cumulative convolution with the sequence $\langle g_t/t \in \mathbb{N} \rangle$. In this regard, it is not only better than exponential smoothing but also simpler.

The above analysis demonstrates the utility of associating recursion with correlation or convolution and examining a recursive transformation in that context. The result is not different than solving the corresponding recursive equation but the context can offer us intuitions that can lead to improvements like the unbiased exponential smoothing method.

Let us now consider a convolution example.

Definition 6.11. *Simple moving average* is cumulative correlation C_c or

convolution K_c where $c_t = 1$ for all $t \in \mathbb{N}$. Equivalently, it is first-order affine recursion $\mathcal{R}_1^{\lambda\mu}$ where $\lambda_t = 1 - \mu_t = 1/(t+1)$.

The above definition implies that

$$a_t = \frac{1}{t+1} \sum_{k=0}^t g_k$$

However, if we limit the average to the n most recent values of g we obtain the following definition

Definition 6.12. *Simple moving average of length n* is cumulative convolution K_c where

$$c_t = (t < n) = \begin{cases} 1 & \text{if } t < n \\ 0 & \text{otherwise} \end{cases}$$

for all $t \in \mathbb{N}$.

This transformation is a convolution but not a correlation or first-order recursion.

$$a_t = \begin{cases} \frac{1}{t+1} \sum_{k=0}^t g_k & \text{if } t < n \\ \frac{1}{n} \sum_{k=0}^{n-1} g_{t-k} & \text{otherwise.} \end{cases}$$

Even if we use the alternative definition of recursion, as in equation (6.38), a recursion of order n is required to implement this function. We thus consider it unlikely for large scale neural transformations at a synaptic level.

Let us now return to correlations, which are computationally more efficient and thus more likely (if we assume that computations translate to energy consumption and evolution tends to create more energy-efficient forms.)

An important property of correlation transformations is their growth (and aging) if we assume that more recent values of a sequence should have greater weight. The following lemma shows the condition for aging applied to first-order affine recursion.

Lemma 6.2. *Let $\mu \in \mathbb{R}^{\mathbb{N}}$ be a sequence such that $\mu_0 = 0$ and $\mu_t \neq 0$ for all $t \in \mathbb{N}_+$. Then a first-order affine recursion $\mathcal{R}_1^{\lambda\mu}$ is aging iff*

$$1 \leq \frac{\lambda_+}{\lambda\mu_+} \quad : t \in \mathbb{N} \quad (6.40)$$

When $\lambda = 1 - \mu$ and $0 < \lambda < 1 : t \in \mathbb{N}_+$, the above aging condition is equivalent to either of the following two equations

$$\lambda_+ \leq \lambda/(1 + \lambda) \quad : t \in \mathbb{N} \quad (6.41)$$

$$\mu_+ \leq 1/(2 - \mu) \quad : t \in \mathbb{N} \quad (6.42)$$

The above conditions are also equivalent to the inequalities below

$$\left. \begin{aligned} \Delta\mu_+ &= \mu_+ - \mu \leq (1 - \mu)^2/(2 - \mu) \\ \Delta\lambda_+ &= \lambda_+ - \lambda \leq -\lambda^2/(1 + \lambda) \end{aligned} \right\} : t|\Delta$$

Proof. Given that $\mu_t \neq 0$ for all $t \in \mathbb{N}_+$, theorem 6.1 implies that there exists a sequence c such that $C_c = \mathcal{R}_1^{\lambda\mu}$. Consequently aging for $\mathcal{R}_1^{\lambda\mu}$ is equivalent to

$$1 \leq \frac{c_+}{c} = \frac{\lambda_+ s_+}{\lambda s} = \frac{\lambda_+}{\lambda \mu_+} \quad : t \in \mathbb{N}.$$

If we assume that $\lambda = 1 - \mu$ the aging condition becomes

$$1 \leq \frac{c_+}{c} = \frac{\lambda_+}{\lambda(1 - \lambda_+)} \quad : t \in \mathbb{N}.$$

If we furthermore assume that $0 < \lambda < 1 : t \in \mathbb{N}_+$ and solve the above inequality for λ_+ , we obtain $\lambda_+ \geq \lambda/(1 + \lambda)$. Given that $0 < \mu = 1 - \lambda < 1 : t \in \mathbb{N}_+$, this is equivalent to $\mu_+ \leq 1/(2 - \mu)$.

Moreover,

$$\left. \begin{aligned} \Delta\lambda_+ &= \lambda_+ - \lambda \geq \lambda/(1 + \lambda) - \lambda = -\lambda^2/(1 + \lambda) \\ \Delta\mu_+ &= \mu_+ - \mu \leq 1/(2 - \mu) - \mu = (1 - \mu)^2/(2 - \mu) \end{aligned} \right\} : t|\Delta.$$

□

Note that the above lemma implies that the smoothing factor λ can actually decrease over time and the transformation still satisfy aging provided that $\Delta\lambda_+ \geq -\lambda^2/(1 + \lambda) : t|\Delta$.

In general, aging is a desirable condition. It is equivalent to the convexity of the sequence $\langle s_t/t \in \mathbb{N} \rangle$ since $c_+ - c = s_+ - 2s + s_- : t \in \mathbb{N}_+$. For any non-convex sequence s , there will be an option in the learning algorithms of the next chapter to enforce aging (using constraint (6.42) to bound μ_+).

As a final example of this section, we can examine cumulative correlations with various growth rates. Note that equation (6.24) implies that $c = s - s_-$, $\lambda = 1 - \mu = 1 - s_-/s = c/s$, and $s_0 = c_0 = 1$. Consequently, we can create correlations with different growth rates if we pick different forms for the sequence s . Table 6.1 shows some examples roughly in increasing growth order. c_0 is always assumed to be 1.

Note that the learning rate $\lambda_t = 1/(t + 1)$ of the constant correlation is the minimum possible learning rate for cumulative correlation when aging is true. This derives from the fact that $\lambda = c/(s_- + c) = 1/(s_-/c + 1) : t$, which is an increasing function of c_t when $c \geq 0$. Given that c_{t-1} is the minimum value of c_t under aging, the minimum value of λ_t is obtained when $c = c_- : t > 0$, that is when c_t is constant. Given that $c_0 = 1$, this implies $c = 1$ and thus $\lambda_t = 1/(t + 1)$. (The logarithmic correlation has a smaller

Description	c_t	s_t	λ_t	$\rho_{c,t}$
Logarithmic	$\log_2 \frac{t+2}{t+1}$	$\log_2(t+2)$	$1 - \frac{\log_2(t+1)}{\log_2(t+2)}$	$\frac{\log_2(1+1/(t+2))}{\log_2(1+1/(t+1))}$
Constant	1	$t+1$	$\frac{1}{t+1}$	1
Diagonal	$t+1$	$\frac{(t+1)(t+2)}{2}$	$\frac{2}{t+2}$	$\frac{t+2}{t+1}$
Linear $s \in \mathbb{R}_+$	$st+1$	$\frac{(t+1)(st+2)}{2}$	$\frac{2(st+1)}{(t+1)(st+2)}$	$\frac{s(t+1)+1}{st+1}$
Binomial $n \in \mathbb{N}_+$	$\binom{t+n-1}{n-1}$	$\binom{t+n}{n}$	$\frac{n}{t+n}$	$\frac{t+n}{t+1}$
Polynomial $d \in \mathbb{R}_+$	$(t+1)^d - t^d$	$(t+1)^d$	$\frac{(t+1)^d - t^d}{(t+1)^d}$	$\frac{(t+2)^d - (t+1)^d}{(t+1)^d - t^d}$
Biased exponential $\gamma \in (0, 1)$	$(1-\gamma)\gamma^{-t}$	γ^{-t}	$1-\gamma$	$\begin{cases} 1/\gamma-1 & \text{if } t=0 \\ 1/\gamma & \text{else} \end{cases}$
Unbiased exponential $\gamma \in (0, 1)$	γ^{-t}	$\frac{\gamma^{-t}-\gamma}{1-\gamma}$	$\frac{1-\gamma}{1-\gamma^{t+1}}$	$\frac{1}{\gamma}$
Exponential decay $\beta \in (0, 1)$, $\gamma \in (\beta, 1]$	$\frac{1-\gamma+\beta^t}{(2-\gamma)\prod_{k=1}^t(\gamma-\beta^k)}$	$\frac{1}{(2-\gamma)\prod_{k=1}^t(\gamma-\beta^k)}$	$1-\gamma+\beta^t$	$\frac{1-\gamma+\beta^{t+1}}{(1-\gamma+\beta^t)(\gamma-\beta^{t+1})}$
Geometric progression $\beta \geq -1$, $\gamma \in (0, 1]$	$\frac{1-\gamma+\beta^t(\gamma-\beta)}{(1-\beta)\gamma^t}$	$\frac{1-\beta^{t+1}}{1-\beta^t}\gamma^{-t}$	$1-\gamma\frac{1-\beta^t}{1-\beta^{t+1}}$	$\frac{1}{\gamma}\frac{1-\gamma+\beta^{t+1}(\gamma-\beta)}{1-\gamma+\beta^t(\gamma-\beta)}$
Exponential polynomial $n \in \mathbb{N}_+$, $\gamma \in (0, 1)$	$\binom{t+n}{n} \frac{t+n-\gamma t}{(t+n)\gamma^t}$	$\binom{t+n}{n} \gamma^{-t}$	$1 - \frac{\gamma t}{t+n}$	$\frac{1}{\gamma} \frac{t+n}{t+1} \left(1 + \frac{1-\gamma}{t+n-\gamma t}\right)$
Factorial	$tt!$	$(t+1)!$	$\frac{t}{t+1}$	$\begin{cases} \frac{1}{(t+1)^2} & \text{if } t=0 \\ \frac{1}{t} & \text{else} \end{cases}$

Table 6.1: Examples of cumulative correlations and the equivalent first-order recursions.

learning rate but it violates aging since $\log_2$ is not a convex function; and indeed it is $c_{t+1} = \log_2(1 + \frac{1}{t+2}) < \log_2(1 + \frac{1}{t+1}) = c_t$.)

Given the above bound on the learning rate, the maximum momentum under aging is $\mu = 1 - 1/(t+1) = t/(t+1)$. This implies a maximum value of 0.5 for μ_1 . In other words, if a correlation or smoothing process should use a momentum $\gamma > 0.5$, they should start at 0.5 and gradually increase it to γ (in at least $\lceil 1/(1-\gamma) \rceil - 1$ steps) if aging is desired.

The unbiased exponential and exponential polynomial forms have been used in many of our experiments. The exponential polynomial momentum is $\mu_t = \frac{\gamma t}{t+n}$. It converges to γ as t tends to infinity while the parameter n affects the initial value μ_1 and the speed of convergence. Intuitively,

the exponential component determines the limit of the momentum but the polynomial component affects its initial value and convergence. The parameter β plays a similar role that affects the initial value and convergence rate in the cases of exponential decay and geometric progression of table 6.1. (In the later case, β also affects the limit if $\beta > 1$.) The momentum of those forms is relatively flexible. In comparison, the biased exponential momentum is fixed while the unbiased exponential momentum converges to a limit γ but its initial value and convergence rate are uniquely determined by its limit γ .

Empirical evidence is required to find the proper momentum form in each learning case. However, the factorial correlation has such fast growth that the momentum μ_t converges to 0 as t goes to infinity. This has a diminishing effect on coherence as we shall see in the next section. We thus consider it of limited utility and we do not expect it to arise in applications. We only include it as an extreme case that is most likely an upper bound, too.

6.5 Recursion and Coherence

In the previous sections, we introduced sequence transformations through correlation, convolution, and finite recursion. The intention behind such transformations is that they affect coherence in predictable ways. This is demonstrated by theorem 6.3 below. However, we first need to define certain quantities.

We consider sequences $g \in \mathbb{C}^{\mathbb{N}}$. We assume the absolute value $|\cdot|$ as a norm in $\mathbb{Z}$ and $\|\cdot\|$ to be a norm in $\mathbb{C}$. We assume ρ to be a ratio estimator such that $\rho(x/y) = x/y$, when x and y are numbers or tuples of length 1. (For example ρ could be R_1 or R_2 .) Finally, we assume an association function $\alpha: \mathbb{N} \rightarrow \mathbb{N}^*$ such that $\alpha(t) = (t-1)$ for all $t \in \mathbb{N}_+$. Then the metric difference of g at $t \in \mathbb{N}_+$ with respect to ρ and α is

$$\Delta_{\rho\alpha}g_t = \rho\left(\frac{\|g_{t-1} - g_t\|}{|t-1-t|}\right) = \|g_t - g_{t-1}\| \quad (6.43)$$

and the first-order incoherence of g in a finite subset $S \subset \mathbb{N} - \{0\}$ with respect to ρ and α is

$$I_{\rho\alpha}g(S) = \frac{1}{|S|} \sum_{t \in S} \|g_t - g_{t-1}\|, \quad (6.44)$$

where $|S|$ indicates the number of elements of S .

We can now present a theorem that demonstrates that correlation decreases incoherence (and thus increases coherence).

Theorem 6.3. *Let $g \in \mathbb{C}^{\mathbb{N}}$ be an arbitrary sequence and $b = C_c g$ be its cumulative correlation transformation by the real sequence $c \geq 0$. (We assume that C_c is well-defined and thus we assume that $s_t = \sum_{k=0}^t c_k \neq 0$ for all $t \in \mathbb{N}$.) We assume ρ to be a ratio estimator such that $\rho(x/y) = x/y$, when x and y are numbers or tuples of length 1. We also assume the association*

function $\alpha: \mathbb{N} \rightarrow \mathbb{N}^*$ defined as $\alpha(t) = (t - 1)$ for all $t \in \mathbb{N}_+$. Then the incoherence of the sequence b in $S_t = \{1, \dots, t\}$ with respect to ρ and α is bound by the corresponding incoherence of g as follows

$$(\forall t \in \mathbb{N}_+) \quad \mathbf{I}_{\rho\alpha} b(S_t) \leq \mathbf{I}_{\rho\alpha} g(S_t) - \frac{1}{t} \sum_{k=1}^t \|g_k - g_{k-1}\| \prod_{j=k}^t \mu_j \quad (6.45)$$

where λ and μ are defined so that $C_c = \mathcal{R}_1^{\lambda\mu}$.

Proof. Theorem 6.1 implies the existence of the above sequences λ and μ while equation (6.24) furthermore implies that

$$\lambda = 1 - \mu = 1 - \frac{s_-}{s} = \frac{c}{s} \quad : t > 0$$

Consequently

$$s_t = \prod_{k=1}^t \frac{s_k}{s_{k-1}} = \prod_{k=1}^t \frac{1}{\mu_k} \quad (6.46)$$

Given that $c \geq 0$, it is $s > 0$ and thus $\lambda \geq 0$ and $\mu > 0$. The fact that $\mu = 1 - \lambda$ then implies that $0 < \mu \leq 1$ and $0 \leq \lambda < 1$.

Now if we define

$$\delta = (g_+ - b)s \quad : t \quad (6.47)$$

we can derive a telescopic recursion for it. Indeed, it is

$$\begin{aligned} \delta &= (g_+ - b)s \\ &= (g_+ - \lambda g - \mu b_-)s && (\text{since } b = C_c g = \mathcal{R}_1^{\lambda\mu} g) \\ &= (g_+ - g + \mu(g - b_-))s && (\text{due to } \lambda = 1 - \mu) \\ &= (g_+ - g)s + (g - b_-)s_- && (\text{since } \mu s = s_-) \\ &= (g_+ - g)s + \delta_- && : t > 0. \end{aligned}$$

Now, by summation and elimination of the common terms, we obtain

$$\delta_t = \delta_0 + \sum_{k=1}^t (g_{k+1} - g_k)s_k = \sum_{k=0}^t (g_{k+1} - g_k)s_k$$

since $\delta_0 = (g_1 - b_0)s_0$ and $b_0 = g_0$. Furthermore, it is

$$b_+ - b = \lambda_+ g_+ + \mu_+ b - b = \lambda_+ (g_+ - b) = \lambda_+ \delta / s \quad : t.$$

Combining the last two equations, we obtain

$$b_+ - b = \frac{\lambda_+}{s} \sum_{k=0}^t (g_{k+1} - g_k)s_k \quad : t|t. \quad (6.48)$$

As a result

$$\begin{aligned}
t\mathbf{I}_{\rho\alpha}b(S_t) &= \sum_{j=0}^{t-1} \|b_{j+1} - b_j\| && \text{(by definition)} \\
&= \sum_{j=0}^{t-1} \left\| \frac{\lambda_{j+1}}{s_j} \sum_{k=0}^j (g_{k+1} - g_k) s_k \right\| && \text{(by (6.48))} \\
&\leq \sum_{j=0}^{t-1} \sum_{k=0}^j \frac{\lambda_{j+1}}{s_j} \|g_{k+1} - g_k\| s_k && \text{(by triangle inequality)} \\
&= \sum_{k=0}^{t-1} \|g_{k+1} - g_k\| s_k \sum_{j=k}^{t-1} \frac{\lambda_{j+1}}{s_j} && \text{(rearranging terms)} \\
&= \sum_{k=0}^{t-1} \|g_{k+1} - g_k\| s_k \sum_{j=k}^{t-1} \left(\frac{1}{s_j} - \frac{1}{s_{j+1}} \right) && \text{(since } \lambda_{j+1} = 1 - \mu_{j+1}) \\
&= \sum_{k=0}^{t-1} \|g_{k+1} - g_k\| s_k \left(\frac{1}{s_k} - \frac{1}{s_t} \right) && \text{(telescopic inner sum)} \\
&= \sum_{k=1}^t \|g_k - g_{k-1}\| \left(1 - \frac{s_{k-1}}{s_t} \right) && \text{(shifting index } k \text{ by 1)} \\
&= t\mathbf{I}_{\rho\alpha}g(S_t) - \sum_{k=1}^t \|g_k - g_{k-1}\| \frac{s_{k-1}}{s_t} && \text{(by definition of incoherence)} \\
&= t\mathbf{I}_{\rho\alpha}g(S_t) - \sum_{k=1}^t \|g_k - g_{k-1}\| \prod_{j=k}^t \mu_j && \text{(by (6.46))}
\end{aligned}$$

□

Theorem 6.3 shows that cumulative correlation with non-negative coefficients (or equivalently first-order affine recursion with positive momentum rates) reduces the incoherence of a sequence g by at least

$$\frac{1}{t} \sum_{k=1}^t \|g_k - g_{k-1}\| \prod_{j=k}^t \mu_j$$

in the first $t + 1$ elements of the sequence g . This equation helps explain why momentum rates are usually 0.9 or higher. If momentum rates were considerably smaller the products $\prod_{j=k}^t \mu_j$ would diminish quickly and the reduction of incoherence would probably diminish as fast.* We will return to

*Theorem 6.3 only provides a lower bound of the reduction of incoherence and thus it does not necessarily imply that the reduction diminishes when the partial products of

this topic in the next chapter.

We can extend the above result to non-cumulative correlation C_{ch} (under a reasonable set of constraints for the function h). For such correlations, the condition $\lambda = 1 - \mu$ is not necessarily true and the proof is thus more complicated. However, the overall conclusion is the same. Transformations derived from correlation or first-order affine recursion increase the first-order coherence of sequences (provided that the coefficients of the correlation are non-negative and the normalization function is positive or, equivalently, that the learning rates of the affine recursion are non-negative and the momentum rates are positive). As a result, such transformations are valuable tools for learning. They allow us to unify and understand existing learning and regularization algorithms; and they help us develop more sophisticated methods.

For higher orders of incoherence, a reduction is not guaranteed with such relatively simple transformations. (The same applies to increases of higher order coherence, of course.) For example, if g is affine, its second-order incoherence is zero and any correlation or convolution with non-affine coefficients is bound to increase it. Of course, affine sequences are unlikely to occur in nature or real applications, but the effect of correlation or first-order recursion on second or higher-order coherence or incoherence is generally not as predictable as with first-order. For higher orders, more sophisticated transformations are usually required.

6.6 Transformation Bias

In section 5.3, we discussed statistical estimators. We saw that estimators can be biased or unbiased depending on whether they generate a biased or unbiased estimate of the expected value of their estimand. In a similar fashion, we can view sequence transformations as estimators of stochastic processes that are represented by the input sequences. We can then define the bias of transformations.

Definition 6.13. The *transformation bias* of a sequence transformation T is defined as

$$\text{Bias}(T) = E_g(Tg - g),$$

where g represents a stochastic sequence, Tg is the transformation of g by T , and E_g represents expectation over sequences g that are generated by an underlying stochastic process. A *biased* or *unbiased transformation* is a sequence transformation with non-zero or zero bias, respectively.

the momentum rates are small or trivial. However, in practice, and particularly for real sequences g and with absolute value as the norm, the above bound tends to be within a constant factor of the actual reduction. This is because the incoherence reduction has then the form $\frac{1-\epsilon_t}{t} \sum_{k=1}^t |g_k - g_{k-1}| \prod_{j=k}^t \mu_j + \frac{\epsilon_t}{t} \sum_{k=1}^t |g_k - g_{k-1}|$ where ϵ_t is a small quantity that depends on the sequence g but often diminishes quickly.

The following theorem shows that cumulative correlations and their corresponding first-order affine recursions are the only such transformations that are unbiased for stationary sequences with non-zero mean.

Theorem 6.4. *A first-order affine transformation $\mathcal{R}_1^{\lambda\mu}$ is unbiased for stationary sequences with non-zero mean iff $\lambda = 1 - \mu$. Equivalently, a correlation transformation is unbiased for such sequences iff it is cumulative.*

Proof. Let g be the input sequence and $a = \mathcal{R}_1^{\lambda\mu}g$ be its first-order affine transformation by $\mathcal{R}_1^{\lambda\mu}$. Then $Ea = \mu Ea_- + \lambda Eg : t > 0$. Stationarity implies that $Ea = Ea_-$ and thus $Ea = Eg$ iff $\lambda = 1 - \mu$ (given that $Eg \neq 0$).

Similarly, if $a = C_{ch}g$ then $Ea = \frac{1}{h} \llbracket \sum_{i=0}^t c_i \rrbracket Eg : t \geq 0$. Therefore $Ea = Eg \neq 0$ iff $h_t = \sum_{i=0}^t c_i$. $\square$

Unbiased transformations play a significant role in learning algorithms. They also help reduce parameterization since they uniquely define the learning rate λ from the momentum rate μ .

On the other hand, transformations do not have to be unbiased. Transformation bias can be useful in various situations. Stationarity is also unlikely to occur early in learning but it is more likely in later stages. In either case, the bias property allows us to understand learning methods better and define alternative unbiased (or biased) forms as we shall see in the next chapter. Bias also provides a perspective that allows us to compare different learning algorithms in a way that is statistically more accurate. In the next chapter, we will elaborate on this issue, too.

6.7 Historical and Bibliographical Remarks

Cross-correlation and convolution have a long history and various applications and interpretations in signal processing and functional analysis [261, 87].

Function or sequence definitions by induction can be traced at least as far back as the Fibonacci sequence. The latter was introduced to European mathematics in 1202 by Leonardo of Pisa, later known as Fibonacci [271], but the sequence had appeared earlier in Indian mathematics circa 200 BCE and between 600 CE and 800 CE (A. Pingala's *matrameru* relates to Fibonacci numbers while a method for calculating the sequence was given by Virahanka [132, 318, 190]).

R. Dedekind proved the uniqueness of inductively defined functions in 1888 [78] while Peano used induction to axiomatize arithmetic on natural numbers in 1889 [265].

The term *recursive function* first appeared in Gödel's famous work on the incompleteness of elementary number theory in 1931 [140]. Those functions were later renamed to *primitive recursive functions* by S. Kleene. In modern set theory terms, they represent first-order finite recursion and a straightforward

application of induction to define functions over the set of natural numbers $\mathbb{N}$.

μ -recursive functions are also functions on natural numbers. They originated in the works of J. Herbrand and K. Gödel in the 1930s [154, 141] based on the known fact at the time that not all ‘computable’ functions can be represented by primitive recursive functions (a common example being the Ackermann function which was published in 1928 [4]).

The μ -recursive functions contain the *constant* and *projection functions* as well as the *successor function* and they are closed under function composition, finite recursion, and minimization [103]. They represent a broader set of functions than finite or primitive recursive functions because of their inclusion of the minimization operator. The class of μ -recursive functions is actually identical to the class of *Turing-computable functions* that was introduced by Alan Turing as well as the class of the *λ -definable functions* introduced by Alonzo Church [345, 54, 188, 77].

The *Church-Turing thesis* attempts to specify the informal set of functions that can be calculated by a method. It appeared in two equivalent forms in 1936.

Its version on λ calculus and recursive functions was introduced by A. Church [54, 188]. An earlier version used the term λ -definable functions and was rejected by Gödel as “thoroughly unsatisfactory”. Church then replaced the later term with recursive functions, referring to the Herbrand-Gödel μ -recursive functions.

The second formulation of computable functions occurred in the context of Turing machines and was proposed by A. Turing [344]. The equivalence between λ -definable functions, Turing machines, and μ -recursive functions was also proved by Turing [345]. Gödel enthusiastically accepted Turing’s thesis and thereafter gave him credit for the definition of computability.

Smoothing methods have been introduced in various mathematical or scientific paradigms. Exponential smoothing (exponential moving average) derives from correlation. The underlying exponential window is attributed to S. Poisson but it first appeared in statistics in late 1950s [36, 161]. Methods such as simple moving average with a fixed window and the *Savitzky-Golay filter* derive from convolution [296]. The *Kolmogorov-Zurbenko filter* is a method that consists of iterations of moving averages over moving averages [404]. (This corresponds to higher-order recursion and it will be discussed in volume II.) Methods such as the *Ramer-Douglas-Peucker algorithm* (a.k.a. *split-and-merge algorithm*) [277, 88, 98] and *Laplacian smoothing* [156] are simple iterative methods. Finally, there are many methods that have emerged from filtering and signal processing. Those include *Kalman filter*, *low-pass filter*, *Butterworth filter*, and *Chebyshev filter* [180, 401, 275, 41, 389].

Although this may seem a paradox, all exact science is dominated by the idea of approximation.

— Bertrand Russell, *The Scientific Outlook*, 1931.

7.1	The Case for Recursion	235
7.1.1	μ -Recursive Functions	235
7.1.2	Recursion in Learning	238
7.2	Synaptic Coherence	240
7.3	Synaptic Gradient Coherence	247
7.3.1	Momentum Gradient Descent	247
7.3.2	Synaptic Cross-Gradient Coherence	250
7.4	The Syc Algorithm	252
7.5	Neural Coherence	254
7.6	Factorized Neural Coherence	255
7.7	The Sync Algorithm	256
7.8	A Research Note	262
7.9	Historical and Bibliographical Remarks	262

Coherent Learning

7.1 The Case for Recursion

Recursion is virtually omnipresent in learning. It extends from recurrent networks and the simplest learning algorithms such as gradient descent (as we shall see in the next section) to complex synergetic components like those of chapter 4 as well as more sophisticated coherent structures and methods that will be presented later in this theory. In this section, we will try to understand the necessity and origins of this common occurrence of recursion.

In order to understand our subject in a more complete way, we will formally define μ -recursive functions in the next subsection. Readers can skip the subsection if they are familiar with the subject. (Our notation is somewhat more concise than the standard notation for μ -recursive functions but it is equivalent to it.)

7.1.1 μ -Recursive Functions

Let us first define partial and total functions. In general, given two sets D and R and a function $f: D \rightarrow R$, we would like to allow f to be potentially undefined in some elements $x \in D$. In other words, we would like to allow the possibility that $\text{dom } f \subset D$.

We will use the notation $f(x) = \uparrow$ to indicate that the function $f: D \rightarrow R$ is undefined at $x \in D$. In order to enable the composition of such functions, $\uparrow$ must be included as a potential input to the functions. For example, if we define $f(x) = g(h(x))$ and h is undefined at x , f must be undefined at x and thus it should be $g(\uparrow) = g(h(x)) = f(x) = \uparrow$.

For compactness, we use the definition

$$D_{\uparrow} = D \cup \{\uparrow\}$$

and we assume that $\uparrow$ has higher precedence than the Kleene star. Thus $D_{\uparrow}^* = (D_{\uparrow})^*$.

Given that a function $f: D_{\uparrow} \rightarrow R_{\uparrow}$ can be seen as the restriction of a

function $F: D_{\uparrow}^* \rightarrow R_{\uparrow}$ to $D_{\uparrow}$, we will consider the domain to be $D_{\uparrow}^*$ in all cases. This leads to the following more formal definition of partial functions.

Definition 7.1. Given two sets D and R , a function $f: D_{\uparrow}^* \rightarrow R_{\uparrow}$ is *partial* iff

$$(\forall x \in D_{\uparrow}^* - D^*) \quad f(x) = \uparrow$$

In that case we define

$$\begin{aligned} \text{dom } f &= \{x \in D^* / f(x) \in R\} \\ \text{ran } f &= \{y \in R / (\exists x \in D^*) f(x) = y\}. \end{aligned}$$

We furthermore define f to be a *total function* in D^* iff $\text{dom } f = D^*$ (or, equivalently, $fD^* \subseteq R$). For any $n \in \mathbb{N}_+$, we define f to be a *total function* in D^n iff $(\forall x \in D^n) f(x) \in R$ (or, equivalently $fD^n \subseteq R$). Finally, we define a function f to be *strictly partial* in D^* or D^n iff it is partial but not total in D^* or D^n , respectively

We assume that functions are equal when undefined. Hence, if $f(x) = \uparrow$ and $g(y) = \uparrow$ then it is $f(x) = g(y)$ for all g, f, x , and y .

For simplicity, we use a concise tuple notation for functions over tuples. Thus if $x = (x_0, \dots, x_{k-1}) \in D^k$, $y = (y_0, \dots, y_{m-1}) \in D^m$, and $z = (z_0, \dots, z_{n-1}) \in D^n$, for some $k, m, n \in \mathbb{N}_+$ then

$$\begin{aligned} f(x, y, z) &= f(x, y, z_1, \dots, z_{n-1}) \\ &= f(x_1, \dots, x_{k-1}, y, z_1, \dots, z_{n-1}) \\ &= f(x_1, \dots, x_{k-1}, y_0, \dots, y_{m-1}, z_1, \dots, z_{n-1}) \end{aligned}$$

for all functions $f: D_{\uparrow}^* \rightarrow R_{\uparrow}$. In other words, any contiguous partitioning of the arguments of a function f into tuples is merely an alternative notation for the same arguments, provided that the order of the arguments remains the same. (We defined indexing of multidimensional arrays the same way in section 1.2).

We can now proceed with the definition of μ -recursive functions.

Definition 7.2. The μ -recursive functions (or general recursive functions) is the smallest set of partial functions $\mathcal{M} \subseteq \mathbb{N}_{\uparrow}^{\mathbb{N}^*}$ that has the following properties

A. $\mathcal{M}$ contains the *constant functions*:

$$(\forall n \in \mathbb{N}) \quad C_n \in \mathcal{M}$$

where $C_n(x) = n$ for all $x \in \mathbb{N}^*$.

B. $\mathcal{M}$ contains the *successor function*:

$$S \in \mathcal{M}$$

where $S(x) = S((x)) = x + 1$ for all $x \in \mathbb{N}$. (If x is a tuple with two or more elements then $S(x) = \uparrow$.)

C. $\mathcal{M}$ contains the *projection functions*:

$$(\forall j \in \mathbb{N}) \quad P_j \in \mathcal{M}$$

where $P_j(x) = x_j$ for all $k > j$ and $x \in \mathbb{N}^k$. (If $x \in \mathbb{N}^k$ and $k \leq j$ then $P_j(x) = \uparrow$.)

D. $\mathcal{M}$ is closed under function composition:

$$(\forall n \in \mathbb{N}_+) (\forall g_0, \dots, g_n \in \mathbb{N}_+^{\mathbb{N}^*}) \\ g_0 \circ (g_1, \dots, g_n) \in \mathcal{M} \text{ iff } g_0, \dots, g_n \in \mathcal{M}.$$

E. $\mathcal{M}$ is closed under finite recursion:

$$(\forall f, g, r \in \mathbb{N}_+^{\mathbb{N}^*}) \quad r \in \mathcal{M} \text{ iff } f, g \in \mathcal{M}$$

where r is defined recursively

$$r(z, x) = \begin{cases} f(x) & \text{if } z = 0 \\ g(r(y, x), y, x) & \text{if } z = S(y). \end{cases}$$

F. $\mathcal{M}$ is closed under the minimization operator μ :

$$(\forall f \in \mathbb{N}_+^{\mathbb{N}^*}) \quad \mu f \in \mathcal{M} \text{ iff } f \in \mathcal{M}$$

where $\mu f(x) = z$ iff $f(z, x) = 0$ and $(\forall y < z) f(y, x) > 0$.

Alternatively, we could replace property A with the condition that the zero function $Z \in \mathcal{M}$, where $Z(x) = 0$ for all $x \in \mathbb{N}^*$. Then C_n can be constructed by composing Z with the successor function S n times

$$C_n = \underbrace{S \circ \dots \circ S}_{n \text{ times}} \circ Z$$

μ -recursive functions have been shown to be *universal*, which means that they are equivalent to Turing machines. In other words, $\mathcal{M}$ is precisely the set of functions that Turing machines can compute.

Let us now try to understand the necessity of the above six properties (A through F) in the definition.

Properties A and B, which define constant functions and the successor function, can be used to construct arithmetic on natural numbers: addition can be defined as successive applications of the successor function,* the *predecessor function* can be defined as the inverse of the successor function (through recursion, undefined at 0), subtraction can be defined as successive applications of the predecessor function, multiplication as successive applications of addition, and integer division as successive applications of subtraction.

Property C is for projection functions and enables extraction of elements from tuples and thus the representation of tuple functions as functions of the elements of the tuples. As a result, this property allows multidimensional functions to be included in $\mathcal{M}$.

Property D permits function composition and therefore the construction

*Formally, such a definition also requires composition and recursion and thus it requires properties A, B, D, and E.

of more complex functions from simpler ones (such as the construction of addition from the successor function S or C_n from Z .)

Property E is for finite recursion and enables the construction of functions not only from other functions but also incrementally from their own previous values.

Finally, property F is for minimization and allows for the calculation of the minimum element $z \in \mathbb{N}$ that satisfies a property $f(z, x) = 0$. (Note that $f(z, x) = t$ is equivalent to $f(z, x) - t = 0$ and $f(z, x) < t$ is equivalent to $f(z, x) = s$ for some $s < t$.)

At a more elementary level, the minimization property introduces the equivalent of a *conditional statement* (from programming languages) which allows computations to branch in two distinct ways. It is not a surprise then that μ -recursive functions are equivalent to Turing machines. (For example, iteration can easily be constructed through recursion and a condition and thus through recursion and minimization.)

Recursion and computability constituted two distinct trends in mathematics that converged to the same set of “computable” functions in the 1930s as we discussed in section 6.7. K. Gödel first came up with a simpler form of recursion in 1931 but after he realized that his definition did not include all computable functions, he modified his definition in 1934 and proposed the general recursive functions, which are equivalent to the μ -recursive functions $\mathcal{M}$. Turing proved the equivalence of Turing machines and μ -recursive functions in 1936. Those developments do not represent a mathematical coincidence but rather a manifestation of the central role and importance of recursion and μ -recursive functions in computations and computability.

In the next section, we will explain how μ -recursive functions relate to learning.

7.1.2 Recursion in Learning

We can understand learning and neural systems in the context of μ -recursive functions if we cast or regress the functionality of such systems to the defining properties of the μ -recursive functions.

Relatively simple and often unidimensional computations that are performed locally by neurons or the units of neural systems (by the functions f_k in our definition 4.2) correspond to the properties of μ -recursive functions that enable arithmetic and calculations (those are property A for constant functions and property B for the successor function).

The inherent multidimensional nature of learning and learning systems corresponds to projection functions and thus property C.

Synthesis and the overall premise that learning systems are composed of smaller components that implement simpler functionality correspond to function composition and thus property D.

Optimization and the overall effort of learning systems to improve their performance in their environment corresponds to function minimization and

thus property F.

The remaining axiomatic property for μ -recursive functions is property E, which is for finite recursion. This is the most elaborate to explain in the context of learning.

In its simplest form, recursion may emerge by the connectivity of the graph G , if the graph contains cycles (as is the case with recurrent networks, which we will discuss later in this theory). However, recursion is considerably more fundamental than connectivity and it emerges virtually everywhere in learning and not merely in graph cycles.

For example, for each training iteration $t = 0, 1, \dots$ of a neural network, we can assume that w_t is a learned parameter of the network, w_0 is its initial value, and gradient descent is applied over a loss function ℓ_t with learning rate η_t . Then it would be

$$\Delta w = -\eta \frac{\partial \ell}{\partial w} \quad : t$$

or, equivalently,

$$w_+ = w - \eta \frac{\partial \ell}{\partial w} \quad : t.$$

However, the above defines a recursion for the value of the parameter w .

Similarly, if a neural system is not a stateless automaton that always generates the same output upon the same input and it does not merely perturb the same output by a random process, it must contain an internal state s (Y_σ in definition 4.2) that affects its output and its next state. In other words, there must be some function h_t such that

$$s = h(s_-) \quad : t.$$

However, this is another occurrence of recursion. If we add synergy and bidirectional feedback or feedback loops, there is yet another source of recursion.

Moreover, recursion permits an additional and perhaps more fundamental interpretation. It is a well-known fact in programming languages that recursive functions can be implemented through iteration and a *stack* data structure (which is often called the *run-time stack*). Hence, recursion can be implemented through iteration and memory. Conversely, it is also a well-known fact in digital circuit design that a recursive element (such as a *flip-flop*) can implement memory. In other words, and in the presence of iteration, recursion is memory and memory is recursion.

In current neural network and machine learning research, memory is not explicitly used. Most learning systems are effectively automata in their inference stage. The learning or training stage is usually an initial stage that terminates after a finite set of training iterations and the network is then a finite state automaton (if we assume finite precision arithmetic and an acyclic graph for the network, which is often the case). Recursion is absent and its importance is often underestimated.

In this theory, recursion has a unique and indispensable role not only because of the above reasons and interpretations but also because it is a primary mechanism for coherence. We will use a number of methods to manipulate coherence in learning systems but all those methods ultimately reduce to recursion.

Given that the set of μ -recursive functions $\mathcal{M}$ is universal, the association of learning and its components with the axiomatic properties of $\mathcal{M}$ is inevitable and perhaps trivial. However, it is important because it can unify learning and learning systems (when combined with coherence) and because it provides a different viewpoint and direction. For example, recursion has not attracted sufficient attention in machine learning literature while memory is practically absent (except as a side effect).

7.2 Synaptic Coherence

In this section, we examine gradient descent through coherence and recursion and propose alternative or more advanced forms. The same modifications can apply to any learning algorithm that iteratively updates the learned parameters of a system as the system processes its data.*

In the following, we assume that $t = 0, 1, \dots$ indicates the training iteration of a learning system N that uses (stochastic) gradient descent. We also assume that w is a learned parameter of N (w_t is its value during iteration t and w_0 is its initial value). If η_t is the learning rate and ℓ_t is the loss at iteration t then it would be

$$w_+ = w - \eta \frac{\partial \ell}{\partial w} \quad : t \geq 0. \quad (7.1)$$

Equivalently, we can write the above equation as

$$w = \mu w_- + \lambda g \quad : t > 0, \quad (7.2)$$

where $g = -\frac{\partial \ell_-}{\partial w_-}$, $\mu = 1$, and $\lambda = \eta_- : t > 0$. The momentum rate is thus constant and equal to 1. In general, this is not a good idea. Let us understand why.

We can easily transform the first-order recursion of equation (7.2) to its corresponding correlation (we can observe that equation (7.1) is telescopic and sum up the terms for successive values of t). We then obtain the following equation

$$w_t = w_0 + \sum_{k=1}^t \lambda_k g_k \quad (7.3)$$

This form of gradient descent has two drawbacks. It has an increased

*This is a broad class of algorithms and it includes virtually all known neural learning algorithms. There are algorithms that do not belong in this class, such as the *ID3* or *CART* algorithms for *decision tree learning*, but those are not currently part of our theory.

sensitivity to initial conditions (the initial value w_0) and it places unnecessary strain on the learning rate λ , which must remain large enough to reduce the effect of the initial value w_0 but also must decrease sufficiently fast to avoid or reduce overfitting pressure.

The above drawbacks are direct effects of the fact that the momentum rate is 1. We can easily avoid them, if we pick $\mu < 1$. In general, the first-order recursion $\mathcal{R}_1^{\lambda\mu}$ of equation (7.2) corresponds to a correlation C_{ch} as defined in equations (6.3) and (6.23) and shown below:

$$\begin{aligned} w_t &= \frac{1}{h_t} \sum_{k=0}^t c_k g_k \\ c_t &= h_t \lambda_t \\ h_t &= \begin{cases} 1 & \text{if } t = 0 \\ \frac{1}{\mu_1 \cdots \mu_t} & \text{if } t > 0, \end{cases} \end{aligned}$$

where $c_0 = 1$ and $g_0 = w_0$. (We defined the latter for simplicity since $g_0 = -\frac{\partial \ell_{-1}}{\partial w_{-1}}$ and as such it does not correspond to a training iteration.) Combining the above three equation into one, we obtain

$$w_t = w_0 \prod_{j=1}^t \mu_j + \sum_{k=1}^t \lambda_k g_k \prod_{j=k+1}^t \mu_j \quad (7.4)$$

We can now see that if $\sup\{\mu_j/j \in \mathbb{N}_+\} < 1$ then the effect of the initial value w_0 on w_t diminishes exponentially with t while the effect of g_k on w_t diminishes exponentially with the difference $t - k$. The gradient descent method defined by $\mathcal{R}_1^{\lambda\mu}$ is less sensitive to initial conditions and more ‘forgiving’ to learning rates.

However, the above form is not anything new. It is merely a rewrite of quadratic weight decay (defined in section 5.6.4) with

$$\mu_+ = 1 - \eta\gamma \quad : t \geq 0. \quad (7.5)$$

Indeed, it is straightforward to transform the weight decay formula to the corresponding $\mathcal{R}_1^{\lambda\mu}$ gradient descent formula:

$$w_+ = w - \eta \left(\frac{\partial \ell}{\partial w} + \gamma w \right) = (1 - \eta\gamma)w + \eta g_+ = \mu_+ w + \lambda_+ g_+ \quad : t \geq 0.$$

There are two main problems with weight decay. The first is that a fixed decay is usually sub-optimal. Given that the magnitude of gradients is not stationary during training, it is highly unlikely that the optimal decay rate would be constant over time. This is not an issue with gradient descent by $\mathcal{R}_1^{\lambda\mu}$ since μ is a time sequence; and it can similarly be resolved in the weight decay case if we turn γ into a time sequence, too. (In most applications, γ is

kept constant but this is not advisable.)

A related issue is that the optimal decay rate often varies across neural structures – such as neural regions, layers, partitions, and even dendrites. In other words, the optimal decay rate depends on both the training iteration and the location within a neural structure. However, we will not address the structural variance in this chapter.

The second problem with weight decay does not relate to variance. It emerges from equation (7.5) and the formula for weight decay. The decay rate γ must be relatively small to avoid overwhelming the gradient g . As a result, typical values of γ are usually in the range 10^{-6} to 10^{-3} for various learning applications. The learning rate η is also a small number to avoid large steps over regions with significantly varying derivatives. It usually starts with an initial value in the range 10^{-4} to 10^{-2} and decreases over time so that the descent process is more reliable over reduced or diminished gradients (which are bound to occur if learning is succeeding) and more sensitive and local in its progression. Consequently, the product $\eta\gamma$ is usually in the range 10^{-10} to 10^{-5} and it decreases over time. This makes the momentum rate $\mu = 1 - \eta\gamma$ to be very close to one. Given that $(1 - 1/n)^n \approx 1/e$ for large n , the above numbers and theorem 6.3 imply that the effects of previous gradients on the coherence of parameters may remain significant for 10^5 to 10^{10} or more training iterations. Such long dependencies are not desirable and may lead to excessive coherence.

At first sight, the above problem may seem hard or even intractable. We cannot reduce the momentum rate μ to limit coherence without increasing the learning rate η or the decay rate γ . However, learning is usually sensitive to the latter rates while the effects of momentum rates are multiplicative, as shown in theorem 6.3. As a result, there can only be a narrow viable range in the middle, if at all.

Empirical approaches are usually used to determine such a combination of learning rates, rate schedules, and weight decay that result in lower generalization error over a validation set. However, such approaches are effectively trapped within their own limitations. They can potentially find an optimal combination in their search space but their search space is overly and unnecessarily constrained.

The next step is to change or eliminate some of the constraints and expand the search space so that a potentially intractable or over-constrained problem can admit better solutions.

The key for the expansion is recursion. The above limitations emerge from the fact that weight decay, gradient descent, and $\mathcal{R}_1^{\lambda\mu}$ are all first-order recursion. If we use a higher-order recursion $\mathcal{R}_k^{\lambda\mu}$, we can bypass the above limitations because the momentum rates μ will operate on a different order and they will not collide with decay or descent. The only advantage of trying to squeeze all those rates into a single order is reduced memory requirements. However, there are better solutions for reducing memory requirements as we

shall see in this chapter and later in the theory. Limiting order should be deemed a severe measure; and using only first order is the most severe among such measures.

If we naively define $\mathcal{R}_k^{\lambda\mu}$ for synaptic coherence, we would need to retain the k previous values of each parameter w in the network. In most cases, this would be excessive and unnecessary. In general, we do not need to retain the exact progression from the past to the present but rather a long-term, a short-term, and a present representation of the instances of the progression. Such a memory or recursion scheme can have arbitrary order but it only requires us to store one or two additional values per parameter.*

We understand that the above requirements can still be excessive when applied at a synaptic level and thus on every synapse of a biological nervous system. We will not address this issue in this section but we will present more economical approximate solutions as we progress in our theory.

There are multiple ways to define or implement the above scheme. It is an instance of *synaptic coherence*, which refers to a set of methods that affect the coherence of the learned parameters that are associated with the connections of a learning system (as opposed to those that are associated with the units or other parts of the system). There are multiple synaptic coherence methods. Our definition below is not unique or optimal. It is rather a first and simple approach. In order to make some formulas shorter, we use a *congruence subscript notation* $a \equiv_n b$ to indicate that $a \equiv b \pmod n$.

Definition 7.3 (Parametric coherence). Let N be a network trained with loss ℓ and w be a learned parameter of N with initial value w_0 (w_t is the value of w during training iteration $t \geq 0$). We assume a learning algorithm that produces an update Δw for w at each iteration. (If the learning algorithm is gradient descent then it would be $\Delta w = -\eta \frac{\partial \ell}{\partial w} : t \geq 0$, where η_t indicates the learning rate during iteration t .) We also assume time sequences μ and λ that represent momentum and learning rates for coherence, respectively. We require that $\mu_0 = 0$ and $\lambda_0 = 1$. *Parametric coherence* then updates w as follows

$$\left. \begin{aligned} \tilde{w}_+ &= \begin{cases} \mu_+ \tilde{w} + \lambda_+ w + \Delta w & \text{if } [t \equiv_h 0] \\ \tilde{w} & \text{otherwise} \end{cases} \\ w_+ &= \begin{cases} \tilde{w}_+ & \text{if } [t \equiv_d 0] \\ w + \Delta w & \text{otherwise} \end{cases} \end{aligned} \right\} : t \geq 0, \quad (7.6)$$

where d and h are positive integers called *drift* and *hysteresis* (or *lag*), respectively. Initially, it is $\tilde{w}_0 = w_0$.

*The distinction between long-term and short-term memory is common and established in neuroscience. It is not entirely clear why memory is processed in such a manner in biological systems but a direct mathematical consequence is that it disassociates recursion order from memory and allows for higher or arbitrary order to be implemented with less memory.

Intuitively, parametric coherence updates each learned parameter w using its update Δw but also maintains a coherent version $\tilde{w}$ of w that is defined through an h -order affine recursion. Then, every d training steps, it replaces w with its coherent version $\tilde{w}$.

Of course, when the parameter w is associated with a connection in N , parametric coherence is synaptic coherence. Given that most learned parameters in a network are associated with synapses, we will use the terms synaptic and parametric coherence interchangeably and disambiguate the usage when it is not clear from the context.

The term hysteresis derives from the Hellenic word ὑστέρησις for “short-coming”, “lateness”, or “lagging behind”. In our case, it refers to the order of the recursion h and thus the fact that $\tilde{w}$, when it changes value, it depends on its value from the h -th previous training iteration. (Hysteresis implies that the state of a system depends on its history; and it often refers to a property of the system that lags behind associated changes or effects.) Furthermore, $\tilde{w}$ is not only hysteretic but also constant within each hysteresis interval (from iteration $ih + 1$ to iteration $ih + h$ for $i \in \mathbb{N}$). As a result, it can be computed through first-order recursion, which avoids the increased memory requirements of higher order. This is shown in equation (7.6).

The drift d refers to the interval that is used to replace the parameter w with its coherent estimate $\tilde{w}$. Intuitively, the parameter w is allowed to “drift” away from its coherent value for d training iterations; during its drift, it affects the coherent estimate (once for each hysteresis interval within the drift); but at the end of each drift, it is forced to “return” to coherence. The parameter is thus like a scout or a probe that explores regions of the search space without any coherence limitations but then it is pulled back to its coherent estimate, which explores the space in a more measured and conservative manner.

Consequently, when the network N is trained with parametric coherence and a gradient method, such as gradient descent, and it is near a narrow local minimum, its parameters may move toward the minimum following their gradients but their coherent estimates are less likely to follow – and if they do not, they will pull the parameters out of the minimum. Conversely, the coherent estimates can be pulled into the minimum only when the “exploring” parameters persistently return to the minimum. Given that narrow minima are less likely to have such a persistent effect, parametric coherence effectively skips them and leads to minima that are broad enough to capture the parameters persistently.

In machine learning and neural network literature, it has been claimed that broad local minima represent solutions that tend to generalize better. Such performance is also a consequence of our axiomatic premise that generalization is primarily determined by coherence. Indeed, narrow minima have lower coherence because relatively smaller changes in the parameters can have more pronounced effects on the loss or even move the network out of the minimum’s basin of attraction.

We can understand this functionality better if we draw an analogy with porous surfaces. If the loss function has such a geometry, we would like to find a broad minimum of the surface instead of being trapped inside a pore. We can initially use large learning rates to reduce the probability of being trapped in a pore but the learning rates should eventually be reduced or diminished if we wish to minimize the generalization error. The learning process then becomes vulnerable to the porous geometry of the loss. Parametric coherence adds an important layer of protection against such vulnerability and allows learning processes to operate with a considerably larger range of learning rates.

In general, the drift d should be larger than the hysteresis h . The hysteresis represents short-term memory and it is a relatively short-term effect; but the drift represents a longer-term effect. (There is no long-term memory in this algorithm, only a long-term effect.)

The case where $d = h$ should thus be deemed as extreme. Weight decay is a special case of synaptic coherence on gradient descent with $d = h = 1$ and $\lambda_+ + \mu_+ = 1 - \eta\gamma : t$. (Indeed, when gradient descent is used and $d = h = 1$, it is $w = \tilde{w}$ (by induction) and equation (7.6) reduces to $w_+ = \mu_+ w + \lambda_+ w + \Delta w = (\mu_+ + \lambda_+)w - \eta \frac{\partial \ell}{\partial w} : t$.)

In other words, weight decay is not merely a short-term method but an immediate coherence method. It is thus likely to lead to excess if the decay rate is not very small.

In order to understand parametric coherence more completely, let us now derive the corresponding correlation formula for it.

Theorem 7.1. *Parametric coherence, as defined in equation (7.6), implies that*

$$w_{t+1} = \begin{cases} \frac{1}{s_q} \left(w_0 + \sum_{i=0}^q (\lambda_{ih+1} w_{ih} + \Delta w_{ih}) s_i \right) & \text{if } t \equiv_d 0 \\ w_t + \Delta w_t & \text{otherwise} \end{cases} \quad (7.7)$$

where $q = \lfloor t/h \rfloor$ and $s_i = 1/(\mu_1 \mu_{h+1} \cdots \mu_{ih+1})$ for $i \in \mathbb{N}$.

Proof. Let us first derive a correlation formula for the sequence $\tilde{w}$. If we define $a_k = \tilde{w}_{kh+1} s_k$ and $b_k = \lambda_{kh+1} w_{kh} + \Delta w_{kh}$, for $k \in \mathbb{N}$, then it is

$$\begin{aligned} a_k &= \tilde{w}_{kh+1} s_k \\ &= (\mu_{kh+1} \tilde{w}_{kh} + \lambda_{kh+1} w_{kh} + \Delta w_{kh}) s_k && \text{(by (7.6))} \\ &= \tilde{w}_{kh} s_{k-1} + b_k s_k && \text{(by the definition of } s_k \text{ and } b_k) \\ &= \tilde{w}_{(k-1)h+1} s_{k-1} + b_k s_k && \text{(by (7.6) repeated } h-1 \text{ times)} \\ &= a_{k-1} + b_k s_k && \text{(by the definition of } a_k) \end{aligned}$$

Summing up the above telescoping sequence and eliminating common terms,

we obtain

$$\begin{aligned}
 a_k &= a_0 + \sum_{i=1}^k b_i s_i \\
 &= (\mu_1 \tilde{w}_0 + \lambda_1 w_0 + \Delta w_0) s_0 + \sum_{i=1}^k (\lambda_{ih+1} w_{ih} + \Delta w_{ih}) s_i \\
 &= w_0 + \sum_{i=0}^k (\lambda_{ih+1} w_{ih} + \Delta w_{ih}) s_i
 \end{aligned}$$

or, equivalently,

$$\tilde{w}_{kh+1} = \frac{1}{s_k} \left(w_0 + \sum_{i=0}^k (\lambda_{ih+1} w_{ih} + \Delta w_{ih}) s_i \right) \quad (7.8)$$

Now, if $t \not\equiv_d 0$, equations (7.6) and (7.7) identically define $w_+ = w + \Delta w : t$. Therefore the only remaining case is when $t \equiv_d 0$. However, in that case, equation (7.6) defines $w_+ = \tilde{w}_+ : t$. Furthermore, the definition of q implies that there exists $r \in \mathbb{N}_h$ such that $t = qh + r$. Therefore

$$w_{t+1} = \tilde{w}_{t+1} = \tilde{w}_{qh+r+1} = \tilde{w}_{qh+1} \quad (7.9)$$

since $\tilde{w}$ is constant in the hysteresis interval $[qh + 1, qh + h]$. Combining equations (7.8) and (7.9) completes the proof. $\square$

The above theorem shows that the value of the parameter w and its update Δw at iterations that are an integer multiple of the hysteresis h have a direct effect on w_{t+1} when t is an integer multiple of the drift d . In all other cases, it is $w_{t+1} = w_t + \Delta w_t$, which means that w_{t+1} is directly affected only by its previous value and its update. In other words, the values and updates of w at iterations $t = ih$, for $i \in \mathbb{N}$, have a more direct effect on future values of the parameter. In a sense, they correspond to learning events that are given more weight.

It is highly unlikely that ‘important’ events in the learning history of a parameter occur periodically and with fixed period h . Furthermore, a large hysteresis is probably a risky choice during periods of large or rapid changes. Similarly, a small hysteresis is a waste of energy and neural resources during periods of small changes and relative stability. Given that a learning network changes more rapidly during its early training iterations, it is advisable that the hysteresis is initially smaller and tends to increase over time.

Similar arguments can apply to the drift albeit the drift determines how often the network uses its ‘prior states’ (or rather a coherent representation of those states) to affect the current one. As such it requires a different treatment.

Those observations will eventually lead to synaptic coherence with adaptive hysteresis and drift later in our theory. For the remaining of this chapter, we assume fixed and predefined hysteresis and drift. In applications, we will almost always define the drift to be an integer multiple of the hysteresis. (This is not necessarily optimal but it simplifies the interactions of gradients over time; and the empirical evidence seems to support it so far.)

Of course, the increased power and flexibility of synaptic/parametric coherence has a cost. For each parameter w , we need to store its coherent estimate $\tilde{w}$. This effectively doubles the memory requirements of a learning model, if we assume that the learned parameters of the model represent the bulk of its requirements.

The momentum rate μ and learning rate λ of coherence must also be computed at each iteration and possibly stored if their computation is recursive. However, momentum and learning rates are often shared across neurons, layers or parts of a neural partition, and even entire networks. Their overhead is thus trivial compared to the overhead of the coherent estimates $\tilde{w}$.

7.3 Synaptic Gradient Coherence

Synaptic gradient coherence refers to a set of methods that affect the coherence of the gradients of the loss function with respect to the learned parameters of a learning system that uses those gradients for learning purposes. Gradient descent is the most common learning method that uses gradients in such a manner.

In the following subsections, we will describe various forms of synaptic gradient coherence starting with simple or existing forms and advancing to more complex ones.

7.3.1 Momentum Gradient Descent

Update momentum or simply *momentum gradient descent* is a method that applies first-order affine recursion on the update Δw of each learned parameter w before using it to update w . It is defined as

$$\left. \begin{aligned} \Delta w &= -\eta g && \text{if } \llbracket t = 0 \rrbracket \\ \Delta w &= \mu \Delta w_- - \eta g && \text{if } \llbracket t > 0 \rrbracket \\ w_+ &= w + \Delta w \end{aligned} \right\} : t \geq 0 \quad (7.10)$$

where $g = \frac{\partial \ell}{\partial w} : t$ and ℓ is the loss used to train the system. The above implies that $\Delta w = \mathcal{R}_1^{1\mu}(-\eta g)$. Hence, we can rewrite update momentum in a more concise form:

$$w_+ = w + (\mathcal{R}_1^{1\mu}(-\eta g)) : t \geq 0. \quad (7.11)$$

The above equation suggests a general method that we can use to generate coherent learning methods. For any given learning algorithm, we can make

its update rule more coherent by applying a coherence operator, such as $\mathcal{R}_k^{\lambda\mu}$, to one or more of its terms or factors. For example, gradient descent, which uses the update rule $w_+ = w - \eta g$, admits various coherent forms defined by the update rules

$$w_+ = w - (\mathcal{R}_k^{\lambda\mu}(\eta g)) \quad : t \geq 0$$

(or the update rules $w_+ = w - \eta(\mathcal{R}_k^{\lambda\mu} g)$, as we shall see shortly) for different choices of the sequences λ and μ and the recursion order k . Of course, the partial products of the momentum sequence μ should be less than one to guarantee an increase in coherence as theorem 6.3 implies. This is usually achieved by limiting μ in the interval $[0, 1]$.

Historically, momentum methods were introduced in the 1960s as a way to speed up the convergence of iterative methods on convex optimization. However, we consider their effect on coherence to be more significant than any speed-up that they may generate. An emphasis on convergence rates in neural network literature and particularly in publications of learning algorithms, which sometimes include theorems that provide bounds or guarantees of convergence under certain conditions, represents an almost irrelevant crossover from optimization and the misguided attempt to reduce learning to optimization.

Convergence speed is an easier topic than generalization. It admits mathematical treatment that often produces theorems and publishable analyses. Yet, such results have limited theoretical and practical value.

The challenging goal of gradient-based learning is not to descend a convex surface around a local minimum in the fastest possible way but rather to escape training minima that are likely to generalize poorly. Unless a proposed learning algorithm requires dramatically greater resources or results in excessive learning deceleration, coherence and generalization are the only properties that truly matter. They are not as amenable to simple and brief mathematical approaches, the underlying problems are considerably harder, but as we argued in section 2.6 and later in section 5.3 provability should not take precedence over complexity.

We have little interest in convergence rates in this theory. When we actually use the term convergence, we do not refer to convergence to the minimum of a convex region of a training set or its transformations. Instead, we consider proximity to a coherent and well-generalizing solution that does not necessarily correspond to such a minimum. We would thus use the term *optimization convergence* to refer to the former and *coherence convergence* or simply *convergence* for the latter.

Momentum gradient descent is important not because it can affect optimization convergence but because it affects coherence. It is actually a second-order recursion on the learning parameters. Indeed, equation (7.10) implies that $\Delta w = w_+ - w$ and thus $w_+ - w = \mu(w - w_-) - \eta g$. Or, equivalently

$$w_+ = (1 + \mu)w - \mu w_- - \eta g$$

$$= (\mathcal{R}_2^{1(1+\mu, -\mu)}(-\eta g)) \quad : t \geq 1$$

with w_0 being arbitrary and $w_1 = w_0 - \eta_0 g_0$.

Alternatively, applying lemma 6.1 to equation (7.10), we can derive a correlation form for the update Δw

$$\Delta w_t = (\mathcal{R}_1^{1\mu}(-\eta g))_t = -\frac{1}{s_t} \sum_{k=0}^t s_k \eta_k g_k \quad (7.12)$$

for $t \in \mathbb{N}$. (s_t is again the inverted product of the momentum rates: $s_0 = 1$ and $s_t = 1/\prod_{j=1}^t \mu_j$ for $t > 0$.) Combining (7.10) and (7.12), we obtain

$$w_t = w_0 + \sum_{i=0}^{t-1} \Delta w_i = w_0 - \sum_{i=0}^{t-1} \frac{1}{s_i} \sum_{k=0}^i s_k \eta_k g_k$$

and by rearranging terms

$$w_t = w_0 - \sum_{k=0}^{t-1} s_k \eta_k g_k \sum_{i=k}^{t-1} \frac{1}{s_i} \quad (7.13)$$

If we now compare the above equation with the corresponding form for gradient descent (equation (7.3)), we can see that the sum of the partial products of the momentum rates $\sum_{i=k}^{t-1} s_k/s_i = \sum_{i=k}^{t-1} \prod_{j=k+1}^{t-1} \mu_j$ ‘undertake’ some of the coherence and convergence burden from the gradient learning rates η_k but the sensitivity to initial conditions (w_0) is still the same. This is actually one of the reasons that momentum is often combined with weight decay in applications.

An alternative form of momentum, which we call *gradient momentum*, operates on the gradients themselves instead of the updates Δw .

$$\left. \begin{aligned} v &= g && \text{if } \llbracket t = 0 \rrbracket \\ v &= \mu v_- + g && \text{if } \llbracket t > 0 \rrbracket \\ w_+ &= w - \eta v \end{aligned} \right\} : t \geq 0 \quad (7.14)$$

Or, more concisely,

$$w_+ = w - \eta \mathcal{R}_1^{1\mu}(g) : t \geq 0. \quad (7.15)$$

Lemma 6.1 implies

$$v_t = (\mathcal{R}_1^{1\mu} g)_t = \frac{1}{s_t} \sum_{k=0}^{t-1} s_k g_k. \quad (7.16)$$

Combining the last two equations, we obtain the correlation form of gradient momentum

$$w_t = w_0 - \sum_{k=0}^{t-1} s_k g_k \sum_{i=k}^{t-1} \frac{\eta_i}{s_i}. \quad (7.17)$$

Comparing (7.13) and (7.17), we can see that the contribution of the gradient g_k to w_t is $s_k \eta_k g_k \sum_{i=k}^{t-1} \frac{1}{s_i}$ in the first case and $s_k g_k \sum_{i=k}^{t-1} \frac{\eta_i}{s_i}$ in the second. In other words, the contribution depends only on the value η_k of the gradient learning rate at iteration k in the first case while it depends on the progression of the rate η from iteration k to $t - 1$ in the second case. We consider the latter to be more robust and coherent but we did not observe a significant difference in generalization between the two methods in a small number of experiments we performed. Mathematically, the two methods are identical iff η is constant.

For the rest of this chapter, we have opted for coherence of the gradient g instead of the update Δw because the gradient learning rates are not always constant and because we would like to separate the gradients from their learning rates for coherence purposes.

An important factor in momentum methods is their bias. Theorem 6.4 and the fact that $\lambda = 1 \neq 1 - \mu$ imply that both update and gradient momentum are biased sequence transformations. In the case of update momentum, we can rewrite the recursion as $\Delta w = \mu \Delta w_- + (1 - \mu)(-\frac{\eta}{1-\mu} g) : t > 0$. If we define $M(\mu, \eta)$ and $M_u(\mu, \eta)$ to be update momentum and unbiased update momentum, respectively, with momentum rate μ and gradient learning η , it is

$$M(\mu, \eta) = M_u\left(\mu, \frac{\eta}{1 - \mu}\right)$$

In other words, update momentum with gradient learning rate η is the same as unbiased update momentum with learning rate $\eta/(1 - \mu)$.

Consequently, when we compare update momentum with plain gradient descent, it is not statistically valid to compare the two methods using the same learning rate η . Our first attempt should be to multiply the learning rate of gradient descent with $1/(1 - \mu)$ since update momentum effectively trains with that rate. However, momentum increases coherence and thus makes the learning process more tolerant of greater learning rates. It is thus possible that gradient descent may diverge or perform poorly with such an increased rate. A more valid method would be to search for a good learning rate in the interval $[\eta, \eta/(1 - \mu)]$ for gradient descent and use that rate for comparison purposes.

A final note about the momentum methods of this section is that they require us to store the update Δw (or the gradient g or the value w_- if their second-order form is used). They effectively double the memory requirements of models the same way that synaptic coherence does.

7.3.2 Synaptic Cross-Gradient Coherence

In this section, we will use coherence to create more elaborate versions of gradient descent. One obvious improvement is to apply the operator $\mathcal{R}_1^{\lambda\mu}$ to gradients instead of the operator $\mathcal{R}_1^{1\mu}$ that momentum methods typically use. If we define $a = \mathcal{R}_1^{\lambda\mu} g$ then a would be a coherent version of the gradient

sequence g . However, this does not imply that the coherent gradients a across different parameters w or different neurons are coherent or comparable with each other. We call the latter *synaptic cross-gradient coherence*.

In general, cross-gradient coherence is as important as it is computationally demanding. It is often more useful and manageable locally in smaller parts of a network that share functionality and learn and adapt more as a group; but it also has utility across such groups. In the latter case, it may be treated differently across various levels of a complex neural system.

Overall, cross-gradient coherence is a topic that admits better solutions than the one we offer in this volume. We will discuss it in greater detail in volume II and propose alternative forms. In this section, we will use normalization, which is a coarse method to enforce cross-gradient coherence across the entire network. It cannot compete with more refined methods but it is simple and computationally efficient and it constitutes a natural starting point.

Given that gradients usually vary across different parameters, one obvious way to make them comparable and increase their proximity (and first-order coherence) is to normalize them by dividing them with their absolute value or some other norm. However, the ratio $g/|g|$ will always be ± 1 when g is a non-zero real number. This most likely represents an overly constrained form when g can be arbitrary. It is probably better to apply individual coherent operators to the gradient g and its norm $|g|$ and divide the coherent estimates instead of applying a coherent operator to their original ratio. (The situation is similar to the ratio-of-means estimator R_2 when compared to the mean-of-ratios estimator R_1 .) Thus we can define $b = \mathcal{R}_1^{\nu\xi}|g|$ and use the ratio $a/(b + \epsilon)$ as a coherent estimate of $g/|g|$. The constant ϵ in the denominator is added to avoid numerical instability when the gradient g and its coherent estimate b diminish.

Furthermore, we would like to be able to use any norm instead of the absolute value. For p -norm the extension is straightforward. We can define $b = \mathcal{R}_1^{\nu\xi}|g|^p$ and use the ratio $a/(b^{1/p} + \epsilon)$. This leads to the update rule

$$w_+ = w - \eta \frac{(\mathcal{R}_1^{\lambda\mu} g)}{(\mathcal{R}_1^{\nu\xi} |g|^p)^{1/p} + \epsilon} : t|\epsilon, p.$$

The corresponding algorithm is shown in figure 7.1.

The algorithm assumes that a_{-1} and b_{-1} are valid expressions, when $t = 0$, but their values do not matter because both μ_0 and ξ_0 are set to 0. (In other words, we do not have to provide initial values for a_{-1} and b_{-1} .)

Synaptic cross-gradient coherence does not specify the momentum and learning sequences λ , μ , ν , and ξ . In most cases, we use unbiased coherent estimates and thus pick $\lambda = 1 - \mu$ and $\nu = 1 - \xi$. In our experiments, we only used sequences from table 6.1. Those represent a wide range and different functional forms of growth rates and they should probably suffice for most

$$\text{Synaptic Cross-Gradient Coherence}(w_0, \ell, \eta, p, \lambda, \mu, \nu, \xi, \epsilon)$$

$$\left. \begin{aligned} \lambda_0 &= \nu_0 = 1 \\ \mu_0 &= \xi_0 = 0 \\ \text{for } t &= 0, 1, \dots \\ g &= \frac{\partial \ell}{\partial w} \\ a &= \mu a_- + \lambda g \\ b &= \xi b_- + \nu |g|^p \\ w_+ &= w - \eta a / (b^{1/p} + \epsilon) \end{aligned} \right\} : t | \epsilon, p$$

Figure 7.1: Synaptic cross-gradient coherence.

applications.

The choice of the above sequences can significantly affect the overall learning algorithm. Many known learning algorithms are special cases of synaptic cross-gradient coherence. For example, the *Adagrad* learning algorithm is equivalent to synaptic cross-gradient coherence with $p = 2$, $\lambda_t = 1 - \mu_t = 1$, and $\nu_t = 1 - \xi_t = 1/(t+1)$. Similarly, the *Adam* learning algorithm is synaptic cross-gradient coherence with $p = 2$, $\lambda_t = 1 - \mu_t = (1 - \beta_1)/(1 - \beta_1^{t+1})$, and $\nu_t = 1 - \xi_t = (1 - \beta_2)/(1 - \beta_2^{t+1})$, where β_1 and β_2 are momentum parameters of Adam.

We should note that the sequences λ and ξ for Adam correspond to unbiased exponential smoothing with $\gamma = \beta_1$ and $\gamma = \beta_2$, respectively (see definition 6.10). In other words, Adam uses unbiased exponential smoothing.

The authors of Adam actually started with exponential moving average but avoided its bias by initializing the moving averages to zero. This is a counterintuitive and biased initialization because it suggests that learning starts at a local minimum. It thus trades one bias for another. The authors then observed that their initialization introduced a bias, and they assumed stationarity to estimate and factor it out. The end result could only be the unbiased exponential smoothing method because it is the only affine transformation that has a constant growth rate and it is unbiased. In a sense, the broader bias of exponential moving average passed unnoticed under a biased (and perhaps unreasonable) initialization.

7.4 The Syc Algorithm

Syc (pronounced “psych”) is a simple algorithm that combines synaptic coherence and cross-gradient coherence so that both the learned parameters and their gradients are coherent. It permits weight decay (which is similar to an additional and unitary hysteresis) and makes the gradient normalization optional. It also uses cumulative correlations so that its coherent estimates are unbiased. The algorithm is shown in figure 7.2. Default or recommend values are shown next to each input argument.

Syc (w_0 ,	initial value
ℓ ,	loss function
$\eta = 10^{-3}$,	gradient learning rate
$\gamma_t = 0$,	weight decay rate
$d = 100$,	drift
$h = 10$,	hysteresis
$\theta_t = 0.9 \frac{t}{t+1}$,	synaptic coherence momentum
$\mu_t = 0.9 \frac{t}{t+1}$,	gradient coherence momentum
$\xi_t = 0.999 \frac{t}{t+1}$,	gradient norm momentum
$p = 2$,	p -norm for cross-gradient coherence
$q = 2$,	q -norm for weight decay
$\epsilon = 10^{-8}$	numerical stability term
)	

$\mu_0 = \xi_0 = 0$	
$\tilde{w}_0 = w_0$	
if ($p = 0$) $b_{-1} = \epsilon$	
for $t = 0, 1, \dots$	
$g = \frac{\partial \ell}{\partial w} + \begin{cases} \gamma w w ^{q-2} & \text{if } \gamma > 0 \\ 0 & \text{otherwise} \end{cases}$	} : $t p, q, d, h, \epsilon, \hat{w}$
$a = \mu a_- + (1 - \mu)g$	
if $p > 0$	
$b = \xi b_- + (1 - \xi) g ^p$	
$\hat{w} = w - \eta a / (b^{1/p} + \epsilon)$	
else if $p = 0$	
$b = \max(b_-, g)$	
$\hat{w} = w - \eta a / b$	
else $\hat{w} = w - \eta a$	
$\tilde{w}_+ = \begin{cases} \theta_+(\tilde{w} - w) + \hat{w} & \text{if } t \equiv_h 0 \\ \tilde{w} & \text{otherwise} \end{cases}$	
$w_+ = \begin{cases} \tilde{w}_+ & \text{if } t \equiv_d 0 \\ \hat{w} & \text{otherwise} \end{cases}$	

Figure 7.2: The Syc algorithm.

On the first part of the loop, the gradient g is computed and it is combined with q -norm weight decay when the decay rate is positive. (Negative or zero rate would thus suppress weight decay.) Then the coherent estimates a and b of the gradient and its p -norm are computed along with the intended next value $\hat{w}$ of w . (Note that $p = 0$ is reserved for the infinity norm, which is $L_\infty = \max$.) Finally, synaptic coherence is computed in the last part of the loop, using the fact that $\theta_+ \tilde{w} + (1 - \theta_+)w + \Delta w = \theta_+(\tilde{w} - w) + \hat{w} : t$.

Note that we need to initialize $b_{-1} = \epsilon$ when $p = 0$ because the instruction

$b_0 = \max(b_{-1}, |g_0|)$ will be executed at iteration $t = 0$. In this case, the chosen initial value guarantees that the coherent estimate of the gradient infinity norm will be greater or equal to ϵ , resulting in increased numerical stability. However, such initialization is not required for $p > 0$ because $\mu_0 = \xi_0 = 0$.

In the actual implementation of Syc, we also use a parameter $s \geq 1$ which defines the number of steps between successive updates of the momentum rates. In other words, the momentum rates $\theta_{\lfloor t/s \rfloor}$, $\mu_{\lfloor t/s \rfloor}$, and $\xi_{\lfloor t/s \rfloor}$ are used at each iteration t .

7.5 Neural Coherence

An obvious disadvantage of synaptic coherence is its memory requirements. For example, for each learned parameter w , Syc requires us to store the coherent estimate a of the gradient, the coherent estimate b of the p -norm of the gradient (raised to power p), and the coherent estimate $\tilde{w}$ of w . This increases the size of the network by about 300%.* The same applies to other coherent learning methods. For example, Adam and update momentum have a memory overhead of 200% and 100%, respectively.

The above overhead has largely been overlooked in neural network literature. We consider that to be an effect of the great inefficiency of von Neumann architectures, the inefficient memory management of current neural network frameworks, and the artificial way of training neural systems with batches of data that are often larger in size than the network. However, if we intend to create a theory that can scale up to the magnitude of biological systems, or if we take into account the evolutionary pressure for reduced energy consumption, we can not ignore such overhead.

As a result, it will become increasingly important to derive more economical forms or approximations of synaptic coherence as our networks grow in size.

In general, the number of neurons in a neural network is much smaller than the number of their synapses.[†] For example, in the human brain, each neuron has about 10,000 connections on average. There are thus four orders of magnitude more synapses than there are neurons. Consequently, if we double the parameters of each synapse, we will effectively double the size of the brain; but if we double the number of parameters in each neuron, or even

*We assume that the learning parameters of the network represent the bulk of its memory requirements and that the momentum rates are shared across multiple units resulting in negligible overhead.

[†]This is not always true with convolutional neural networks because they share their kernel parameters across each convolution layer. The number of learned parameters can thus be less than the number of units in the underlying network. However, as we explained in section 1.5.3, CNNs are a limited and ephemeral construct that does not belong to complex neural systems. Even if we ignore their other issues, sharing parameters across many neurons is a thermodynamic liability that can only survive as long as we use equally inefficient architectures.

if we increase them 100-fold, we will only increase the memory requirements by $1/10000 = 0.01\%$ and $100/10000 = 1.0\%$, respectively.

The above estimates suggest that we should either approximate synaptic coherence using fewer parameters or ‘migrate’ most of coherence from individual synapses to groups of synapses (such as the dendrites of a neuron), individual neurons, or the parts of a neural partition. We will use the terms *dendritic* and *neural coherence* to refer to cases of such coherence. We will also use multiple terms, such as *cluster* or *nuclear coherence*, to refer to coherence of groups of neurons depending on their structure, connectivity or topology, and function.

We will elaborate on those types of coherence later in the theory. In this chapter, we only present a form of neural coherence that approximates synaptic cross-gradient coherence for gradient descent.

Mathematically, if we apply the above arguments to our neural network definition 4.2, we can see that the bulk of learned parameters of a network are the parameters ψp_k of the polynomial functions p_k for each layer k . The functions f_k usually have zero learned parameters or a small number of such parameters per unit. The total number is thus at most linear with the number of units in the network. Similarly, the aggregating functions are often mere concatenations or summations of inputs with no learning parameters; and when they do have learning parameters, their number is usually at most linear with the aggregated inputs and thus at most linear with the number of units in one or more layers.

State and cross-connections increase the number of inputs and they may associate more than one parameter per unit. However, those parameters are actually part of the polynomial functions since both state and cross-connection inputs are passed to the polynomials functions through the corresponding aggregating functions.

Consequently, if we wish to derive a memory efficient representation of synaptic coherence, we must focus on the parameters ψp_k of each layer L_k .

We will provide such an algorithm shortly.

7.6 Factorized Neural Coherence

From a theoretical viewpoint, the high memory overhead of synaptic coherence stems from the fact that synaptic properties are associated with synapses or connections in a network. Those are edges of an underlying graph and their number can thus grow quadratically with the number of nodes in the graph.

Actual neural networks do not correspond to such quadratic graphs. They are sparse, with each unit or neuron connecting to a small minority of the total units in the network. Yet, the number of connections is considerably larger than the number of units because each unit can have multiple connections (and it usually has many).

Consequently, if $G = (V, E)$ is a graph with a set of vertices V and a set

of edges E , any coherence method that operates directly on the edges and stores information for each edge is bound to have memory requirements that are linear with the number of edges $|E|$.

On the other hand, we can reduce the above memory overhead and make it linear with the number of vertices $|V|$, if we associate a coherence estimate c_v with each node $v \in V$ and define the coherence of each edge $(u, v) \in E$ as function of the coherence of its nodes

$$c_{u,v} = f_u(c_u, c_v) \diamond g_v(c_u, c_v),$$

where f_u and g_v are functions that may depend on the units u and v , respectively, and $\diamond$ is a generic mathematical operation that does not depend on the units (such as summation or product).

The above scheme is effectively a *factorization method* because it ‘factors’ the computation of $c_{u,v}$ into the computations of c_u and c_v . The latter as well as the functions f_u and g_v may require the storage of a few parameters per unit but if the number of such parameters is bounded by a (small) constant, the memory requirements will be linear with the number of nodes $|V|$ in the graph. Assuming that each unit $v \in V$ has significantly fewer coherence parameters than predecessors in G (and thus incoming connections), the memory overhead of factorized coherence will be a small and rather insignificant fraction of the size of the network.

We use the terms *factorized neural coherence* or *neural coherence factorization* to refer to such algorithms or approaches for coherence. *Sync* (acronymically derived from the words “synaptic”, “neural”, and “coherence”) is a neural coherence algorithm that generates an approximation of synaptic cross-gradient coherence through coherence factorization in order to reduce the overall memory overhead. The algorithm is similar in structure to Syc and is described in more detail in the next section.

7.7 The Sync Algorithm

Let us derive a cross-gradient coherence factorization method for the parameters of the polynomial functions of a neural network.

We assume a neural network $N = (G, F, Y, S)$ as in definition 4.2. We also assume a unit $v \in L_k$, which means that v is a unit of layer L_k . If P_k is the output of the polynomial function p_k of L_k then $P_{k,v}$ is the element of P_k that corresponds to v . We will write P_v to indicate $P_{k,v}$. (In other words, we assume v to be a global index into the elements of N instead of a local index into the elements of L_k .) Assuming that $G = (V, E)$, if $u \in V$ is a unit that provides input to v , we use the notation $\psi_{u,v}$ to indicate the learned parameters of p_k that are associated with the edge $(u, v) \in E$. (This is equivalent to $(\psi p_k)_{u,v}$ if we assume that the units provide local indexing into the layers.)

Using corollary 2.1, we obtain a formula for the gradient of $\psi_{u,v}$

$$\frac{\partial \ell}{\partial \psi_{u,v}} = \frac{\partial \ell}{\partial Y_k} f'_k(P_k) \frac{\partial P_k}{\partial \psi_{u,v}}.$$

If we assume affine polynomial functions and unidimensionality in the non-polynomial functions* then the above equation reduces to

$$\frac{\partial \ell}{\partial \psi_{u,v}} = \frac{\partial \ell}{\partial Y_v} f'_v(P_v) Y_u. \quad (7.18)$$

Equation (7.18) represents a factorization of the gradient into the factor $\frac{\partial \ell}{\partial Y_v} f'_v(P_v) = \frac{\partial \ell}{\partial P_v}$ that depends only on v and the factor Y_u that depends only on u . Consequently, we can individually apply coherence to those two factors to generate a coherent form of the gradient.

However, the second factor is the output of the unit u . If we apply a coherence operator to it over time, we will be using outputs that correspond to different inputs that can be entirely unrelated. This is not necessarily incorrect since we will be using those estimates to affect the gradients only; but it may introduce an undesired amount of coherence.

A simple approximation is to apply coherence only to $\frac{\partial \ell}{\partial P_v}$ and not to Y_u . A probably more accurate solution is to use mean values to try to approximate the product $\frac{\partial \ell}{\partial P_v} Y_u$. We will first present the simpler algorithm and then generate its more complex version that uses mean values.

The outputs of the units of the network and the gradients of their parameters represent time sequences because they change at each training iteration t . In order to make the formulas more compact under split notation, we separate unit indexing from time indexing and use the former as an argument. Hence, we define $y(v)_t = (Y_v)_t$ and $\delta(v)_t = (\partial \ell / \partial P_v)_t$.

Assuming sequences λ , μ , ν , and ξ with $\mu_0 = \xi_0 = 0$ and $\lambda_0 = \nu_0 = 1$, we define

$$\left. \begin{aligned} \alpha &= \mathcal{R}_1^{\lambda\mu} \delta &= C_{ch} \delta \\ \beta &= \mathcal{R}_1^{\nu\xi} |\delta|^p &= C_{df} |\delta|^p \end{aligned} \right\} : (v),$$

where c , h , d , and f are defined so that

$$C_{ch} = \mathcal{R}_1^{\lambda\mu} \quad \text{and} \quad C_{df} = \mathcal{R}_1^{\nu\xi}. \quad (7.19)$$

Consequently,

$$\left. \begin{aligned} \alpha &= \mu \alpha_- + \lambda \delta &= \frac{1}{h} \sum_{i=0}^t c_i \delta_i \\ \beta &= \xi \beta_- + \nu |\delta|^p &= \frac{1}{f} \sum_{i=0}^t d_i |\delta_i|^p \end{aligned} \right\} : (v)_t | \lambda_t, \mu_t, \nu_t, \xi_t, p. \quad (7.20)$$

(We would not use the correlations in the first form of the algorithm but we

*We will relax those conditions in volume II.

will need them in the second.)

Now, we can define

$$\left. \begin{aligned} a &= \alpha y \\ b &= \beta |y|^p \end{aligned} \right\} : (u, v)_t | y(u)_t, p$$

and substitute them into the Syc algorithm to obtain the first version of Sync, which is shown in figure 7.3. For simplicity, we have removed weight decay

Sync $((G, F, Y, S),$	aaa network to be trained
$\ell,$	loss function
$\eta = 10^{-3},$	gradient learning rate
$d = 100,$	drift
$h = 10,$	hysteresis
$\theta_t = 0.9 \frac{t}{t+1},$	synaptic coherence momentum
$\mu_t = 0.9 \frac{t}{t+1},$	gradient coherence momentum
$\xi_t = 0.999 \frac{t}{t+1},$	gradient norm momentum
$p = 2,$	p -norm for cross-gradient coherence
$\epsilon = 10^{-8}$	numerical stability term
)	

$\mu_0 = \xi_0 = 0$
 for $(r, v) \in E$ $\tilde{w}(r, v)_0 = \tilde{w}(r, v)_0$
 for $t = 0, 1, \dots$
 for $v \in V$
 $\delta = \partial \ell / \partial P_v$
 $\alpha = \mu \alpha_- + (1 - \mu) \delta$
 if $p \leq 0$ continue
 $\beta = \xi \beta_- + (1 - \xi) |\delta|^p$
 $\left. \begin{aligned} & \end{aligned} \right\} : (v)_t | p, v, \ell_t, (P_v)_t, \lambda_t, \mu_t, \nu_t, \xi_t$

for $(r, v) \in E$
 $\left. \begin{aligned} & g = \begin{cases} \frac{\alpha y}{\beta^{1/p} |y| + \epsilon} & \text{if } p > 0 \\ \alpha y & \text{otherwise} \end{cases} \\ & \hat{w} = w - \eta g \\ & \tilde{w}_+ = \begin{cases} \theta_+ (\tilde{w} - w) + \hat{w} & \text{if } t \equiv_h 0 \\ \tilde{w} & \text{otherwise} \end{cases} \\ & w_+ = \begin{cases} \tilde{w}_+ & \text{if } t \equiv_d 0 \\ \hat{w} & \text{otherwise} \end{cases} \end{aligned} \right\} : (r, v)_t \left| \begin{array}{l} y(r)_t, \alpha(v)_t, \beta(v)_t, \\ \eta_t, \theta_t, \hat{w}, p, d, h \end{array} \right.$

Figure 7.3: First version of the Sync algorithm.

and the infinity norm when $p = 0$. (If $p = 0$ there would be no gradient normalization in Sync.)

Let us now derive a more accurate (and more complex) version of Sync.

For that purpose, we define two additional sequences

$$\left. \begin{aligned} \hat{\alpha} &= \mathcal{R}_1^{\lambda\mu} y &= C_{ch} y \\ \hat{\beta} &= \mathcal{R}_1^{\nu\xi} |y|^p &= C_{df} |y|^p \end{aligned} \right\} : (v).$$

By their definition, it is

$$\left. \begin{aligned} \hat{\alpha} &= \mu\alpha_- + \lambda y &= \frac{1}{h} \sum_{i=0}^t c_i y_i \\ \hat{\beta} &= \xi\beta_- + \nu|y|^p &= \frac{1}{f} \sum_{i=0}^t d_i |y_i|^p \end{aligned} \right\} : (v)_t | \lambda_t, \mu_t, \nu_t, \xi_t, p. \quad (7.21)$$

The sequence a is defined in Syc as $a = \mathcal{R}_1^{\lambda\mu} g$. There is one such sequence $a(u, v)$ and a gradient sequence $g(u, v)$ for each connection $(u, v) \in E$. Consequently, the exact equation is $a = \mathcal{R}_1^{\lambda\mu} g : (u, v) | \lambda, \mu$; and because $\mathcal{R}_1^{\lambda\mu} = C_{ch}$ and $g(u, v) = \delta(v)y(u)$ by definition, it is

$$a = \mathcal{R}_1^{\lambda\mu}(\delta y) = C_{ch}(\delta y) : |a(u, v), \delta(v), y(u). \quad (7.22)$$

Similarly, the sequence b of Syc is

$$b = \mathcal{R}_1^{\lambda\mu} |\delta y|^p = C_{ch} |\delta y|^p : |b(u, v), \delta(v), y(u). \quad (7.23)$$

We would now approximate a with a function of the sequences α and $\hat{\alpha}$; and we would likewise approximate b with a function of the sequences β and $\hat{\beta}$.

If X and Z are two random variables with expected values EX and EZ then we can consider the following tautology

$$(X - EX)(Y - EY) = XY - XEY - YEX + EXEY,$$

which implies that

$$XY \approx XEY + YEX - EXEY$$

when X and Y tend to be close to their expectations EX and EY , respectively.

For any sequence x , we can define the sequence of its averages $\bar{x} = \frac{t}{t+1}\bar{x}_- + \frac{1}{t+1}x = \frac{1}{t+1} \sum_{i=0}^t x_i : t|t$. Then we can apply the above approximation to obtain

$$x_i y_i \approx x_i \bar{y}_t + y_i \bar{x}_t - \bar{x}_t \bar{y}_t \quad (7.24)$$

for $i \leq t$. (Note that we used the averages $\bar{y}_t$ and $\bar{x}_t$ instead of $\bar{y}_i$ and $\bar{x}_i$, which effectively assumes stationarity.)

By definition (equation (7.22)), it is

$$a = \frac{1}{h} \sum_{i=0}^t c_i \delta_i y_i : t|t : |a(u, v), \delta(v), y(u). \quad (7.25)$$

If we now apply (7.24) to (7.25), we can approximate each product $\delta_i y_i$ with $\delta_i \bar{y}_t + y_i \bar{\delta}_t - \bar{\delta}_t \bar{y}_t$ for $i \leq t$. The result is

$$\begin{aligned}
 a &\approx \frac{1}{h} \sum_{i=0}^t c_i (\delta_i \bar{y} + y_i \bar{\delta} - \bar{\delta} \bar{y}) \\
 &= \bar{y} \langle C_{ch} \delta \rangle + \bar{\delta} \langle C_{ch} y \rangle - \bar{\delta} \bar{y} \frac{1}{h} \sum_{i=0}^t c_i \\
 &= \bar{y} \alpha + \bar{\delta} \hat{\alpha} - \bar{\delta} \bar{y} \frac{1}{h} \sum_{i=0}^t c_i \quad : t|t : |a(u, v), \alpha(v), \hat{\alpha}(u), \delta(v), y(u).
 \end{aligned}$$

Similarly, we can derive an approximation for b

$$\begin{aligned}
 b &\approx \frac{1}{f} \sum_{i=0}^t d_i (|\delta_i|^p \overline{|y|^p} + |y_i|^p \overline{|\delta|^p} - \overline{|\delta|^p} \overline{|y|^p}) \\
 &= \overline{|y|^p} \langle C_{df} |\delta|^p \rangle + \overline{|\delta|^p} \langle C_{df} |y|^p \rangle - \overline{|\delta|^p} \overline{|y|^p} \frac{1}{f} \sum_{i=0}^t d_i \\
 &= \overline{|y|^p} \beta + \overline{|\delta|^p} \hat{\beta} - \overline{|\delta|^p} \overline{|y|^p} \frac{1}{f} \sum_{i=0}^t d_i \quad : t|t, p : |b(u, v), \beta(v), \hat{\beta}(u), \delta(v), y(u).
 \end{aligned}$$

Furthermore, if the correlations C_{ch} and C_{df} are cumulative, that is if $\lambda = 1 - \mu$ and $\nu = 1 - \xi$ as we assumed in Sync and the first version of Sync, it would be $h_t = \sum_{i=0}^t c_i$ and $f_t = \sum_{i=0}^t d_i$. The above approximations then reduce to

$$\begin{aligned}
 a &\approx \bar{y} \alpha + \bar{\delta} \hat{\alpha} - \bar{\delta} \bar{y} \quad : t : |a(u, v), \alpha(v), \hat{\alpha}(u), \delta(v), y(u) \\
 b &\approx \overline{|y|^p} \beta + \overline{|\delta|^p} \hat{\beta} - \overline{|\delta|^p} \overline{|y|^p} \quad : t|p : |b(u, v), \beta(v), \hat{\beta}(u), \delta(v), y(u).
 \end{aligned}$$

Adding the above approximations to our algorithm, we obtain that second version of Sync, which is shown in figure 7.4.

Sync has a memory overhead of 100% since the overhead of parametric coherence remains but the overhead of gradient and cross-gradient coherence is now linear with the number of units in the network.

We should note that the approximations of Sync effectively assume that the covariance $\text{Cov}(y(u), \delta(v)) = \text{Cov}(Y_u, \frac{\partial \ell}{\partial P_v})$ is small for each connection (u, v) in the network. The algorithm would be exact if the variables Y_u and $\frac{\partial \ell}{\partial P_v}$ were *uncorrelated* (their covariance would then be zero by definition).

In general, this is not the case. There is usually some covariance between the above random variables. Our limited empirical evidence so far suggests that the covariance tends to be small on average; but more evidence is required to make a definitive statement. Conversely, when the covariance is persistently large, using Sync may introduce a coherence bias.

Sync	$((G, F, Y, S),$	network to be trained
	$\ell,$	loss function
	$\eta = 10^{-3},$	gradient learning rate
	$d = 100,$	drift
	$h = 10,$	hysteresis
	$\theta_t = 0.9 \frac{t}{t+1},$	synaptic coherence momentum
	$\mu_t = 0.9 \frac{t}{t+1},$	gradient coherence momentum
	$\xi_t = 0.999 \frac{t}{t+1},$	gradient norm momentum
	$p = 2,$	p -norm for cross-gradient coherence
	$\epsilon = 10^{-8}$	numerical stability term
	)	

$\mu_0 = \xi_0 = 0$	
for $(r, v) \in E$	$\tilde{w}(r, v)_0 = \tilde{w}(r, v)_0$
for $t = 0, 1, \dots$	
for $v \in V$	
$\delta = \partial \ell / \partial P_v$	
$\alpha = \mu \alpha_- + (1 - \mu) \delta$	
$\hat{\alpha} = \mu \hat{\alpha}_- + (1 - \mu) y$	
$\bar{\delta} = (t \bar{\delta}_- + \delta) / (t + 1)$	
$\bar{y} = (t \bar{y}_- + y) / (t + 1)$	
if $p \leq 0$ continue	
$\beta = \xi \beta_- + (1 - \xi) \delta ^p$	
$\hat{\beta} = \xi \hat{\beta}_- + (1 - \xi) y ^p$	
$(\overline{ y ^p}) = (t(\overline{ y ^p})_- + y ^p) / (t + 1)$	
$(\overline{ \delta ^p}) = (t(\overline{ \delta ^p})_- + \delta ^p) / (t + 1)$	
for $(u, v) \in E$	
$g = (\bar{y} \alpha + \bar{\delta} \hat{\alpha} - \bar{\delta} \bar{y}) y$	
if $p > 0$	
$b = (\overline{ y ^p}) \beta + (\overline{ \delta ^p}) \hat{\beta} - (\overline{ \delta ^p})(\overline{ y ^p})$	
$g = \frac{g}{b^{1/p} y + \epsilon}$	
$\hat{w} = w - \eta g$	
$\tilde{w}_+ = \begin{cases} \theta_+(\tilde{w} - w) + \hat{w} & \text{if } t \equiv_h 0 \\ \tilde{w} & \text{otherwise} \end{cases}$	
$w_+ = \begin{cases} \tilde{w}_+ & \text{if } t \equiv_d 0 \\ \hat{w} & \text{otherwise} \end{cases}$	

}

: $(v)_t | p, v, \ell_t, (P_v)_t, \lambda_t, \mu_t, \nu_t, \xi_t$

}

: t

$w(u, v)_t, \tilde{w}(u, v)_t$
 $y(u)_t, \delta(v)_t$
 $\alpha(v)_t, \hat{\alpha}(v)_t,$
 $\beta(v)_t, \hat{\beta}(v)_t,$
 $g, b, p, \hat{w}, d, h$

Figure 7.4: The Sync algorithm.

If we wish to develop a better algorithm, we can sample the covariance and form a criterion for switching between Syc and Sync for individual connections, dendrites, or other groups of connections depending on the value of the associated covariances and the available resources. This will lead us to a hybrid algorithm between Syc and Sync that will be presented in more detail in volume II. There are additional algorithms that use different approximations as well as exact estimates for certain cases. We will elaborate on the topic and present such algorithms in volume II.

The most important part of this discussion is the opportunity for algorithmic evolution in learning once we start looking for more economical solutions instead of accepting learning algorithms as they are. If we then start examining the statistical properties of underlying variables, important relationships and phenomena may emerge. In nature, natural selection and energy consumption steered evolution in that direction virtually from the beginning of life.

7.8 A Research Note

There are four general research goals and directions behind coherent learning algorithms such as Syc and Sync.

The first goal is to provide a unifying framework that can represent existing learning algorithms and help us understand their differences and trade-offs in a broader and more formal way.

The second goal is the algorithms themselves as a pair of more flexible and more sophisticated (or less simplistic) learning methods that can be used in applications to train networks and accumulate important empirical evidence and architectural validation.

Our third goal was mentioned in the previous section and it is about understanding the importance of economical and efficient solutions. Efficiency and economy are not optional features in learning. They are essential driving forces.

Finally, our fourth and most important goal is to introduce coherence as a main learning construct and mechanism, with Syc and Sync as initial examples or solutions, and steer neural network research toward synaptic and neural coherence so that more sophisticated and effective methods can be developed by the community. This is actually a recurring theme in Mathesis. The theory is not intended to be a destination or an end but rather a departure and a beginning.

7.9 Historical and Bibliographical Remarks

Our notation for partial functions is a slight modification of the notation used in [104]. (We write $f(x) = \uparrow$ instead of $f(x)\uparrow$ to indicate that f is undefined at x .)

The term hysteresis was used in the 1890s by J. A. Ewing to describe the behaviour of magnetic materials but its first known use traces back to 1881 as a retardation of an effect from changing forces or conditions (as though through viscosity or internal friction).

Momentum as a way to speed up optimization methods was introduced in 1962 and was published in 1964 [274]. It was first used in the context of gradient descent in neural networks in 1986 [290]. An analogy with physical systems and the momentum of moving objects is often used to describe momentum gradient descent. (The parameter w is seen as a moving point mass and its gradient is the acceleration along the loss surface.)

The Adagrad learning algorithm was published in 2011 [97] and the Adam learning algorithm in 2014 [187]. Unbiased exponential smoothing was added in Mathesis earlier but it appeared in Adam in 2014 and it was an independent discovery. Although this seems to be the first published application of unbiased smoothing, the authors of Adam only used their result as a component of their learning algorithm. They did not identify the broader issue with exponential moving average; and they did not propose their result as a general or independent unbiased estimator.

Overall, Adam has been a significant influence and has provided reinforcement for Mathesis. The popularity of the algorithm and its success have effectively demonstrated the importance of coherence and the capacity of straightforward normalization methods as a cross-gradient coherence mechanism.

This is only a foretaste of what is to come, and only the shadow of what is going to be.

— Alan Turing, article in *The Times of London*, June 11, 1949.

8.1	Synaptic Design	265
8.2	Striders, Corollas, and Flowers	267
8.3	Growing Flowers	272
8.4	Data Superposition	275
8.5	MNIST	277
8.6	Implementation Woes	287
8.7	Mathesis for MNIST	289
	8.7.1 SE Block Extension	289
	8.7.2 Local Pooling	290
	8.7.3 CSE Blocks with Dropout	291
	8.7.4 Squeeze-and-Excitation Flower Nets	292
	8.7.5 Diversity in the Ensemble	294
	8.7.6 Data Coherence	299
	8.7.7 Training and Testing the SEF Ensemble	302
8.8	Publication Bias	306
8.9	Flowers for ImageNet	307
8.10	The N Programming Language	307
8.11	Historical and Bibliographical Remarks	311

Preliminary Evaluation

In this chapter, we provide a preliminary evaluation of some parts of Mathesis using an optical character recognition data set. We have interleaved the evaluation with specific forms of coherence and few other concepts that define our approach and affect our results.

Data coherence and structural coherence are the two main elements. We implement them with methods that use superposition to combine training samples and methods that specify a new class of connectivity or *synaptic design* derived from simple mathematical concepts. Diversity and synergy are also important. Our final model is thus a diverse ensemble of structurally coherent networks trained on data with increased coherence.

The performance of our model is remarkable and provides the first validation of our theory. However, this is only an initial evaluation on a single data set. Multiple experiments and more extensive evaluations of Mathesis across different data sets will take place in volumes II and III.

8.1 Synaptic Design

Throughout this chapter, we assume a neural network $N = (G, F, Y, S)$ as in definition 4.2. We similarly assume that $G = (V, E)$.

Given a set of units $U \subseteq V$ in the network, we can consider the set $I_U = \{I_u/u \in U\}$ or the set $S_U = \{S_u/u \in U\}$, which define incoming and successive connections, respectively.[†] We can then consider the structure of those sets. In the following, we only consider the sets S_U , for $U \subseteq V$, since the same conditions also apply to the sets I_U .

Synaptic coherence was defined in section 5.10. It is an important neural property that tends to cluster similar functionality, reduce cross-connection requirements and redundancy, and improve generalization (as it is often a reflection of coherence in the environment or the underlying computation). Consequently, locality is a desirable property in the set S_V – as well as its

[†]We remind the readers that $I_u = G^{-1}(u)$ for each unit $u \in V$; and if l_u indicates the index k of the layer L_k that contains u then $S_u = I_u \cap L_{l_u-1}$.

subsets S_U , for $U \subseteq V$. If synaptic incoherence $\mathbf{I}_N$ is used to measure it, $\mathbf{I}_N V$ should be kept small.

Diversity in the set S_V is also important because it can support broader functionality under learning conditions. A gradual shift in structural differences is desirable because it enables a similar and gradual shift of functionality across the units V of the network N .

If we combine the above properties and consider the set S_{L_k} for each layer L_k of N and $k > 0$, we can define a set of constraints or goals.

For each unit $v \in L_k$, the set S_v should be relatively small (compared to the size of L_{k-1}) and such that the synaptic incoherence $\mathbf{I}_N v$ of the unit v is small. The set S_v should also tend to be different across different units $v \in L_k$ so that the set S_{L_k} has a number of elements that is proportional to the number of units in L_k . The size of the set S_{L_k} is actually a measure of diversity in L_k . A more complex measure should take into account the intersections of sets S_u and S_v , for all distinct units $u, v \in L_k$. Diversity combined with a gradual shift of structural differences would furthermore imply that successive units $u, v \in L_k$ tend to have a larger intersection $S_u \cap S_v$.

As an extreme (and poor) example, we can consider full connectivity. If L_k is a fully connected layer then, for each unit $u \in L_k$, the set S_u is maximal since it is $S_u = L_{k-1}$; and its synaptic incoherence is large or maximal (depending on the precise definition of such incoherence). Furthermore, $S_{L_k} = \{L_{k-1}\}$, which is a singleton set. S_{L_k} is thus minimal and diversity is meaningless. As a result, a fully connected layer is not merely a poor choice but most likely the worst possible choice of connectivity among those that can represent a solution to a given problem. (The only exceptions are small artificial or logic problems that have no practical value. Such problems should not be handled with neural networks or learning.)

In general, the structure of the sets S_{L_k} , for $k > 0$, is an important design consideration as well as an element of hereditary information that can be passed from one generation to the next, improved by evolution, and fine-tuned locally by neural plasticity. The same applies to the sets S_U for all subsets $U \subseteq V$ and thus all partitions of N .

We use the term *neural subset* to refer to any such subset U but we will usually operate on specific subsets such as layers or subsets of layers that have specific properties. We define *successive neural subsets* to be two or more neural subsets that can be arranged in a sequence so that each provides input to the next. Furthermore, we define *synaptic structure* to be the structure of two successive neural subsets. The first subset is the *synaptic domain* of the structure while the second is its *synaptic range*. Finally, we use the term *synaptic map* to refer to a partition of V and the synaptic structure of its parts. *Synaptic design* is merely a reference to methods that define such maps.

In the next section, we will define some of the simplest possible synaptic structures and the associated maps. We will use some of those maps in the

experiments of this chapter. However, we will introduce considerably more complex structures and maps in volumes II and III of Mathesis.

8.2 Striders, Corollas, and Flowers

In set theory and combinatorics, a *sunflower* or Δ -*system* is a set of sets that have identical pairwise intersections. Hence, S is a sunflower iff

$$(\forall a, b \in S) \ a \neq b \rightarrow a \cap b = \bigcap_{x \in S} x.$$

For example, the set $S = \{C \cup P_i / i \in \mathbb{N}_5\}$ constitutes a sunflower, if $P_i \cap P_j = \emptyset$ for all $i \neq j$. Figure 8.1 shows a Venn diagram of the above sets and explains why such sets are called sunflowers. The common pairwise intersection of the elements of a sunflower is called the *core* or *kernel* of the sunflower. For obvious reasons, we use the term *petals* to refer to the elements of a sunflower, excluding their common core (as in the botanical definition but deviating from the mathematical convention in set theory). Hence, in figure 8.1, the petals are the sets P_i , for $i \in \mathbb{N}_5$.

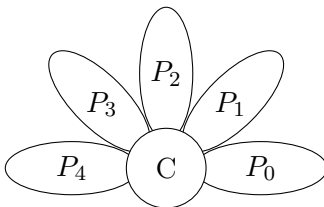


Figure 8.1: A sunflower with five petals $C \cup P_i$, for $i < 5$.

Intuitively, a sunflower constrains its elements so that their pairwise intersection is the same as the intersection of all its elements. Equivalently, it requires that its petals be pairwise disjoint. We can relax this strict requirement and allow successive petals to intersect as shown in figure 8.2. This is particularly useful when the petals represent entities that admit a topological or geometric arrangement or interpretation.

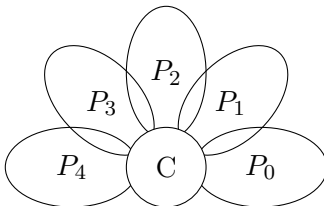


Figure 8.2: A flower with five petals $C \cup P_i$, for $i < 5$.

We can formally define a generalization of such a structure as follows.

Definition 8.1. A *flower* is a set of sets that can be written in the form $F = \{C\} \cup \{P_i/i \in \mathbb{N}_n\}$, for some $n \in \mathbb{N}$, and so that its elements satisfy the following conditions

- a. $(\forall i \in \mathbb{N}_n) \ C \cap P_i = \emptyset$
- b. $(\forall i, j \in \mathbb{N}_n) \ i < j - 1 \rightarrow P_i \cap P_j \subset P_{j-1} \cap P_j$
- c. $(\forall j, k \in \mathbb{N}_n) \ j + 1 < k \rightarrow P_j \cap P_k \subset P_j \cap P_{j+1}$

The sets P_i are the *petals* of the flower while the set C is the *flower core* or *kernel*. The set $C \cup \bigcup_{i < n} P_i$ defines the *domain* of the flower. The sizes of successive petal differences $s = |P - P_-| : i > 0$ are the *petal strides* of the flower. When $s = |P| : i > 0$ (or equivalently $P \cap P_- = \emptyset : i > 0$), the flower represents a *sunflower*. When $C = \emptyset$, the flower is called a *corolla*.

Intuitively, a set of sets is a flower if and only if one of its elements is disjoint from the rest, so that it can act as the core of the flower, and the rest of the elements can be arranged so that, for any given element P , the pairwise intersection of P and any other element Q decreases with the distance between P and Q in either direction of the arrangement. Hence, pairwise intersections are maximal for successive elements P and Q .

This local maximization and gradual drift of the petals tends to create greater coherence and clusters of gradual functionality when a flower is used to define the synaptic structure of neural subsets. In that case and if the flower has a non-empty core, the core represents units of the first subset that are connected to all units of the second subset. The core units are thus the equivalent of global variables within their synaptic structure. Consequently, core units should be allocated with caution and restraint.

Let us now define a concept that can help us generate a simple type of flower.

Definition 8.2. A *flower strider* or simply *strider* is a tuple of the form

$$(f, p, n, s, c) \in \mathbb{N}^5$$

where $0 < s \leq p \leq f \leq p + (n - 1)s$. When $c = 0$, the strider is called a *corolla strider*. When $f = p + (n - 1)s$, the strider is called *straight*. A *ring strider* allows $f < p + (n - 1)s$. A sequence of flower striders $(f, p, n, s, c) : i$ is *segmented* iff $n_- | f : i$, where $a | b$ denotes that a divides b . In that case, the ratios $g = f/n_- : i$ are the *segment sizes* of the striders. (We define $g_0 = 1$.) Finally, a strider sequence is *petal aligned* if $g | p : i$, it is *stride aligned* if $g | s : i$, and it is *fully aligned* when it is both petal and stride aligned.

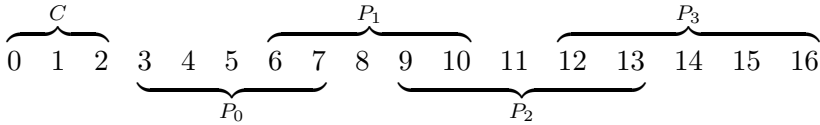
We will elaborate on ring striders and use them in experiments in volume II. For the remaining of this chapter, we will use straight striders only.

Intuitively, when a strider is used to represent a flower, f will be the size of the flower's corolla, which is the union of its petals, p will be the petal size, which will be the same for all petals, s will be the stride, which will also be

the same for all petals, n will be the number of petals, and c will be the size of the flower's core. A strider can thus be used to define a specific type of flower.

Definition 8.3. A *linear flower* is a flower with constant petal size and stride. It can be defined by a straight strider* (f, p, n, s, c) over a domain $D = \{x_j/j \in \mathbb{N}_{f+c}\}$. In that case, the core is the set $C = \{x_j/j \in \mathbb{N}_c\}$ and the petals are the sets $P_i = \{x_{j+c+is}/j \in \mathbb{N}_p\}$ for $i < n$. p is the constant *petal size*, s is the constant *petal stride*, n is the number of petals, and f is the size of the corolla of the flower. *Segmented flowers* are defined by a sequence of segmented striders.

For example, the straight strider $(14, 5, 4, 3, 3)$ over the domain $\mathbb{N}_{17}$ defines a flower with core defined by the set $\{0, 1, 2\}$ and petals defined by the sets $\{3, 4, 5, 6, 7\}$, $\{6, 7, 8, 9, 10\}$, $\{9, 10, 11, 12, 13\}$, and $\{12, 13, 14, 15, 16\}$. If we arrange the elements of the domain in a row, we can illustrate the core and the petals as shown below:



Flowers are useful because they can be used to define synaptic structures and maps.

Definition 8.4. A synaptic structure is a *flower structure* iff it is defined by a flower over its synaptic domain and so that the units of petal P_i are the only inputs from the synaptic domain to the units of segment i of the synaptic range. A *flower map* is a synaptic map of a neural network that contains only flower structures. The structures and the map are *linear* when the flowers are linear. Similarly, the structures and the map are *segmented* when the flowers are segmented.

In order to understand flower structures more completely, let us consider some examples. For simplicity, we assume that there are no flower cores.

Figure 8.3 shows three successive layers in a neural network with two flower structures that are defined by two linear sunflowers. The sunflowers are defined by the straight striders $(f_1, p_1, n_1, s_1, c_1) = (12, 3, 4, 3, 0)$ and $(f_2, p_2, n_2, s_2, c_2) = (8, 4, 2, 4, 0)$, respectively. The segment of the second layer has size $g_2 = f_2/n_1 = 2$, which divides p_2 and s_2 . If we interpret the third layer as a sunflower defined by the strider $(f_3, p_3, n_3, s_3, c_3) = (4, 2, 2, 2, 0)$, we obtain that $g_3 = f_3/n_2 = 2$, which divides p_3 and s_3 . This synaptic map is thus fully aligned.

Linear sunflowers with empty cores are the sparsest among all linear flowers with the same corolla. Their sparsity furthermore increases with their

*A linear flower can also be defined by a ring strider as we shall see in volume II.

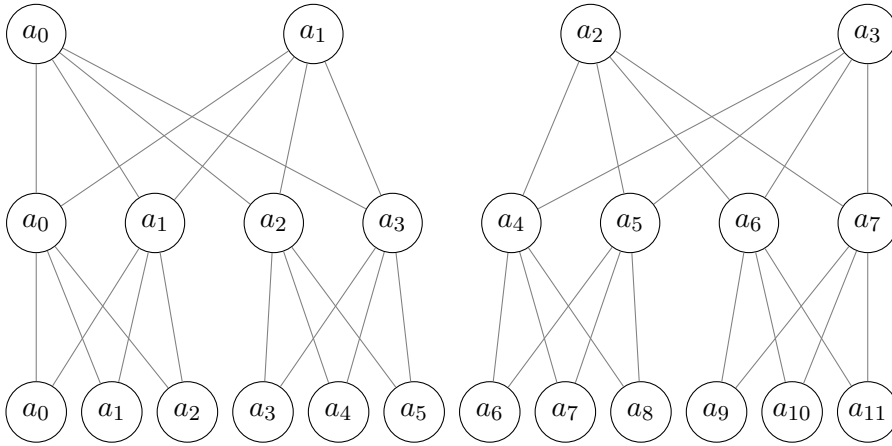


Figure 8.3: Flower structure of three successive layers defined by two linear sunflowers.

number of petals. Indeed, if $\langle (f_k, p_k, n_k, s_k, 0)/k \in \mathbb{N} \rangle$ is a sequence of straight striders that generate flower structures in successive layers of a network, the number of connections between layers L_k and L_{k+1} is

$$f_+p = f_+f - f_+(n-1)s \geq f_+f - f_+(n-1)p \quad :k$$

since $s \leq p:k$ by definition.* Given that $s = p:k$ for sunflowers, the number of connections is minimal for sunflowers and a decreasing function of the number of petals n_k .

Sunflowers tend to be overly sparse, especially when the corollas are large. Decreasing their number of petals to increase density is problematic because, for any given size of the corolla, it makes the petals larger and thus reduces locality and coherence. More importantly, sunflowers partition the corolla into disjoint petals and thus create a stepwise transition of structural differences across petals. If we assume that small clusters and a gradual transition of structure and functionality are better for generalization, sunflowers are clearly at a disadvantage. Of course, they are better than full connectivity but they usually do not perform as well as other flowers.

We can easily change the previous example and use flowers in its structure instead of sunflowers. We show an example in figure 8.4. The flowers are defined by straight striders $(f_1, p_1, n_1, s_1, c_1) = (12, 6, 4, 2, 0)$ and $(f_2, p_2, n_2, s_2, c_2) = (8, 5, 4, 1, 0)$, respectively. Given that $g_2 = f_2/n_1 = 2$, $p_2 = 5$, and $s_2 = 1$, the first flower structure is neither segment nor petal

*The condition $s \leq p$ is required by definition in striders because if the stride s were bigger than the petal p there would be $s - p$ dangling units after the end of each petal that will not be used as input for any other units. For ring striders, this condition can be relaxed without wasting units but the relaxed version tends to make connections longer, if the units are geometrically allocated according to their indices, and does not seem to offer any advantages.

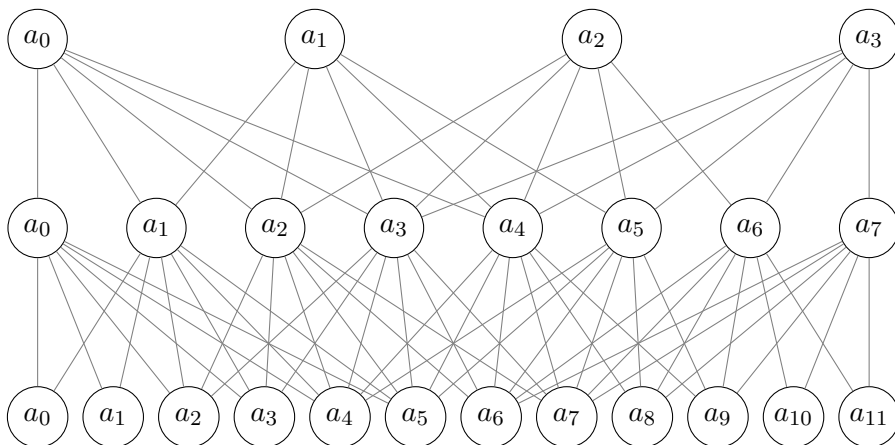


Figure 8.4: Flower structure of three successive layers defined by two flowers.

aligned.*

CNNs are local in their spatial dimensions but fully connected in their filters. Consequently, when flower structures are used in CNNs, they apply to the filters only and make them local. We will refer to such networks as *flower nets*.

As an example, figure 8.5 shows a Venn diagram of the flowers in the first three layers of a flower net that will be used for optical character recognition later in this chapter. The flowers are actually corollas since there are no cores in the flower net. All those corollas have seven petals; and the petals are depicted as ellipses with areas that are proportional to the number of filters they contain. The figure also shows the filter connectivity of the first three layers. However, only one filter from each segment of the third layer is included to make the structure more visible. (The third layer of the network contains 98 filters but only 7 are shown in the figure.)

Segmentation and alignment are important properties of synaptic structures but they are beyond the scope of this chapter. We will discuss them in detail in volume II.

At this point, we should indicate that segmentation and alignment effectively constrain petals and strides so that no petals contain partial segments. Although such alignment tends to improve generalization, it represents an excessive restriction for many linear flowers. As a result, we have opted for petal alignment only in the experiments of this chapter.

Plasticity will eventually introduce misalignment even in the best flower structures. A limited amount of misalignment can easily arise from irregularities in the underlying functionality of a network. However, it is easier

*This is just an example for variability. We can easily make the first flower structure fully aligned if we pick $(f_2, p_2, n_2, s_2, c_2) = (8, 6, 2, 2, 0)$. If we furthermore interpret the third layer as a sunflower the same way we did in the previous example, the flower map becomes fully aligned.

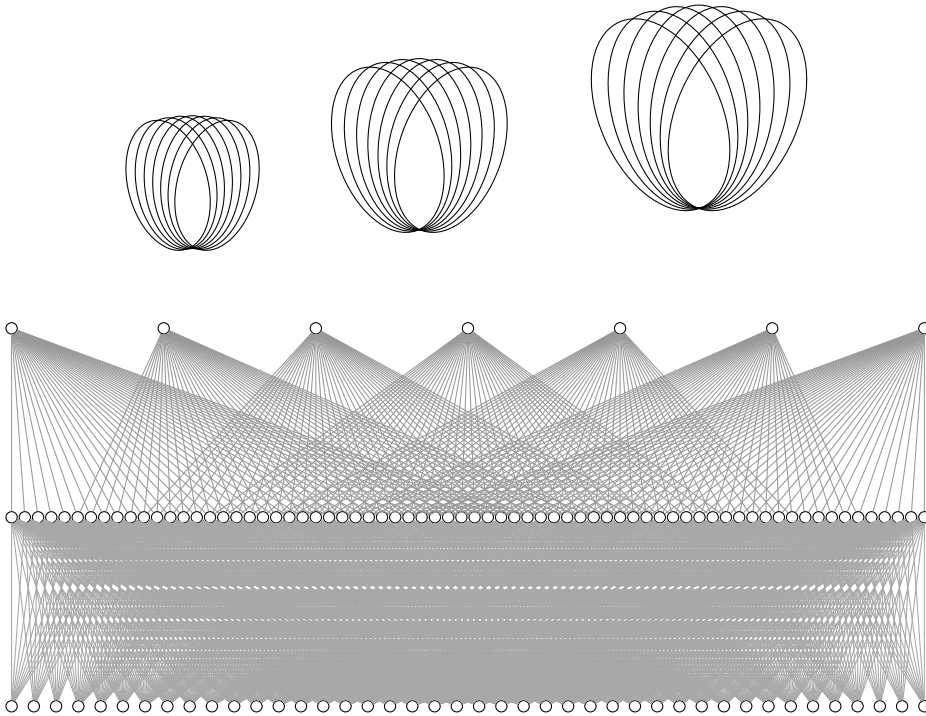


Figure 8.5: A Venn diagram and filter connectivity of the first three layers of a flower net. Only a single filter for each segment of the third flower is shown in the graph to make the structure more visible.

and better to learn such misalignment (through plasticity methods) than to design it somehow and include it in advance. Consequently, if we expect some amount of misalignment in optimal structures, starting from fully aligned structures is not necessarily a better choice. There may be more to add and overcome in full alignment than in partial one. Our choice and our emphasis will often be on petal alignment, especially for linear flowers that are generated by straight striders.

8.3 Growing Flowers

Neural network layers vary significantly within individual networks and more across different networks and applications. A main difference is that the layers usually have different sizes as measured by their number of units or number of filters for CNNs. The number of filters tends to grow with the layer depth in CNNs. Flower nets are similar in that regard. As a result, we would like to generate flowers of varying and usually increasing sizes.

For alignment purposes, we would like to define some form of *multiplication* between a real number m and a flower F so that the flower F changes proportionally to m and its alignment is maintained. However, those two

conditions are often impossible to satisfy. Moreover, proportional growth introduces ambiguity because it is not clear whether it applies to the number of petals as well. If the number of petals is affected then multiplication must define how the growth is distributed between the petals and the strides. Finally, multiplication must define how the flower core is affected.

For linear flowers, the operation translates to multiplication of a number m with the strider (f, p, n, s, c) that has generated the flower – but the same issues apply. Consequently, there are various forms of multiplication that produce different approximations to proportional growth (or reduction if $m < 1$) and alignment.

In general, we use the term *flower multiplication* for the multiplication of a flower with a real number m . Similarly, we use the term *strider multiplication* for the multiplication of a strider with m . We will discuss those types of multiplication in more detail in volume II; and we will present various multiplication methods.

In this section, we will define a specific type of strider multiplication that we will use exclusively in the remaining of this chapter. Given that strider multiplication involves a number of parameters that define its result, we do not define it independently of the strider. Instead, we have chosen an object-oriented approach where the strider contains parameters that define how multiplication works for it.

Definition 8.5. A *fair strider* is a straight strider that enforces segmentation and attempts to update its parameters proportionally (and fairly) upon multiplication. If we assume a sequence of such striders $\langle (f_i, p_i, n_i, s_i, c_i) / i \leq k \rangle$ that are generated through multipliers $\langle m_i / i < k \rangle$ so that $(f, p, n, s, c)m = (f_+, p_+, n_+, s_+, c_+) : i$ then

$$\left. \begin{aligned} f' &= \lceil mf/n \rceil n \\ \mu &= f'/f \\ n_+ &= \begin{cases} \max(1, \lceil nr\mu^e \rceil) & \text{if } a \\ 1 + \lceil (n-1)r\mu^e \rceil & \text{otherwise} \end{cases} \\ s_+ &= \max(1, \lceil s\rho\mu^\epsilon \rceil) \\ f_+ &= f' + \max(\lceil (n_+s_+ - f')/n \rceil n, 0) \\ p_+ &= f_+ - (n_+ - 1)s_+ \\ c_+ &= \lceil c\mu^\gamma \rceil \end{aligned} \right\} : i | a, r, e, \rho, \epsilon, \gamma$$

where a specifies whether the number of petals n or the quantity $n - 1$ should be multiplied, r is the rate of growth of n (or $n - 1$), e is an exponent applied to the multiplier (usually $0 \leq e \leq 1$), ρ is the rate of growth for the stride s , ϵ is the multiplier exponent for the stride, and γ is the multiplier exponent for the core.

Let us now explain how fair strider multiplication works. The strider first

tries to estimate the value of the corolla but it forces segmentation and thus defines the auxiliary variable f' to be the integer multiple of n that is nearest to the product mf . If f' is to be the new corolla then the actual multiplier between the two corollas will be $\mu = f'/f$. This multiplier is then used to compute n_+ and s_+ using their growth rates and exponents. Finally, the corolla and the petal are defined so that the strider remains straight, or equivalently $p_+ = f_+ - (n_+ - 1)s_+$. However, the quantity $\max(\lceil (n_+s_+ - f')/n \rceil n, 0)$ is added to the corolla to ensure that $s_+ \leq p_+$. This quantity is a multiple of n so that the updated corolla will also be a multiple of n to initiate or preserve segmentation in the sequence.

Listing 8.1 shows python code for fair striders, which are implemented through the class `FairStrider`.

Listing 8.1: Fair strider code.

```
class FairStrider(FlowerStrider):
    """A fair strider attempts to update its parameters proportionally
    upon multiplication as shown below:
        (f, p, n, s, c) * m = (fx, px, nx, sx, cx)
    where
        fx = round(m * f / self.n) * self.n,
        nx = 1 + (n-1) * n_r * (fx/f)**n_e, if not a_n,
        nx = n * n_r * (fx/f)**n_e, if a_n,
        sx = s * s_r * (fx/f)**s_e,
        cx = c * (fx/f)**c_e,
        fx += max(0, ((nx*sx - fx - 1) // self.n + 1) * self.n,
        px = fx - (nx - 1)*sx.
    Attributes:
        f (integer): flower size. Default: None.
        p (integer): petal size. Default: None.
        n (integer): number of petals. Default: None.
        s (integer): petal stride. Default: None.
        c (integer): core size. Default: None.
        retain (Boolean): whether to retain previous values.
            Default: True.
        n_e (float): growth exponent for the number of petals n.
            Default: 0.5.
        s_e (float): growth exponent for the petal stride s.
            Default: 0.5.
        n_r (float): growth rate for the number of petals n.
            Default: 1.
        s_r (float): growth rate for the petal stride s.
            Default: 1.
        a_n (Boolean): whether to apply growth to n or n-1.
            Default: True.
        c_e (float): growth exponent for the core c. Default: 1.
    """
    def __init__(self, f, p, n, s, c=0, retain=True,
                  n_e=0.5, s_e=0.5, n_r=1, s_r=1, a_n=True, c_e=1):
        # Construct base object.
        super().__init__(f, p, n, s, c, retain)
        # Set the multiplicative parameters
        self.n_e = n_e
        self.s_e = s_e
        self.n_r = n_r
        self.s_r = s_r
        self.a_n = a_n # flag for applying multipliers to n.
        self.c_e = c_e
```

```

def mul(self, m):
    """Multiply the strider by the real number m.
    Args:
        m (float): the multiplier for the strider.
    Returns:
        The next setting of the strider."""
    # Make the next flower size divisible by the previous
    # number of petals
    f = round(m * self.f / self.n) * self.n
    # Update the multiplier with its forced value.
    m = f / self.f
    # Estimate the stride s and the petal number n
    s = max(1, round(m ** self.s_e * self.s_r * self.s))
    n_weight = m ** self.n_e * self.n_r
    n = (max(1, round(n_weight * self.n)) if self.a_n else
         round(n_weight * (self.n-1)) + 1)
    # Make sure petal p greater or equal to stride s
    ns = n * s
    if f < ns:
        # f must remain a multiple of self.n
        f += ((ns - f - 1) // self.n + 1) * self.n
    p = f - ns + s
    # Estimate the new core
    c = round(self.c * m ** self.c_e)
    # Update the strider
    self.f, self.p, self.n, self.s, self.c = f, p, n, s, c
    # Possibly retain the new setting.
    self._retain()
    # Return the new setting.
    return f, p, n, s, c

```

The code assumes a base class `FlowerStrider` that defines a strider as a tuple (f, p, n, s, c) along with a Boolean flag *retain* that specifies whether the strider should retain its previous values (in a list) instead of overwriting them. The base class also provides a function `_retain()` which is used to append the current value of the strider to its list, if the strider retains its values. Finally, the class `FairStrider` is derived from the class `FlowerStrider`. In other words, a fair strider is a flower strider with additional parameters and code to implement strider multiplication.

In the remaining of this chapter, we will use flower nets with linear flowers and fixed number of petals across their layers. We will generate those flowers through fair striders with $r = 1$, $e = 0$, and $a = \text{True}$ (so that multiplication does not affect the number of petals n). We will also use $\rho = 1$ and $\epsilon = 1$ so that the corolla f , the petal p , and the stride s will change proportionally to the multiplier m (when the products mf , mp , and ms are integers, $n|f$, and $n|mf$) and thus retain the alignment of the flower.

8.4 Data Superposition

In this section, we will introduce the first method from a sequence of *superposition* methods that generate data by combining two or more related data samples (such as samples of the same class for classification problems). Such methods can serve multiple purposes. They can reduce information. For

example, they can remove unnecessary details from the source data but retain important features so that the generated data can make generalization easier; or they can create variations or generic abstractions from a set of samples and introduce them as new samples in the data set. Superposition methods can also add information. For example, they can repair features that are missing or are compromised, such as obscured objects in an image, broken strokes in handwriting, missing words or grammatical errors in text etc.

Definition 8.6. Assume a data set* D over two sets X and Y . Furthermore, assume that $D = \{(x_i, y_i) \in X \times Y \mid i \in \mathbb{N}_n\}$, for some $n \in \mathbb{N}_+$. A *data superposition method* is a tuple of three functions (C, T, S) such that

- a. $C: 2^{X \times Y} \rightarrow X^{\geq 2}$ is a choice function that chooses tuples of two or more (usually distinct) elements from D :

$$C(D) = [(x_{j_1}, \dots, x_{j_{k_j}}) \mid 2 \leq k_j, j_1, \dots, j_{k_j} \in \mathbb{N}_n, \phi(j_1, \dots, j_{k_j})]$$

where $X^{\geq 2} = \bigcup_{k \geq 2} X^k$ is the set of tuples of X that contain at least two elements and ϕ represents some arbitrary condition on the samples.

- b. $T: X^{\geq 2} \rightarrow X^{\geq 2}$ is a transformation function that transforms the elements of each chosen tuple so that some unspecified properties of those elements tend to coincide or come closer in their index space (usually spatially):

$$T((x_{j_1}, \dots, x_{j_{k_j}})) = (z_{j_1}, \dots, z_{j_{k_j}})$$

- c. $S: \bigcup_{k \geq 2} (X \times Y)^k \rightarrow X \times Y$ is a *superposition function* that combines the elements of a transformed tuple to create a new data sample.

A data superposition method is called *element-wise* or *local* when its superposition function is element-wise or local, respectively. When the choice function chooses tuples of fixed length k , the superposition method *operates* on k -tuples and it has a fixed *sample width* k .

In our experiments, we often use element-wise superposition methods that operate on pairs. For example, in section 8.7.6 we use such a pairwise method with an element-wise superposition function that computes the geometric mean of pixel values.

In general, data superposition adds coherence to a data set. This is because the result of a superposition is usually a sample that lies between the samples of its transformed tuple or along some extrapolating dimensions or directions relative to the samples. The result is often close to one or more of those samples and it represents a variant of them or a variant of an underlying concept. Consequently, if we use a metric to cluster the samples based on their distance, the superposition data set will tend to have reduced variance and increased coherence across or within its clusters.

*We should recall that a data set is defined as a binary relation in definition 5.6.

Data sets with increased coherence are useful because they tend to generate more coherent models. For example, when we use a superposition data set, the generated superposition samples bias the trained model and tend to make it more coherent around original data samples that have participated in superposition samples. Learning effectively transfers the increased coherence from the superposition data to the model. In a sense, learning is now biased in favor of small regions or neighborhoods of data samples instead of individual samples. Inevitably, coherence will decrease in certain highly transitional regions of the input space. However, this only makes the model more discriminant in those regions while it is more coherent on average.

Superposition methods will eventually lead to a different data approach in Mathesis. *Reflection* will introduce an important cognitive ability and learning process. It will blend boundaries around the concept of data and initiate a research paradigm that will reduce the role and necessity of persistent data in learning.

8.5 MNIST

The *MNIST data set* (aka *MNIST database*) is a handwritten decimal-digit data set that was generated by mixing data sets provided by NIST. It contains a training set and a test set of 60,000 and 10,000 images, respectively. Each image represents a single digit. The original black and white images were scaled to fit into a 28×28 pixel box and they were anti-aliased. In other words, the data set contains a total of 70,000 grayscale images of individual digits and each image is 28×28 pixels.

The MNIST data set is one of the smallest and most popular data sets in the history of neural networks and pattern recognition. It has been used by numerous students, researchers, and research groups since it was created in 1998. Its test error rate was gradually reduced and eventually diminished by neural networks and CNNs in particular. A brief summary and references to empirical studies of the data set can be found on its Wikipedia page.

In this section, we consider three such studies that arguably improved the state-of-the-art at the time of their publication. Our emphasis is on later studies because they are more modern and updated, and because our interests include synergy and synergetic effects.

The first study was published in 2012 and it claimed a test error rate of 0.23%. It used “elastic distortions” as a data *preprocessing-and-augmentation* step and trained an ensemble of 35 CNNs consisting of two convolution layers and two fully connected layers each. There was also some form of averaging of “columns”, which seem to be layers of the networks, and the overall system was called *Multi Column Deep Neural Network* (MCDNN) architecture. We will likewise refer to this study as MCDNN.

We have not attempted to reproduce the MCDNN study. There are some ambiguities in the definition of network ‘columns’ and in the data

preprocessing-and-augmentation steps. Elastic distortion and its parameters are neither defined nor referenced.

Moreover, it is not meaningful to reproduce a study where an experiment is performed only once and there are no measures of statistical significance or reliability of the final outcome. The study does include the test error rates of the 35 CNNs. Their average and standard deviation or confidence interval can easily be computed. However, those CNNs constitute a single ensemble. The 0.23% test error estimate is thus the result of a single ensemble and it is not reliable. The MNIST data set is so small and the test error so effectively diminished that there is considerable variance even across identical experiments with the dataset. If we perform such an experiment only once, we cannot be certain whether the result is a relatively accurate estimate of the error, a fluke, or an accident.

In that regard, the MCDNN paper is not an exception but rather an instance of an unfortunate practice in machine learning and neural network literature. The majority of such publications, especially publications in previous decades, do not provide any form of statistical significance. For generic methods and systems, there are often tests over multiple data sets but each experiment is usually performed only once. As a result, those publications report multiple results that are individually unreliable. Given that there is considerable *publication bias** in neural network research and significant variance of outcomes across identical experiments, such studies often lead to inaccurate or false claims.

It is not clear how this practice started. It might have been the result of expedience and limited computing power, especially in the 1990s or earlier. With current computing standards, it is easy to repeat an experiment multiple times. Especially for a data set as small as MNIST, there is simply no excuse for authors of studies not to do so. Even for larger data sets, where variance tends to be smaller, it is important to perform an experiment at least two or three times to reduce the probability of flukes, publication bias, and irreproducible results.

In general, it is important that we specify and upgrade evaluation standards in neural network research and practice, especially as the field becomes more mainstream and influential. As a start, empirical studies that do not provide any measurements of statistical significance or other measurements that establish some form of reliability of their results should automatically be rejected and withheld from publication until such measurements are provided. Neural network research is already in a critical stage. Popularity and commercial applications will increase pressure. Without proper standards, the field may drift and degrade to a deplorable state like that of medical research, clinical trials, or even vaccine studies.

*Publication bias is defined section 9.2, item F. It effectively represents a bias for publishing positive results, which collectively distort distributions and generate biased perceptions or estimates of a phenomenon or quantity. It is an important element and factor in *metascience*, which is the main topic of that section.

Let us now return to the MCDNN study. Our experiments with CNN ensembles for MNIST have shown that the test error can easily vary by an absolute difference of about 0.03% or more across identical training sessions. As a result, the 0.23% error rate that is reported by the MCDNN study could easily be anywhere in the interval [0.20%, 0.26%]. Without a confidence interval or some other form of statistical significance, we cannot tell how accurate or inaccurate the estimate is. Given the variance of our own experiments and the test errors of the seven five-column MCDNNs that are reported in the paper (0.37%, 0.26%, 0.32%, 0.33%, 0.31%, 0.26%, and 0.46%), it is likely that the 35-net MCDNN can reliably achieve a test error rate below 0.30% and often 0.26% or less. However, we do not feel that a stronger statement can safely be made without further empirical evidence.

A second study that used an ensemble of five similar CNNs and dropout or dropconnect appeared in 2013. The study used randomized affine transformations but no elastic distortions as a data preprocessing-and-augmentation step. It claimed a test error of 0.21% and declared that their approach “surpasses the state-of-the-art result of 0.23%.”

Once more, there were no measurements or estimates of statistical significance. The study performed the experiment three times, once in a plain form with no dropout or dropconnect, once with dropout only, and once with dropconnect only; and it reported test errors of 0.21%, 0.27%, and 0.21%, respectively.

Those results are as unreliable as the MCDNN result and they can as easily be flukes, accidents, or anything in between. The claim of the authors that their approach “surpasses the state-of-the-art” cannot be derived from their experiments. Their 0.21% error rate is as questionable and nonestablished as the supposed prior state-of-the-art with 0.23% error.*

We have attempted to repeat the experiments of the above paper in the plain form and with dropout only. There is some ambiguity in the paper. For example, in the beginning of section 6, the authors state that all their experiments use SGD with momentum, “the momentum parameter fixed at 0.9.” However, in section 6.1 they state that their learning method uses “no momentum.” We tried SGD with momentum and without it. The momentum version clearly wins but it does not achieve the claimed 0.21% error. We repeated the experiments at least 10 times each but the best we could get with SGD was around 0.25% on average for the plain version.

However, we must indicate that we implemented our solution in Pytorch. As we shall explain in section 8.6, there is an issue with the image transformations in Pytorch. It is probably a glitch or an unfortunate convention, and it can easily account for the observed differences. As a result, we cannot claim that the estimate of the above study is inaccurate. It is unreliable but that is

*Senior and well-known neural network researchers were involved in both of those studies. They should have advised their younger and less experienced colleagues to provide more reliable empirical evidence and show restraint in their claims.

all we can say.

Incidentally, there is an important empirical result in the above study that seems to have escaped notice by the authors. The dropout models outperform the plain models, as one might expect, but the ensemble of the plain models outperforms the dropout ensemble. This may seem to be a counterintuitive result. Or it may be assumed to be a fluke or an accident. Yet, it is neither.

We have repeatedly observed this phenomenon in more than 100 different experiments where we included dropout. If we add dropout to the networks of an ensemble or larger synergetic network, the networks usually perform better individually but their ensemble or synergetic network tends to perform worse. In other words, dropout improves individual networks but reduces synergy. This raises an obvious question. Why?

We believe that this is the first substantial evidence that supports the synergy conjecture. When we use dropout, we effectively select from a distribution of networks; and when we apply dropout to multiple identical or similar networks, the outcomes tend to regress toward the mean of the distribution. As a result, dropout models are more coherent and tend to generalize better individually, but they are more similar to each other. They are more mimetic, their diversity and dissent are reduced, and their synergy suffers. In other words, dropout tends to push models into the mimetic region of the synergy conjecture and thus reduce their synergy.

The above is not a verdict against dropout. It is rather a guidance that dropout should be used with caution and clear limits in synergetic networks. It also suggests that dropout should be applied to diverse parts of a synergetic network and perhaps with different settings and forms.

We can now consider the third MNIST study, which is not a publication but rather a reference to uploaded code on GitHub. The code achieves a largely reproducible test error rate of 0.17% and can truly claim to surpass or define the state-of-the-art for MNIST – at least, until the first results of Mathesis in section 8.7.

The associated model is an ensemble of 20 CNNs. Each CNN contains nine convolution layers, three *squeeze-and-excitation* (SE) blocks that will be defined shortly, and a final fully connected layer. We will refer to this solution as the *MNIST SE ensemble* or simply *SE ensemble*.

There is a randomization error in the code, but fortunately it only affects the range of the results and their confidence interval.

If we look at the results that are reported at the end of the code on GitHub, we can see that the SE ensemble was trained and tested fifteen times to obtain an average test error and thus a more reliable estimate of its performance. The average test error was 0.17%. However, we can also see that the ensemble test errors varied very little, from 0.16% to 0.18%, across the 15 experiments. This is suspicious. It is contrary to our experience and our many experiments with MNIST. The resulting 95% confidence interval is quite narrow and it is $0.17\% \pm 0.00468\%$.

After we examined the code of the SE ensemble in more detail, we found the cause of the above narrow intervals. The CNNs were randomly initialized at the beginning of each experiment but the randomization seed was reset to the same fixed value: the seed of CNN i was set to i , for $i \in \{0, 1, \dots, 19\}$. As a result, the CNNs were randomly initialized in each training session but they were identically initialized across the fifteen training sessions.

Such an initialization is strictly contraindicated. It can only increase the publication bias. For example, if different people or different trials involve the same experiment and each uses a different but fixed random initialization like the above, one of the experiments will inevitably have lower or equal error rate than the rest. Publishing such results or the best among them as the state-of-art will simply introduce a bias. And the more often that such a contraindicated initialization is used, the greater the bias in the final estimate.

A proper approach is to initialize the seed once using an irrelevant quantity such as system time or simply let the system proceed with its default initialization. A twofold initialization can be used to avoid potential correlations among training sessions that are issued in close temporal proximity. In that case, system time or the default initialization can be used to seed a random number generator, the generator can then produce one or more random integers uniformly in a large interval (such as $[1, 2^{32} - 1]$), and the random integer(s) can then be used to seed all random number generators that are used by the training session. However, we have found that system time or default initialization usually provide sufficiently random seeds. We have not noticed any measurable difference between those and the twofold seed initialization; but we used the twofold method in most of our experiments.

The SE ensemble is implemented in TensorFlow and the code that we used for the twofold random seed initialization is shown in figure 8.2.

Listing 8.2: Random seed initialization code.

```
@classmethod
def set_seed(cls, seed=None):
    """Sets the seed of various sources of randomness.
    Args:
        seed (integer): randomization seed. When None, a random seed
            in the interval [1, 0xffffffff] will be used. Default: None.
    Returns:
        The given or randomly chosen seed.
    """
    if seed is None:
        seed = random.randint(1, 0xffffffff)
    # Python
    random.seed(seed)
    environ['PYTHONHASHSEED'] = str(seed)
    # Numpy
    numpy.random.seed(seed)
    # TensorFlow
    tensorflow.random.set_seed(seed)
    # Return the seed
    return seed
```

The code starts with the default seed initialization of Python, which is arguably sufficient even for cryptography applications. It derives a random integer uniformly in the interval $[1, 2^{32} - 1]$ and treats it as a seed for the random modules that are used by the code, namely the Python, NumPy, and TensorFlow random modules. (The code also returns the random seed so that it can be recorded and the experiment can be repeated with precisely the same settings, if necessary.)

Using the above initialization, we trained and tested the twenty-model SE ensemble fifteen times. The test error rates now varied between 0.15% and 0.20% while the confidence interval was $0.17133\% \pm 0.00807\%$. Those numbers represent a 150% relative increase of the range and a 72.44% relative increase in the confidence interval length but only a 0.76% relative increase in the mean error rate. When we increased the number of training sessions to 32, the range remained the same but the confidence interval was $0.17094\% \pm 0.00443\%$. (Of course, confidence intervals tend to decrease with the number of trials.)

Those results suggest that the range of the MNIST SE ensemble test error is not $[0.16\%, 0.18\%]$ but probably $[0.15\%, 0.20\%]$. They also suggest that the average test error is slightly larger but indeed very close to 0.17%. For simplicity, and for the remaining of this chapter, we will assume the average test error and the state-of-the-art for MNIST to be 0.17%.

We can now describe the SE ensemble in more detail. As a first step, we formally define and (briefly) discuss SE blocks.

Definition 8.7. A *squeeze-and-excitation block*, or *SE block* for short, is a neural structure that attempts to aggregate and squeeze information across a dimension of its input in order to generate an array that will be used to discount the various elements of the input. Given input $X \in \mathbb{R}^{d_0, \dots, d_n}$, the block uses average pooling to reduce the input to an array $R \in \mathbb{R}^{d_j}$, for some $j \leq n$. The reduced (or squeezed) array R is then mapped to an array $Y = F_2(F_1(R)) \in [0, 1]^{d_j}$, where F_1 and F_2 correspond to two neural network layers with no cross-connections, affine polynomial transformations, and relu and logistic activation functions, respectively. It is assumed that the output of F_1 is an array in $\mathbb{R}^s$, where $s < d_j$ and often much smaller. Finally, the array Y is used to scale the input X identically across the other n dimensions. The final output is thus an array T such that $T_i = X_i Y_{i_j}$, for $i = (i_0, \dots, i_n) \in \mathbb{N}^{d_0, \dots, d_n}$.

Intuitively, an SE block aggregates (or squeezes) the information across n dimensions of its input by averaging them. It reduces the aggregated information further by its first layer, which decreases the size of the remaining dimension from d_j to s . The intention is to remove noise or insignificant detail from the original information and keep only prominent or otherwise important elements across the chosen dimension. Those elements are then expanded in the array Y to match the original size of the chosen dimension; and they are multiplied with the corresponding elements of the input X

so that they can discount them. The logistic activation function is used in F_2 so that the elements of Y will be in the interval $[0, 1]$ and can act as a discounting multiplier. The multiplication effectively creates a conjunctive relation between the input and information derived from its chosen dimension.

Technically, X and Y are inputs into the final layer of the SE block that come from cross-connections and successive connections, respectively. Their multiplication is simply the polynomial (but non-affine) function P of that layer.

In the original publication of squeeze-and-excitation networks, the authors defined the layers that correspond to F_1 and F_2 to be fully connected so that they can capture dependencies across the various elements of dimension j . However, this is not a sound argument. The layers can be locally connected and still capture such dependencies, if the dependencies are not dense. This is probably the case for most large networks or networks that are not minimal for a given application or data set. Full connectivity unnecessarily restricts the network because it often leads to configurations where the squeezed dimension size s is very small. We shall examine this issue in more detail in volume II.

SE blocks are usually used in CNNs, averaging over the spatial dimensions and squeezing the information of the filters. If we interpret the filters as features of the original data, SE blocks tend to retain important features in the data and discount or eliminate unimportant ones. This is particularly useful for computer vision and any application that involves images or video.

Let us now return to the MNIST SE model. Our code below is almost identical to the code that has been uploaded on GitHub except for some minor changes: we combined the convolutions and the SE block into a single function to make the code more succinct and we removed an unnecessary reshape operation from the SE block.

The code assumes that the functions `multiply` and `concatenate` and the classes `Input`, `Conv2D`, `BatchNormalization`, `Dense`, `AveragePooling2D`, `GlobalMaxPooling2D`, and `GlobalAveragePooling2D` have been imported from the module `tensorflow.keras.layers`. It also assumes that the class `Model` and the module `regularizers` have been imported from `tensorflow.keras`. Detailed information about all those functions and classes can be found in the TensorFlow and Keras API documentation. Their names are self-descriptive and their functionality should be obvious. For example `Conv2D` implements a two-dimensional convolution layer.

Listing 8.3: Squeeze-and-excitation (SE) CNN for MNIST.

```
def cse_block(x, filters=128, squeezed=4):
    """Convolution-squeeze-and-excitation (CSE) block.
    Args:
        x (array): input array.
        filters (integer): number of filters in the block.
            Default: 128.
        squeezed (integer): number of filters in the squeezed part
            of the block. Default: 4.
    Returns:
```

```

        The output of the block.
    """
    x = Conv2D(filters, (3,3), activation='relu', padding='same')(x)
    x = Conv2D(filters, (3,3), activation='relu', padding='same')(x)
    x = Conv2D(filters, (3,3), activation='relu', padding='same')(x)
    x = BatchNormalization()(x)
    y = GlobalAveragePooling2D()(x)
    y = Dense(squeezed, activation='relu')(y)
    y = Dense(filters, activation='sigmoid')(y)
    x = multiply([x, y])
    return x

def make_model(input_shape):
    """Make a model with three squeeze-and-excitation blocks.
    Args:
        input_shape (iterable): shape of the input array (excluding
            batch dimension).
    Returns:
        The generated squeeze-and-excitation model.
    """
    s = Input(shape=input_shape)
    x = cse_block(s)
    x = cse_block(x)
    x = AveragePooling2D(2)(x)
    x = cse_block(x)
    x = AveragePooling2D(2)(x)
    x = concatenate([GlobalMaxPooling2D()(x),
                     GlobalAveragePooling2D()(x)])
    x = Dense(10, activation='softmax', use_bias=False,
              kernel_regularizer=regularizers.l1(0.00025))(x)
    return Model(inputs=s, outputs=x)

```

The above SE model is as straightforward as it is concise and elegant. It involves three consecutive *convolution-squeeze-and-excitation* (CSE) blocks interleaved with two average-pooling functions. The output of the second pooling function is used by two global pooling functions that feed their results to a fully connected layer. Finally, the fully connected layer, which uses L_1 regularization to improve generalization and synergetic effects, produces ten softmax outputs, one for each decimal digit.

The CSE blocks are equally straightforward. They contain three two-dimensional convolution layers (with 3×3 kernels and relu activation functions) followed by a batch normalization layer and an SE block.

Finally, the SE ensemble uses 20 SE models and computes the average of their outputs.

Each SE model contains 1,189,516 parameters. The distribution of those parameters across their layers as well as the output dimensionality of each layer are shown in table 8.1. The 20-model SE ensemble does not contain any additional parameters other than the parameters of its SE models, which amount to 23,790,320 parameters total.

The SE models were trained on data preprocessed and augmented with random affine transformations. The data were initially transformed so that pixel values were in the symmetric interval $[-0.5, 0.5]$. A random affine transformation was then applied. The transformation combined a rotation with a random angle uniformly distributed in the interval $[-10^\circ, 10^\circ]$, a

Layer	Output Shape	Parameters
Input	(28, 28, 1)	
Conv2D	(28, 28, 128)	1,280
Conv2D	(28, 28, 128)	147,584
Conv2D	(28, 28, 128)	147,584
BatchNormalization	(28, 28, 128)	512
GlobalAveragePooling2D	(128)	
Dense	(4)	516
Dense	(128)	640
Multiply	(28, 28, 128)	
Conv2D	(28, 28, 128)	147,584
Conv2D	(28, 28, 128)	147,584
Conv2D	(28, 28, 128)	147,584
BatchNormalization	(28, 28, 128)	512
GlobalAveragePooling2D	(128)	
Dense	(4)	516
Dense	(128)	640
Multiply	(28, 28, 128)	
AveragePooling2D	(14, 14, 128)	
Conv2D	(14, 14, 128)	147,584
Conv2D	(14, 14, 128)	147,584
Conv2D	(14, 14, 128)	147,584
BatchNormalization	(14, 14, 128)	512
GlobalAveragePooling2D	(128)	
Dense	(4)	516
Dense	(128)	640
Multiply	(14, 14, 128)	
AveragePooling2D	(7, 7, 128)	
GlobalMaxPooling2D	(128)	
GlobalAveragePooling2D	(128)	
Concatenate	(256)	
Dense	(10)	2,560
Total parameters		1,189,516

Table 8.1: Output shapes and number of parameters for the layers of the MNIST SE model.

translation with a random shift relative to the width and height of the image and uniformly distributed in the interval $[-0.1, 0.1]$, a scaling with a random scale coefficient uniformly distributed in the interval $[0.8, 1.2]$, and a shear operation with a random angle chosen uniformly in the interval $[0^\circ, 10^\circ]$ and assumed to be in counter-clockwise direction.

The SE models were trained with Adam using batches of 32 samples and a learning schedule that involved thirteen epochs with learning rate 10^{-3} , three epochs with learning rate 10^{-4} , and three epochs with learning rate 10^{-5} . The schedule also contained one final epoch with learning rate 10^{-5} that trained the model on the original training data without the random affine transformations.

Given that the MNIST training set contains 60,000 samples, it corresponds to 1875 batches of 32 samples each. Consequently, the above training schedule contained $20 \cdot 1875 = 37500$ learning parameter updates, each involving 23,790,320 parameters, or a total of 892,137,000,000 parameter updates.*

The excellent performance of the MNIST SE ensemble derives from four main factors. The first is its sheer size. The ensemble is so large relative to the training set that it approximately contains one parameter for every two pixels in the training set. (The total number of pixels in the training set is $28 \cdot 28 \cdot 60000 = 47,040,000$.)

The second factor is the depth of the SE models. Each SE model contains three CSE blocks, three pooling layers (the two global pooling layers operate in parallel), a concatenation layer, and a fully connected layer. Given that each CSE block contains three convolution layers, a batch normalization layer, a pooling layer, two fully connected layers, and a multiply layer, each SE model has a depth of 29 layers. If we exclude layers that have no trainable parameters (such as the pooling, multiply, and concatenate layers) the network has a depth of 19 layers. Even if we count only the convolution and fully connected layers, the network has a depth of 16 layers.

The third factor for the above performance is synergy. There are 640 SE models in the 32 SE ensembles that we trained. Their test errors are distributed in the interval $[0.16\%, 0.37\%]$, the average test error is 0.25498%, and the 95% confidence interval is $0.25498\% \pm 0.00268\%$. In other words, the ensemble reduced the average error compared to its individual networks by about $1 - 0.17094/0.25498 = 32.96\%$ while it reduced the length of the range by $1 - (0.20\% - 0.15\%)/(0.37\% - 0.16\%) = 76.19\%$. Those are impressive reductions.

The last (but not least) factor is the SE blocks. They are elements and hints of a broader perspective into neural functionality. For they are a significant but relatively simple and straightforward instance of a general class of more elaborate neural structures that will be introduced in volume II. Those structures can capture complex relations between information elements such as filters or classifications, they can reduce the error of individual models, and they can truly thrive under synergy and coherence. The performance of the MNIST SE ensemble is only the first empirical evidence of their power

*There are few non-trainable parameters that correspond to the means and standard deviations of the batch normalization layers but they only represent about 0.065% of the total number of parameters. Their effect is thus negligible. If we do remove those parameters from the estimates, there will be a total of 23,774,960 trainable parameters in the SE ensemble and a total of 891,561,000,000 parameter updates during its training.

and significance.

8.6 Implementation Woes

We have implemented a considerable portion of Mathesis in Pytorch. We also implemented the MNIST SE ensemble in Pytorch so that we could add neural state and more elaborate synergy to it, use local networks such as flower nets, apply more coherent learning such as synaptic coherence or the Syc and Sync algorithms, and increase data coherence with data superposition.

However, we were not able to reproduce the MNIST results of the previous section with our Pytorch implementation. The best average test error rate we could get with an identical model and identical preprocessing, augmentation, and learning settings was about 0.21%. This represents a relative increase of 23.53% over the 0.17% average test error of the TensorFlow implementation. Even when we initialized the networks with the exact same weights, the results quickly diverged and the final difference remained the same.

We eventually traced the source of the above discrepancy to be the image transformations in the data augmentation steps. When we removed the transformation, the Pytorch and TensorFlow implementations of the SE ensemble produced virtually indistinguishable error rates. However, those augmentation steps are necessary because they prevent the SE models from fitting the MNIST training data. The 0.17% average test error cannot be achieved without them.

We tried various approaches (including tweaking the augmentation steps or using earlier versions of Pytorch) in our effort to make the two implementations comparable. However, we did not manage to reduce the difference significantly.

We believe that the above differences suggest a glitch in the implementation of the randomized affine transformations in the Torchvision package of Pytorch. Given that affine image transformations are not commutative (for example rotation followed by translation is not the same as translation followed by rotation), it is possible that the error discrepancy is the outcome of a convention in Torchvision that is unfortunate for optical character recognition. However, we feel that a glitch is a more likely explanation.

Our next step was to implement a small part of Mathesis in TensorFlow. We opted for flower nets and data superposition.

Flowers are easy to implement in Pytorch. Grouped convolutions partition the filters into equal size subsets and thus correspond to the corolla of a sunflower (or a sunflower with an empty core). As a result, sunflowers can be implemented with a combination of a regular and a grouped convolution for the core and the corolla, respectively. Flowers are more complex but they can be implemented with an operation that extracts blocks from a multidimensional array, such as the *unfolding* operation in Pytorch, a convolution on the unfolded array, and a convolution for the core of the flower, if any. This is the *parallel flower implementation* and it has time complexity $O(1)$ with

respect to the number of petals n if we assume that the unfolding operation also has time complexity $O(1)$. The *sequential flower implementation* uses a sequence of convolutions: one for each petal and one for the core. It has time complexity $O(n)$.^{*} As a result, the sequential implementation should not be used in any application, except perhaps at an initial development stage or for debugging purposes.

On the other hand, there are many issues and weaknesses in TensorFlow and Keras. Flowers only demonstrated some of them. The block-extracting operation is called *extract_patches* and it is part of the *image* package of TensorFlow. However, this operation is so slow that the parallel implementation of flowers in TensorFlow is more than five times slower than the sequential implementation for the cases we tried. Grouped convolutions have similar issues in Keras. We observed a nearly linear increase with the number of groups n . We suspect that grouped convolutions are sequentially implemented in Keras. They are not available in TensorFlow.

Consequently, we were forced to use the sequential implementation in our MNIST experiments with TensorFlow and limit the number of petals in our flowers. We use the terms *oligopetalous*[†] and *polypetalous* for flowers that have few or many petals, respectively. The generalization of our ensemble was probably reduced because we used oligopetalous flowers only. Diversity suggests that some polypetalous flowers should also be included. In general, there should be a distribution of flowers that covers both types. Coherence in functionality across a flower implies a small stride. Given that the common subset across all petals has size $\max(0, p - (n - 1)s)$, a small stride would imply a large common subset, if the number of petals is also small. Those are not necessarily poor flower configurations. However, if all flowers of a synergetic network are constrained as above, the network will most likely suffer.

Finally, we believe that more elaborate synergy and coherent learning can improve the results further. However, we will leave those, as well as a synergetic network with both oligopetalous and polypetalous flowers, as an exercise to our readers.

The MNIST test error is so small that reducing the error further is hard. Many useful approaches and architectures will produce a negligible difference, if at all. As the authors of the MCDNN paper noted, “this is the first time an artificial method comes close to the $\approx 0.2\%$ error rate of humans on this task. Many of the wrongly classified digits either contain broken or strange strokes, or have wrong labels.”

We largely agree with the MCDNN authors but not entirely. We examined the misclassified samples of the SE ensemble and found that about half of them could still be reliably classified. A handwriting recognition expert or a

^{*}The convolutions are independent of each other and they could be performed in parallel. However, current neural network frameworks do not support such forms of parallelism.

[†]From the Hellenic word *ὀλίγος* for “little” or “few”.

committee of humans could probably reduce the error close to 0.1%. Some of those samples require directional interpretations of strokes and a distinction between stroke endpoints such as a pen-lift, a pen-down, or an accidental lift that broke a stroke. Such expertise is hard to acquire through learning but it is not impossible.

In the next section, we will demonstrate that even a small part of Mathesis, namely an ensemble of oligopetalous flower nets with data superposition and few other modifications, can actually bring the average test error rate of MNIST close to 0.1%.

8.7 Mathesis for MNIST

Let us now develop a simple Mathesis solution for MNIST. We will use the SE ensemble as a starting point and gradually change it, making a total of seven different modifications based on Mathesis. We will first describe some relatively simpler (and probably less important) modifications. Then we will add data superposition to create a more coherent data set. Finally, we will train a diverse and more coherent ensemble on the coherent data set.

8.7.1 SE Block Extension

Our first modification of the SE model is a change in the SE block to include a linear term. The linear term will be an additional fully connected layer that is parallel with the second layer and produces an array B from the output of the first layer. In other words, $B = F_3(F_1(R))$. We call the new layer the *bias layer*. The final output of the new block is an array T such that $T_i = B_{i_j} + X_i Y_{i_j}$, for $i = (i_0, \dots, i_n) \in \mathbb{N}^{d_0, \dots, d_n}$. The code is shown below and it only includes one additional line.*

Listing 8.4: CSE block with a bias layer.

```
def cse_block(x, filters=128, squeezed=4):
    """Convolution-squeeze-and-excitation (CSE) block with bias.
    Args:
        x (array): input array.
        filters (integer): number of filters in the block.
            Default: 128.
        squeezed (integer): number of filters in the squeezed part
            of the block. Default: 4.
    Returns:
        The output of the block.
    """
    x = Conv2D(filters, (3,3), activation='relu', padding='same')(x)
    x = Conv2D(filters, (3,3), activation='relu', padding='same')(x)
    x = Conv2D(filters, (3,3), activation='relu', padding='same')(x)
```

*An obvious improvement is to replace the fully connected layers with locally connected layers and increase the size of the squeezed dimension. However, as we mentioned in section 8.5, the error of the MNIST SE ensemble is so small that such changes usually have negligible contributions. We leave it as an exercise to our readers to determine whether this is the case for local connectivity and perhaps the bias layer in the SE block.

```

x = BatchNormalization()(x)
y = GlobalAveragePooling2D()(x)
y = Dense(squeezed, activation='relu')(y)
b = Dense(filters, activation='relu')(y)
y = Dense(filters, activation='sigmoid')(y)
x = add(b, multiply([x, y]))
return x

```

8.7.2 Local Pooling

Our second modification of the SE model is to replace the two global pooling layers in the end of the SE model with a local average pooling layer that reduces the spatial information instead of removing it altogether. In that case, we also need to flatten out the array so that we can use it as input to the final fully connected layer. Specifically, we replace the following lines from listing 8.3

```

x = AveragePooling2D(2)(x)
x = concatenate([GlobalMaxPooling2D()(x),
                 GlobalAveragePooling2D()(x)])

```

with the code

```

x = AveragePooling2D(final_kernel, final_strides)(x)
x = Reshape((f*final_spatial_size))(x)

```

where `final_kernel`, `final_strides`, and `final_spatial_size` are now parameters of the model.

In our experiments, we used final pooling with 8×8 kernels and 3×3 strides. Given that the input to the final pooling operation is a $14 \times 14 \times 128$ array and there is no padding, the output is a $3 \times 3 \times 128$ array. In other words, our final pooling operation compressed the spatial information into a 3×3 array and the final spatial size is thus 9.

Given that the convolutions use padding and 3×3 kernels, each of the first six convolutions adds two padding pixels in each image dimension. Given that there is a 2×2 pooling after the sixth convolution, each of the remaining three convolutions adds four padding pixels in each dimension. Combining the above with the 28×28 size of the original images, we obtain that the padded images are 52×52 pixels. The receptive fields of the nine convolutions in the padded images have size $k \times k$, where $k = 3, 5, 7, 9, 11, 13, 17, 21, 25$; and the final pooling operation results in receptive fields of 39×39 pixels ($39 = 25 + 2 \cdot (8 - 1)$). Consequently, the nine blocks after the final pooling contain the pixels $\{(x, y) / 0 \leq x - 6i \leq 39 \text{ and } 0 \leq y - 6j \leq 39\}$, for $i, j \in \{0, 1, 2\}$. Given that the padded images have 12 padding pixels on each side, a 39×39 block cannot miss more than a single row and a single column from the original 28×28 image.

The above represents a conservative change of *perspective*.* A more aggressive change may work better for larger data sets, but it is unlikely that

*The concept of perspective is beyond the scope of this chapter. We will define and discuss it in volume II.

it will make a difference for MNIST – or any data set with such diminished error rate.

8.7.3 CSE Blocks with Dropout

Our third change of the SE model is to add optional dropout to the CSE blocks. We actually did not add any dropout to the first block but we added dropout layers with dropout probability $p = 0.1$ and $p = 0.2$ in the other two blocks. We also added a dropout layer with probability $p = 0.2$ before the final layer of the network. The updated code is shown below.

Listing 8.5: Dropout SE model for MNIST.

```
def cse_pd_block(x, filters=128, squeezed=4, dropout=0, pooling=0):
    """Convolution-squeeze-and-excitation (CSE) block with bias and
       optional pooling and dropout.
    Args:
        x (array): input array.
        filters (integer): number of filters in the block.
            Default: 128.
        squeezed (integer): number of filters in the squeezed part
            of the block. Default: 4.
        dropout (float): dropout probability. Default: 0.
        pooling (integer): pooling size. If zero, no pooling is used.
            Default: 0.
    Returns:
        The output of the block.
    """
    # Make the convolutions and their dropout, if any
    x = Conv2D(filters, (3,3), activation='relu', padding='same')(x)
    if dropout > 0:
        x = Dropout(dropout)(x)
        x = Conv2D(filters, (3,3), activation='relu',
                    padding='same')(x)
        x = Dropout(dropout)(x)
    else:
        x = Conv2D(filters, (3,3), activation='relu',
                    padding='same')(x)
    x = Conv2D(filters, (3,3), activation='relu', padding='same')(x)
    x = BatchNormalization()(x)
    # Make the SE block
    y = GlobalAveragePooling2D()(x)
    y = Dense(squeezed, activation='relu')(y)
    b = Dense(filters, activation='relu')(y)
    y = Dense(filters, activation='sigmoid')(y)
    x = add(b, multiply([x, y]))
    # Add pooling and dropout in the end
    if pooling > 0:
        x = AveragePooling2D(pooling)(x)
    if dropout > 0:
        x = Dropout(dropout)(x)
    return x

def make_model(input_shape):
    """Make a model with three squeeze-and-excitation blocks.
    Args:
        input_shape (iterable): shape of the input array (excluding
            batch dimension).
    Returns:
        The generated squeeze-and-excitation model.
```

```

"""
s = Input(shape=input_shape)
x = cse_pd_block(s)
x = cse_pd_block(x, dropout=0.1, pooling=2)
x = cse_pd_block(x, dropout=0.2)
x = AveragePooling2D(pool_size=(8, 8), strides=(3, 3))(x)
x = Reshape((128*9))(x)
x = Dropout(0.2)(x)
x = Dense(10, activation='softmax', use_bias=False,
          kernel_regularizer=regularizers.l1(0.00025))(x)
return Model(inputs=s, outputs=x)

```

The dropout values and their positions in the code are just a guess based on our experience with neural networks. They are not the outcome of any measurements or empirical estimates on MNIST and they are most likely suboptimal.

As we explained in section 8.5, dropout can have a negative effect when used across identical or similar networks within a larger synergetic network. Our intention here is to use within a diverse ensemble and across networks with different configurations.

8.7.4 Squeeze-and-Excitation Flower Nets

A fourth and more important change to the SE ensemble is to use local networks such as flower nets instead of CNNs. We can easily obtain such networks if we replace the convolution layers of the SE model with linear flowers. Given that the first convolution operates on pixels with a single numeric value, it corresponds to a trivial flower that has $f = p = n = 1$. This is just a regular convolution. As a result, our flower nets will contain an initial convolution followed by a sequence of eight flowers and three SE blocks.

We call such squeeze-and-excitation flower nets *SEF models* or *SEF networks*. The code of their specific instance for MNIST is shown in the listing below. The code assumes a generic class `Flower(f, p, n, s, c=0, **kwargs)` that generates a flower with a corolla of size `f`, `n` petals of size `p` each, stride `s` between the petals, and a core of size `c`. `kwargs` are keyword arguments for the underlying convolution, such as `kernel_size`, `strides`, `padding` etc.

Listing 8.6: A SEF network for MNIST.

```

def get_flower(f, p, n, s, c=0):
    """Create and return a flower with 3x3 kernels, padding,
       and relu activation functions.
    Args:
        f (integer): corolla size.
        p (integer): petal size.
        n (integer): number of petals.
        s (integer): petal stride.
        c (integer): flower core. Default: 0.
    Returns:
        The created flower.
    """

```

```

    return Flower(f, p, n, s, c, kernel_size=(3,3),
                  activation='relu', padding='same')(x)

def sef_block(x, flowers, squeezed=4, dropout=0.1, pooling=2):
    """Squeezed-and-excited flower block (SEF block) with bias
        and optional pooling and dropout.
    Args:
        x (array): input array.
        flowers (iterable): iterable of flower tuples (f, p, n, s, c).
        squeezed (integer): number of filters in the squeezed part
            of the block. Default: 4.
        dropout (float): dropout probability. Default: 0.1.
        pooling (integer): pooling size. If zero, no pooling is used.
            Default: 0.
    Returns:
        The output of the block.
    """
    # Make the flowers and their dropout, if any.
    last_flower = flowers[-1]
    if dropout > 0:
        for i in range(len(flowers)-1):
            x = get_flower(*flowers[i])(x)
            x = Dropout(dropout)(x)
        x = get_flower(*last_flower)(x)
    else:
        for flower in flowers:
            x = get_flower(*flower)(x)
    x = BatchNormalization()(x)
    # Make the SE block
    filters = last_flower[0] + last_flower[4]
    y = GlobalAveragePooling2D()(x)
    y = Dense(squeezed, activation='relu')(y)
    b = Dense(filters, activation='relu')(y)
    y = Dense(filters, activation='sigmoid')(y)
    x = add(b, multiply([x, y]))
    # Add pooling and dropout in the end
    if pooling > 0:
        x = AveragePooling2D(pooling)(x)
    if dropout > 0:
        x = Dropout(dropout)(x)
    return x

def make_model(input_shape, flowers):
    """Make a SEF model with three SEF blocks.
    Args:
        input_shape (iterable): shape of the input array (excluding
            batch dimension).
        flowers (iterable): iterable of flower tuples (f, p, n, s, c).
            It must contain at least nine flowers. If more than nine
            flowers are present, only the first nine are used.
    Returns:
        The generated SEF model.
    """
    s = Input(shape=input_shape)
    x = Conv2D(flowers[0][0], (3,3), activation='relu',
              padding='same')(s)
    x = sef_block(flowers[1:3])
    x = sef_block(flowers[3:6], dropout=0.1, pooling=2)
    x = sef_block(flowers[6:9], dropout=0.2)
    x = AveragePooling2D(pool_size=(8, 8), strides=(3, 3))(x)
    x = Reshape((128*9))(x)
    x = Dropout(0.2)(x)
    x = Dense(10, activation='softmax', use_bias=False,

```

```

        kernel_regularizer=regularizers.l1(0.00025))(x)
    return Model(inputs=s, outputs=x)

```

8.7.5 Diversity in the Ensemble

Diversity is our fifth change to the SE ensemble and it is as important as locality. It is combined with flower growth across the layers of each SEF model so that there is variety and diversity not only in the settings of the flowers but also in their growth.

We model flower growth through fair striders and strider multiplication. As we stated in section 8.3, we use fair striders with $r = 1$, $e = 0$, $a = \text{True}$, $\rho = 1$ and $\epsilon = 1$. Consequently, the number of petals is fixed in each flower net but the corolla, petal size, and stride change proportionally with the multiplier m .

We assume all flower cores to be zero. For simplicity, we drop the cores from the notation below and write (f, p, n, s) to indicate the fair strider $(f, p, n, s, 0)$. Given an initial fair strider (f_0, p_0, n_0, s_0) and an array of multipliers $[m_i/i \leq 8]$, we generate the array of striders $[(f_i, p_i, n_i, s_i)/i \leq 9]$ such that

$$(f_+, p_+, n_+, s_+) = m(f, p, n, s) : i. \quad (8.1)$$

Furthermore, we pick the multipliers so that

$$mf, ms \in \mathbb{N} \text{ and } n|mf : i. \quad (8.2)$$

Given that the striders are straight, mp is also an integer because $mp = m(f - (n - 1)s) = mf - (n - 1)ms : i$. As a result, equation (8.1) becomes

$$(f_+, p_+, n_+, s_+) = (mf, mp, n, ms) : i. \quad (8.3)$$

For an initial strider (f, p, n, s) , the multiplier arrays are chosen from the set

$$M_s = \{1^8\} \cup \{A_k/k = 1, 2, \dots, 8\} \cup \{B_{s,d}/d = 1, 2, 3\}$$

where

$$\begin{aligned} 1^8 &= [1, 1, 1, 1, 1, 1, 1, 1], \\ A_k &= [1, 1, \frac{k+1}{k}, 1, 1, \frac{k+2}{k}, 1, 1] \text{ and} \\ B_{s,d} &= [\frac{s+(k+1)d}{s+kd}/k = 0, 1, \dots, 7] \end{aligned}$$

and so that condition (8.2) is satisfied.

We constrained the corollas to be of the form $f = n_- \frac{(n-1)s}{\beta} : i$, where $\beta \in \{1, \dots, n-2\}$ and $\beta|(n-1)s : i$. We will explain this formula in more detail in volume II when we discuss alignment. At this point, we will only

observe that the segments and the petals are

$$\left. \begin{aligned} g &= f/n_- = \frac{(n-1)s}{\beta} \\ p &= f - (n-1)s = (n_- - \beta)g \end{aligned} \right\} : i.$$

Consequently, the above striders and the corresponding linear flowers are petal aligned but they are not stride aligned since $\frac{s}{g} = \frac{\beta}{n-1} < 1 : i$. Of course, if we allowed $\beta = n - 1 : i$, the flowers would be fully aligned for that value of β . In that case, it would be $p = (n_- - n + 1)g : i$. However, we opted for flowers with constant number of petals n ; and we excluded the case $\beta = n - 1$ because it corresponds to sunflowers, when n is fixed. Diversity suggests that we should have at least one sunflower in our ensemble. Yet, we chose to exclude sunflowers because we think they are unlikely to make a measurable difference for MNIST.

As a result, our flowers have the form

$$(f, p, n, s, c) = (ng, (n - \beta)g, n, s, 0) \quad : i \quad (8.4)$$

where $\beta \in \{1, 2, \dots, n - 2\}$ and $g = \frac{(n-1)s}{\beta} : i$.

In our SEF ensemble, we also limited petal strides to be small so that there is more overlap and increased structural (and functional) coherence across consecutive petals. We also limited the petal number n to be small because of the framework and implementation issues that were mentioned in section 8.6. More specifically, we chose $s \in \{2, 3, \dots, 9\}$ and $n \in \{3, 4, \dots, 7\}$.

In order to avoid very large or small flower nets, we constrained the number of parameters in each network to be between 999,000 and 1,300,000 parameters. We also constrained the corollas of the first and last flower layers so that $f_0 \geq 40$ and $f_8 \geq 165$ for growing flower nets.

Finally, we combined the above constraints and selected flower nets. The selection code is shown in the listing below. The code assumes a function `get_net_size(strider, final_pool=[8,3,9])` that computes the number of parameters of a SEF network defined by the given strider and final pooling parameters.

Listing 8.7: SEF model selection.

```
# Fair strider parameters
fair_kwa = dict(n_e=0, s_e=1, n_r=1, s_r=1, a_n=True, c_e=1)
# Strider selection parameter
min_fair_size = 999000
max_fair_size = 1300000
klm = [(1, 1, 1), (1, 2, 3), (2, 3, 4), (3, 4, 5), (4, 5, 6),
        (5, 6, 7), (6, 7, 8), (7, 8, 9), (8, 9, 10)]

# Auxiliary functions to generate multiplier lists
def ml(s, d=1, n=8) : return [(k+d)/k for k in range(s, s+n*d, d)]
def mk(k, l, m): return [1, 1, 1/k, 1, 1, m/l, 1, 1]

def get_fair_tuple(f, p, n, s, mul, **kwa):
    """Create a fair strider and apply a multiplier to it.
```

```

Args:
    f (integer): corolla.
    p (integer): petal size.
    n (integer): number of petals.
    s (integer): petal stride
    mul (iterable of floats): multipliers.
Returns:
    A tuple containing the generated strider, the size of the
    corresponding SEF network, initial flower parameters f, p,
    n, s (c is assumed to be 0), and the multiplier array mul.
"""
fs = FairStrider(f, p, n, s, c=0, **kwa)
for m in mul:
    fs.mul(m)
sz = get_net_size(fs)
return (fs, sz, f, p, n, s, mul)

def select_fair_tuples(petals, strides, mul_deltas,
    min_first_f, min_last_f, klm, min_size, max_size,
    strider_fun, **strider_kwa):
    """Select SEF models.
    Args:
        petals (iterable of integers): possible petal sizes.
        strides (iterable of integers): possible petal strides.
        mul_deltas (iterable of integers): list of parameters for
            multiplier arrays of derived by the function ml().
        min_first_f (integer): minimum corolla for the first layer.
        min_last_f (integer): minimum corolla for the last flower.
        klm (iterable of integers): list of parameters for
            multiplier arrays of derived by the function mk().
        min_size (integer): lower bound on the number of parameters
            in the SEF network.
        max_size (integer): upper bound on the number of parameters
            in the SEF network.
        strider_fun (callable): function to generate a strider.
        strider_kwa (dictionary): keyword parameters for the
            strider function.
    Returns:
        A list of tuples, each containing a strider, the size of
        the corresponding SEF network, initial flower parameters
        f, p, n, s (c is assumed to be 0), and the multiplier array
        that is used to generate the strider value and thus
        the flowers in the network.
    """
    # Initialize the results
    fair_list = []
    # Select models
    for n in petals:
        n1 = n - 1
        b_sup = n - 1
        for s in strides:
            n1s = n1 * s
            for b in range(1, b_sup):
                # Make sure b divides (n-1)*s
                if n1s % b != 0: continue
                # Set the corolla and the petal size
                f = n * n1s // b
                p = f - n1s
                # The initial corolla should be sufficiently large
                if f < min_first_f: continue
                # Use the first type of multipliers
                for k, l, m in klm:
                    # Check divisibility constraints

```



```

        if s % k != 0 or f % k != 0: continue
        fok = f // k
        if (fok * l) % n != 0: continue
        if (fok * m) % n != 0: continue
        # Set the multiplier and make the strider
        mul = mk(k, l, m)
        st = strider_fun(f, p, n, s, mul, **strider_kwa)
        # The final corolla should be sufficiently large
        if m > 1 and st[0][-1][0] < min_last_f: continue
        # The network should have permissible size
        if min_size < st[1] < max_size:
            fair_list.append(st)
    if n1 % b != 0: continue
    # Use the second type of multipliers
    for d in mul_deltas:
        # Set the multiplier and make the strider
        mul = ml(s, d)
        st = strider_fun(f, p, n, s, mul, **strider_kwa)
        # The final corolla should be sufficiently large
        if st[0][-1][0] < min_last_f: continue
        # The network should have permissible size
        if min_size < st[1] < max_size:
            fair_list.append(st)

    return fair_list

def make_models(input_size):
    """Make and return the models of the ensemble.
    Args:
        input_size (tuple): shape of the input array.
    Returns:
        A list with the generated models.
    """
    fair_list = select_fair_tuples(
        petals=range(2,8), strides=range(2,10),
        mul_deltas=range(1,4),
        min_first_f=40, min_last_f=165, klm=klm,
        min_size=min_fair_size, max_size=max_fair_size,
        strider_fun=get_fair_strider, **fair_kwa)
    # Sort the fair list using the initial stride s and the
    # number of petals n
    fair_list.sort(key=lambda x: (x[5], x[4]))
    # Find and delete the last model with n=7 and s=8
    j = 0
    for i, (_fs, _sz, _f, _p, n, s, _mul) in enumerate(fair_list):
        if n == 7 and s == 8:
            j = i
    del fair_list[j]
    # Make and return the models
    return [make_model(input_size, fs_tuple[0])
            for fs_tuple in fair_list]]

```

In the above code, the call to the function `select_fair_tuples()` generates an ensemble with 21 networks and 24,568,551 parameters total. This is about 3.27% larger than the SE ensemble in terms of parameters and 5% larger in terms of models. In order to make the comparison between the two ensembles fair, we removed one network from the SEF ensemble, and that was the last network with $n_0 = 7$ and $s_0 = 8$.^{*} The final SEF ensemble thus

^{*}In our experiments, we implemented a selection algorithm that uniformly selected models across pairs (n, s) and used it to pick 20 models. For simplicity, we manually removed the extra model in the code of listing 8.7. The outcome is the same SEF ensemble

contains 20 models and 23,554,743 parameters total. It is thus comparable to the SE ensemble (it contains the same number of models and it is about 1% smaller in terms of parameters.)

Table 8.2 shows the number of parameters for each selected SEF network and its constant β value. (The β values are constant among the flowers of each SEF network because of equations 8.3 and 8.4.) The table also shows

Net	Parameters	β	f_0	p_0	n_0	s_0	Mult.	f_8	p_8	n_8	s_8
1	1,058,034	1	84	72	7	2	A_2	168	144	7	4
2	1,274,412	1	60	48	5	3	$B_{3,1}$	220	176	5	11
3	1,256,484	2	63	45	7	3	$B_{3,1}$	231	165	7	11
4	1,172,568	3	42	24	7	3	$B_{3,2}$	266	152	7	19
5	1,288,884	1	72	54	4	6	A_1	216	162	4	18
6	1,148,187	2	126	90	7	6	A_6	168	120	7	8
7	1,005,708	1	84	63	4	7	$B_{7,1}$	180	135	4	15
8	1,151,652	1	140	112	5	7	$\mathbb{1}^8$	140	112	5	7
9	1,090,272	2	70	42	5	7	$B_{7,2}$	230	138	5	23
10	1,135,146	2	147	105	7	7	$\mathbb{1}^8$	147	105	7	7
11	1,046,316	3	98	56	7	7	$B_{7,1}$	210	120	7	15
12	1,209,180	1	96	72	4	8	A_2	192	144	4	16
13	1,186,500	1	96	72	4	8	$B_{8,1}$	192	144	4	16
14	1,276,572	2	80	48	5	8	A_1	240	144	5	24
15	1,225,812	2	80	48	5	8	$B_{4,1}$	240	144	5	24
16	1,128,342	2	120	80	6	8	$A_{4,1}$	180	120	6	12
17	1,257,492	3	112	64	7	8	A_2	224	128	7	16
18	1,234,056	3	112	64	7	8	$B_{8,1}$	224	128	7	16
19	1,180,992	1	108	81	4	9	A_3	180	135	4	15
20	1,228,134	3	126	72	7	9	A_3	210	120	7	15
23,554,743											

Table 8.2: Number of parameters of the SEF networks along with their β values, their initial and final flowers, and their multiplier arrays.

the multipliers that defined flower growth in the networks and the initial and final flowers of the networks.

Table 8.3 shows the strider parameters for each flower in the SEF ensemble. Those parameters uniquely define the ensemble. We have omitted the core c because it is always zero. Each array $[f, p, n, s]$ indicates a strider setting and the corresponding linear flower. When the same settings are used in three consecutive layers, we use the notation $[f, p, n, s] \times 3$.

in both cases.

Net	$\begin{bmatrix} f_i & p_i & n_i & s_i \end{bmatrix}$ $i = 0, 1, 2$	$\begin{bmatrix} f_i & p_i & n_i & s_i \end{bmatrix}$ $i = 3, 4, 5$	$\begin{bmatrix} f_i & p_i & n_i & s_i \end{bmatrix}$ $i = 6, 7, 8$
1	$\begin{bmatrix} 84 & 72 & 7 & 2 \end{bmatrix} \times 3$	$\begin{bmatrix} 126 & 108 & 7 & 3 \end{bmatrix} \times 3$	$\begin{bmatrix} 168 & 144 & 7 & 4 \end{bmatrix} \times 3$
2	$\begin{bmatrix} 60 & 48 & 5 & 3 \end{bmatrix}$	$\begin{bmatrix} 120 & 96 & 5 & 6 \end{bmatrix}$	$\begin{bmatrix} 180 & 144 & 5 & 9 \end{bmatrix}$
3	$\begin{bmatrix} 80 & 64 & 5 & 4 \end{bmatrix}$	$\begin{bmatrix} 140 & 112 & 5 & 7 \end{bmatrix}$	$\begin{bmatrix} 200 & 160 & 5 & 10 \end{bmatrix}$
4	$\begin{bmatrix} 100 & 80 & 5 & 5 \end{bmatrix}$	$\begin{bmatrix} 160 & 128 & 5 & 8 \end{bmatrix}$	$\begin{bmatrix} 220 & 176 & 5 & 11 \end{bmatrix}$
5	$\begin{bmatrix} 63 & 45 & 7 & 3 \end{bmatrix}$	$\begin{bmatrix} 126 & 90 & 7 & 6 \end{bmatrix}$	$\begin{bmatrix} 189 & 135 & 7 & 9 \end{bmatrix}$
6	$\begin{bmatrix} 84 & 60 & 7 & 4 \end{bmatrix}$	$\begin{bmatrix} 147 & 105 & 7 & 7 \end{bmatrix}$	$\begin{bmatrix} 210 & 150 & 7 & 10 \end{bmatrix}$
7	$\begin{bmatrix} 105 & 75 & 7 & 5 \end{bmatrix}$	$\begin{bmatrix} 168 & 120 & 7 & 8 \end{bmatrix}$	$\begin{bmatrix} 231 & 165 & 7 & 11 \end{bmatrix}$
8	$\begin{bmatrix} 42 & 24 & 7 & 3 \end{bmatrix}$	$\begin{bmatrix} 126 & 72 & 7 & 9 \end{bmatrix}$	$\begin{bmatrix} 210 & 120 & 7 & 15 \end{bmatrix}$
9	$\begin{bmatrix} 70 & 40 & 7 & 5 \end{bmatrix}$	$\begin{bmatrix} 154 & 88 & 7 & 11 \end{bmatrix}$	$\begin{bmatrix} 238 & 136 & 7 & 17 \end{bmatrix}$
10	$\begin{bmatrix} 98 & 56 & 7 & 7 \end{bmatrix}$	$\begin{bmatrix} 182 & 104 & 7 & 13 \end{bmatrix}$	$\begin{bmatrix} 266 & 152 & 7 & 19 \end{bmatrix}$
11	$\begin{bmatrix} 72 & 54 & 4 & 6 \end{bmatrix} \times 3$	$\begin{bmatrix} 144 & 108 & 4 & 12 \end{bmatrix} \times 3$	$\begin{bmatrix} 216 & 162 & 4 & 18 \end{bmatrix} \times 3$
12	$\begin{bmatrix} 126 & 90 & 7 & 6 \end{bmatrix} \times 3$	$\begin{bmatrix} 147 & 105 & 7 & 7 \end{bmatrix} \times 3$	$\begin{bmatrix} 168 & 120 & 7 & 8 \end{bmatrix} \times 3$
13	$\begin{bmatrix} 84 & 63 & 4 & 7 \end{bmatrix}$	$\begin{bmatrix} 120 & 90 & 4 & 10 \end{bmatrix}$	$\begin{bmatrix} 156 & 117 & 4 & 13 \end{bmatrix}$
14	$\begin{bmatrix} 96 & 72 & 4 & 8 \end{bmatrix}$	$\begin{bmatrix} 132 & 99 & 4 & 11 \end{bmatrix}$	$\begin{bmatrix} 168 & 126 & 4 & 14 \end{bmatrix}$
15	$\begin{bmatrix} 108 & 81 & 4 & 9 \end{bmatrix}$	$\begin{bmatrix} 144 & 108 & 4 & 12 \end{bmatrix}$	$\begin{bmatrix} 180 & 135 & 4 & 15 \end{bmatrix}$
16	$\begin{bmatrix} 140 & 112 & 5 & 7 \end{bmatrix} \times 3$	$\begin{bmatrix} 140 & 112 & 5 & 7 \end{bmatrix} \times 3$	$\begin{bmatrix} 140 & 112 & 5 & 7 \end{bmatrix} \times 3$
17	$\begin{bmatrix} 70 & 42 & 5 & 7 \end{bmatrix}$	$\begin{bmatrix} 130 & 78 & 5 & 13 \end{bmatrix}$	$\begin{bmatrix} 190 & 114 & 5 & 19 \end{bmatrix}$
18	$\begin{bmatrix} 90 & 54 & 5 & 9 \end{bmatrix}$	$\begin{bmatrix} 150 & 90 & 5 & 15 \end{bmatrix}$	$\begin{bmatrix} 210 & 126 & 5 & 21 \end{bmatrix}$
19	$\begin{bmatrix} 110 & 66 & 5 & 11 \end{bmatrix}$	$\begin{bmatrix} 170 & 102 & 5 & 17 \end{bmatrix}$	$\begin{bmatrix} 230 & 138 & 5 & 23 \end{bmatrix}$
20	$\begin{bmatrix} 147 & 105 & 7 & 7 \end{bmatrix} \times 3$	$\begin{bmatrix} 147 & 105 & 7 & 7 \end{bmatrix} \times 3$	$\begin{bmatrix} 147 & 105 & 7 & 7 \end{bmatrix} \times 3$
21	$\begin{bmatrix} 98 & 56 & 7 & 7 \end{bmatrix}$	$\begin{bmatrix} 140 & 80 & 7 & 10 \end{bmatrix}$	$\begin{bmatrix} 182 & 104 & 7 & 13 \end{bmatrix}$
22	$\begin{bmatrix} 112 & 64 & 7 & 8 \end{bmatrix}$	$\begin{bmatrix} 154 & 88 & 7 & 11 \end{bmatrix}$	$\begin{bmatrix} 196 & 112 & 7 & 14 \end{bmatrix}$
23	$\begin{bmatrix} 126 & 72 & 7 & 9 \end{bmatrix}$	$\begin{bmatrix} 168 & 96 & 7 & 12 \end{bmatrix}$	$\begin{bmatrix} 210 & 120 & 7 & 15 \end{bmatrix}$
24	$\begin{bmatrix} 96 & 72 & 4 & 8 \end{bmatrix} \times 3$	$\begin{bmatrix} 144 & 108 & 4 & 12 \end{bmatrix} \times 3$	$\begin{bmatrix} 192 & 144 & 4 & 16 \end{bmatrix} \times 3$
25	$\begin{bmatrix} 96 & 72 & 4 & 8 \end{bmatrix}$	$\begin{bmatrix} 132 & 99 & 4 & 11 \end{bmatrix}$	$\begin{bmatrix} 168 & 126 & 4 & 14 \end{bmatrix}$
26	$\begin{bmatrix} 108 & 81 & 4 & 9 \end{bmatrix}$	$\begin{bmatrix} 144 & 108 & 4 & 12 \end{bmatrix}$	$\begin{bmatrix} 180 & 135 & 4 & 15 \end{bmatrix}$
27	$\begin{bmatrix} 120 & 90 & 4 & 10 \end{bmatrix}$	$\begin{bmatrix} 156 & 117 & 4 & 13 \end{bmatrix}$	$\begin{bmatrix} 192 & 144 & 4 & 16 \end{bmatrix}$
28	$\begin{bmatrix} 80 & 48 & 5 & 8 \end{bmatrix} \times 3$	$\begin{bmatrix} 160 & 96 & 5 & 16 \end{bmatrix} \times 3$	$\begin{bmatrix} 240 & 144 & 5 & 24 \end{bmatrix} \times 3$
29	$\begin{bmatrix} 80 & 48 & 5 & 8 \end{bmatrix}$	$\begin{bmatrix} 140 & 84 & 5 & 14 \end{bmatrix}$	$\begin{bmatrix} 200 & 120 & 5 & 20 \end{bmatrix}$
30	$\begin{bmatrix} 100 & 60 & 5 & 10 \end{bmatrix}$	$\begin{bmatrix} 160 & 96 & 5 & 16 \end{bmatrix}$	$\begin{bmatrix} 220 & 132 & 5 & 22 \end{bmatrix}$
31	$\begin{bmatrix} 120 & 72 & 5 & 12 \end{bmatrix}$	$\begin{bmatrix} 180 & 108 & 5 & 18 \end{bmatrix}$	$\begin{bmatrix} 240 & 144 & 5 & 24 \end{bmatrix}$
32	$\begin{bmatrix} 120 & 80 & 6 & 8 \end{bmatrix} \times 3$	$\begin{bmatrix} 150 & 100 & 6 & 10 \end{bmatrix} \times 3$	$\begin{bmatrix} 180 & 120 & 6 & 12 \end{bmatrix} \times 3$
33	$\begin{bmatrix} 112 & 64 & 7 & 8 \end{bmatrix} \times 3$	$\begin{bmatrix} 168 & 96 & 7 & 12 \end{bmatrix} \times 3$	$\begin{bmatrix} 224 & 128 & 7 & 16 \end{bmatrix} \times 3$
34	$\begin{bmatrix} 112 & 64 & 7 & 8 \end{bmatrix}$	$\begin{bmatrix} 154 & 88 & 7 & 11 \end{bmatrix}$	$\begin{bmatrix} 196 & 112 & 7 & 14 \end{bmatrix}$
35	$\begin{bmatrix} 126 & 72 & 7 & 9 \end{bmatrix}$	$\begin{bmatrix} 168 & 96 & 7 & 12 \end{bmatrix}$	$\begin{bmatrix} 210 & 120 & 7 & 15 \end{bmatrix}$
36	$\begin{bmatrix} 140 & 80 & 7 & 10 \end{bmatrix}$	$\begin{bmatrix} 182 & 104 & 7 & 13 \end{bmatrix}$	$\begin{bmatrix} 224 & 128 & 7 & 16 \end{bmatrix}$
37	$\begin{bmatrix} 108 & 81 & 4 & 9 \end{bmatrix} \times 3$	$\begin{bmatrix} 144 & 108 & 4 & 12 \end{bmatrix} \times 3$	$\begin{bmatrix} 180 & 135 & 4 & 15 \end{bmatrix} \times 3$
38	$\begin{bmatrix} 126 & 72 & 7 & 9 \end{bmatrix} \times 3$	$\begin{bmatrix} 168 & 96 & 7 & 12 \end{bmatrix} \times 3$	$\begin{bmatrix} 210 & 120 & 7 & 15 \end{bmatrix} \times 3$

Table 8.3: Flower parameters for each flower in the SEF ensemble.

8.7.6 Data Coherence

Our sixth and perhaps most important change to the SE ensemble is increased data coherence through data superposition.

We have retained the randomized affine transformations that are used in the SE ensemble for data augmentation and with precisely the same settings.

This makes for a fairer comparison, too. However, we have changed the initial transformation of the data to be in the interval $[0, 1]$ instead of $[-0.5, 0.5]$ to simplify the superposition formula. (0 represents the pixel value of the white background of the handwritten digits.)

Our superposition method simply picks pairs of samples from the training set that have the same classification and correspond to images with the same minimum bounding box; then it computes the element-wise pixel-level geometric mean of the two images.

Formally, we assume that X is a set of images and Y is a set of classifications for such images. If $D = \{(x_i, y_i) \in X \times Y / i \in \mathbb{N}_n\}$ represents our training set and B is a function that returns the minimum bounding box of the non-zero pixels of an image then, per definition 8.6, our superposition method is a tuple (C, T, S) such that

$$\begin{aligned} C(D) &= [(x_i, x_j)/i, j \in \mathbb{N}_n, i \neq j, B(x_i) = B(x_j), y_i = y_j], \\ T((x, s)) &= (x, s) \text{ for all } x, s \in X \\ S((x, y), (s, y)) &= (\sqrt{xs}, y) \text{ for all } x, s \in X \text{ and } y \in Y \end{aligned}$$

where the square-root and multiplication in the formula of the superposition function are assumed to be element-wise. In other words, $(\sqrt{xs})_\alpha = \sqrt{x_\alpha s_\alpha}$ for all $\alpha \in x_*$.

The above superposition method is relatively simple and limited. This is reflected by the fact that its transformation T is the identity function.

We can improve our method if we use a more elaborate transformation function. For example, we can remove the condition $B(x_i) = B(x_j)$ from the definition of the function C but change the transformation function T so that it scales and translates the second image s to make its minimum bounding box coincide with the minimum bounding box of x . Consequently, if there are c classes in the training set and there are n_k samples in each class k , the size of $C(D)$ will be bound from below by*

$$|C(D)| \geq \sum_k \binom{n_k}{2} = \frac{1}{2} \sum_k n_k^2 - \frac{n}{2} \geq \frac{n^2}{2c} - \frac{n}{2}$$

since $\sum_k n_k = n$. Furthermore, if there is a considerable variance among minimum bounding boxes (so that a great majority of image pairs have different boxes) and the classes are uniformly distributed (so that $n_k \simeq \frac{n}{c}$) it would be $|C(D)| \simeq \frac{n^2}{c} - n$.

Given that the number of classes c is usually much smaller than the number of training samples n , the above superposition can generate a quadratic amount of data. If we furthermore use arbitrary sample width instead of

*The bound is conservative. It does not account for the fact that the transformation function T is asymmetric when the images have different minimum bounding boxes. If such differences occur with probability p in a data set, a better bound will be about $1 + p$ times larger for that set.

using pairs only, the generated data will grow exponentially.*

In general, quadratic growth is more than enough in most cases. For example, $n = 60,000$ and $c = 10$ for MNIST. As a result, $|C(D)| \geq 179,970,000$. If we assume a considerable minimum-bounding-box variance and a uniform distribution of classes, it will be $|C(D)| \simeq 360,977,374$. If we calculate the size exactly,[†] it is $|C(D)| = 357,180,141$. This is clearly more data than we need.

If we use our simple superposition method that requires identical minimum bounding boxes, the numbers are much smaller but there are still 7,594,466 distinct pairs of samples in the training set that can be used for superposition. Table 8.4 shows the distribution of samples and superposition pairs across the 10 classes of MNIST. Given that there is a large number of pairs in each

Class	Samples	Pairs
0	5923	585,447
1	6742	576,021
2	5958	251,913
3	6131	362,743
4	5842	265,031
5	5421	198,296
6	5918	453,932
7	6265	282,220
8	5851	363,815
9	5949	457,815
	60,000	7,594,466

Table 8.4: Samples and pairs with identical minimum bounding boxes for each class in the MNIST training set.

class, we opted for our simple superposition method for MNIST.

On the other hand, our superposition method is simple and limited not only because of its approach to bounding boxes and its lack of data transformations but also because of its superposition function. If the background of handwriting images is represented with pixels that have value zero, the element-wise geometric mean of such images will tend to remove detail and segments from the images, effectively performing a real-valued intersection. Such an operation can remove or reduce embellishments, unimportant information, or noise from the data but it cannot address issues such as missing

*With arbitrary sample width, we can get the following exponential lower bound: $|C(D)| \geq \sum_k \sum_{w=2}^{n_k} \binom{n_k}{w} = \sum_k (2^{n_k} - n_k - 1) = \sum_k 2^{n_k} - n - c \geq c2^{n/c} - n - c$.

[†]If n_k indicates the number of samples in class k and $n_{k,b}$ indicates the number of samples of class k that have minimum bounding box b then $|C(D)| = \sum_k n_k(n_k - 1) - \frac{1}{2} \sum_{k,b} n_{k,b}(n_{k,b} - 1) = \sum_k n_k^2 - \frac{1}{2} \sum_{k,b} n_{k,b}^2 - \frac{n}{2}$.

or accidentally broken strokes. On the contrary, it can potentially proliferate such issues. It can address the problem indirectly – with training bias so that the model can adapt to such samples – but it cannot repair the samples themselves. This is an important limitation and a source of substantial bias.

Consequently, if we are to use our simple superposition method on the MNIST data set, it should be with some restraint. We cannot expect to use millions of such samples and reduce the MNIST error.

We will introduce advanced superposition methods in volume II. Those methods will serve multiple purposes and provide more complex and sophisticated functionality. They will be largely probabilistic and draw samples from distributions creating various combinations of data, transformations, and superposition functions. Their most important aspect is arguably the fact that they will eventually lead us to reflection. However, we will not discuss those topics further in this chapter, as they are beyond the scope of volume I.

For our current experiment with MNIST, we will use our simple superposition method on the training set to generate a limited amount of superposition samples. We will then combine those samples with the original training set and train our SEF ensemble. We expect to see some non-trivial gains in the end.

In order to limit the number of superposition samples, we use a selection method. For simplicity, we picked a straightforward method that enumerates the superposition pairs within each class and minimum bounding box and enumerates the boxes as string keys in a Python dictionary that maps boxes to samples. Our method then selects the first 10,000 pairs from each class.* Therefore, there is a total of 100,000 superposition samples added to the training set. This choice implies that the new training set contains 160,000 samples with a 5-to-3 ratio between superposition and original samples. There is nothing special or optimal about the above parameters – except that the ratio numbers are consecutive Fibonacci numbers and they consequently approximate the *golden ratio* $\frac{1+\sqrt{5}}{2}$. However, we do not have any evidence that the golden ratio is an appropriate ratio in this case. It is only a guess.

8.7.7 Training and Testing the SEF Ensemble

Our seventh and last modification of the SE ensemble applies to the training the ensemble.

Our new training set is about 2.67 times larger because it contains the superposition samples. We chose to increase our batch size, and we defined it to be 80 samples. Given that there is a 5-to-3 ratio between superposition and original samples, our new batches will contain 50 superposition and 30 original samples on average. (This is comparable to the 32 original samples in the training batch of the SE ensemble.)

*Formally, we assume a rank function $R: X^2 \rightarrow \mathbb{N}$ for pairs of images and we change the definition of the function C to be $C(D) = [(x_i, x_j)/i, j \in \mathbb{N}_n, i \neq j, B(x_i) = B(x_j), y_i = y_j, \text{ and } R(x_i, x_j) < 10,000]$.

Our training set now represents $160,000/80 = 2000$ batches, which is somewhat larger than the $60,000/32 = 1875$ batches in the original training set. The configurations are different and they are not necessarily comparable. The majority of our samples are now superposition samples.

In the end, we have to pick epochs and learning rates and define a learning schedule. We are not aware of any method or rationale behind the training schedule of the SE ensemble. We cannot evaluate its merits.

A validation set can always be used to determine a good learning schedule. This is harder for MNIST because the error is so small, many training sessions will be required for a reliable result, but it is possible.

In our case, and because our goal is to approach 0.1% error rate, we defined the schedule based on the training error. We trained with learning rate 10^{-3} until the training error rate of the SEFs was reliably near but above 0.50%; then we trained with learning rate 10^{-4} until the error rate was about 0.25%; and finally we trained with learning rate 10^{-5} until the training error was about 0.20%. This gave us a schedule of 8, 2, and 2 epochs, respectively. As in the learning schedule of the SE ensemble, we added training on the original and plain (non-augmented) data with learning rate 10^{-5} in the end. However, instead of using a small batch size and a single training epoch on the plain data, we used a batch size of 1000 samples and trained for 20 epochs. This choice led to a training error of about 0.15% for each individual SEF net at the end of its training. Table 8.5 shows our schedule in detail.

Training data	Learning rate	Epochs	Batch size	Batches per epoch	Updates
S+A	10^{-3}	8	80	2000	16000
S+A	10^{-4}	2	80	2000	4000
S+A	10^{-5}	2	80	2000	4000
Plain	10^{-5}	20	1000	60	1200

Table 8.5: Learning schedule for the SEF ensemble. S+A indicates a combination of superposition data and data augmentation.

We have not implemented any of our coherent learning methods (such as Syc or Sync) in TensorFlow. We have no plans to do so either. Development of Mathesis will not continue in TensorFlow. As a result, we were forced to use Adam. It is not a poor choice, but it is not a great choice either.

In our first experiment, we trained the SEF ensemble and measured the test error in the end. We repeated the experiment 10 times and we computed the average test error and the 95% confidence interval. The result was remarkable. The average error was 0.113% and the confidence interval was $0.113\% \pm 0.00679\% = [0.10621\%, 0.11979\%]$. If this is an accurate estimate, it represents a dramatic improvement of $1 - 0.113\%/0.17\% = 33.53\%$ over

the state-of-the-art.

Given that the error rate is so small we decided to repeat the experiment 10 more times just in case this was somehow a fluke. The average error turned out to be 0.117% the second time and the confidence interval was $0.117\% \pm 0.01069\% = [0.0010631, 0.0012769]$. The considerably larger width for the confidence interval suggested that there might have been a bit of fortunate chance in the first experiment.

In order to remove all doubt and convince ourselves – as well as observe how significance varies with the number of trials – we trained and tested the SEF ensemble 60 more times and combined all the results. A plot of the average test error and the confidence intervals as we increased the number of trials is shown in figure 8.6. The first ensembles had error close to 0.11% but

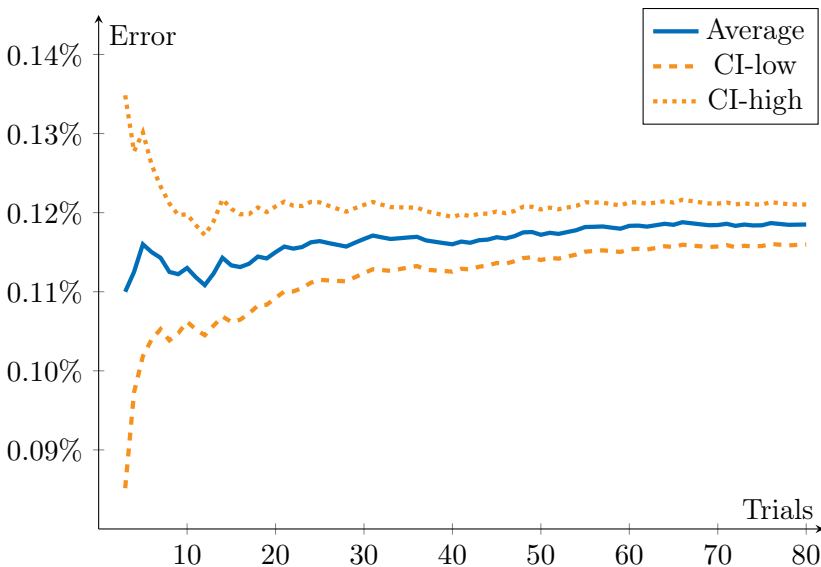


Figure 8.6: Average test error and confidence intervals of the SEF ensemble on MNIST.

their average eventually converged to $0.11850\% \pm 0.00253\%$. This represents a relative 7.73% increase from the initial ones and it demonstrates how sensitive error estimates are for the MNIST data set. When the error rate is so small, training a model or an ensemble only once or a few times is not sufficient. For the MNIST data set and the above level of error about 24 trials seem to be required to be within 2% of convergence and about 48 trials to be within 1%.

Of course, most data sets do not require so many trials. The MNIST data set is special because it is so small and the error almost trivial. However, training and testing a model once or twice is probably insufficient in most cases.

We believe that we have clearly established our result and have dramatically improved the state-of-the-art for MNIST. Our final error estimate of

0.1185% represents a 30.29% improvement over the previous state-of-the-art that was achieved by the SE ensemble.

We have not estimated the individual contribution of each component and method in our system. In order to do so, we should use the error rate of a single SE model as a baseline and estimate how much better our model becomes with each of our changes. We have not done a sufficient number of such experiments and we cannot provide accurate estimates.

However, our main experiment includes 80 SEF ensembles and thus 1600 individual SEF networks. The test errors of those networks are in the interval $[0.11\%, 0.28\%]$, their average is 0.19204%, and their 95% confidence interval is $0.19204\% \pm 0.00141\%$. This is the lowest that any individual model has ever achieved on MNIST. Given that the error of a single SE network is 0.25498%, the SEF network represents $(0.25498 - 0.19204) / (0.25498 - 0.11850) = 46.12\%$ of the gains over a single SE network. The remaining 53.88% of the gains are thus the result of synthesis and synergy. Moreover, all of our experiments with individual models indicated that the majority of the gains of such models derive from coherence.

Overall, it seems that synergy and coherence account for about 85% to 90% of the gains of our SEF ensemble while the rest of the factors (such as diversity, alignment, dropout etc.) contribute the remaining 10% to 15%. If our theory is correct, this result is not surprising but rather expected. The title of volume I is "synthesis and coherence" because those are the two most important learning principles in this volume.

For illustration purposes, we show the errors of the SE and SEF networks and ensembles in figure 8.7.

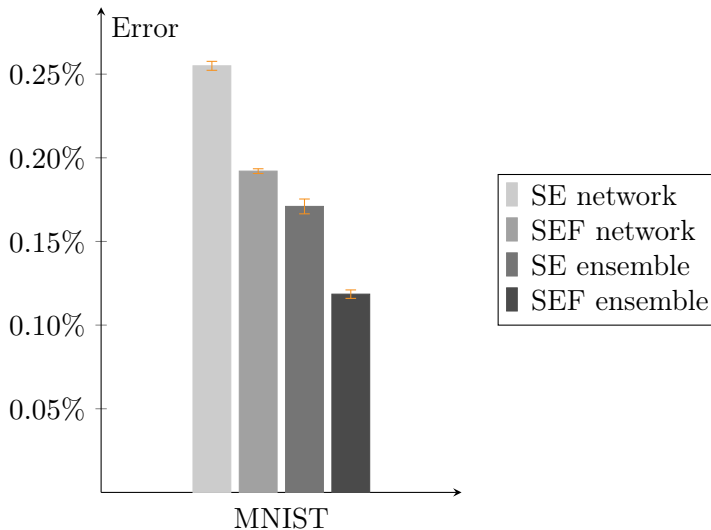


Figure 8.7: Average test error rates and confidence intervals of the SE and SEF networks and ensembles on MNIST.

We believe that more sophisticated synergy (an ensemble is merely voting or averaging) as well as coherent learning can improve the state-of-the-art further. There are many other factors and components of our system that can be improved. The SEF ensemble proved to be spectacularly successful because its design has been guided by Mathesis. However, most of the specific values for the various configuration parameters of the system and the data are nothing more than educated guesses. There is clearly room for improvement.

On the other hand, the error rate for MNIST is now so minimal that most improvements or changes in the system are likely to have a small or negligible effect. In a sense, our SEF ensemble could represent the end of the road for the MNIST data set – a spectacular end but an end nevertheless.

8.8 Publication Bias

In neural network and machine learning literature, there is an extensive publication bias that is independent of replication. Let us understand this issue in more detail.

Arguably the most common setting in neural network and machine learning research is to partition a data set into a training set and a test set. Researchers or research teams then train on the training set, test on the test set, and publish their results if they are positive. This setting is relatively simple and allows for easy comparison between systems developed by different researchers or teams because they all test on the same test set and their results are thus directly comparable.

On the other hand, this is simply a recipe for publication bias. Even if we assume that the test set is used only once by each researcher or team, those researchers and teams collectively fit the test set, especially when their number is great. For each popular data set, there have probably been many thousands of students, engineers, researchers or teams that have used the data set, tried their approach, and published or reported positive results; and most of them probably used the test set more than once.

The result of all this is that published test error rates tend to underestimate the true generalization error. This is particularly acute for smaller or popular data sets.

The MNIST data set is one of the smallest and most popular data sets of all time. Publication bias is thus a significant concern. Our estimates across various studies, including most of our own studies and results, suggest that publication bias has artificially reduced the test error by about 18% or more (relative) in all the cases that we tried. Given that differences of test error estimates across many published studies are smaller than 18%, we can understand how unreliable relevant results and claims can be. This has been aggravated further by MNIST studies that have performed their experiment only once. The only conclusion that we can draw about such studies is that they are reliably unreliable.

In general, when publication bias reaches a critical mass, it is not merely an issue of underestimating the true generalization error. It can lead to various erroneous claims and conclusions, including incorrect selection of models, methods, and parameters.

In volume II, we will present methods that estimate and factor out the publication bias and provide more accurate estimates of the true generalization error. Such methods are important or crucial for the development of the field. However, we will not discuss them further in this volume. We hope that the neural network community will independently study and address this problem and come up with similar or better solutions before we publish volume II.

8.9 Flowers for ImageNet

We initially planned to include an evaluation of flower nets on the ImageNet data set in volume I. However, for various practical and logistical reasons, we decided to move the ImageNet experiments to volume II.

The simplest experiment that one can do is to take any CNN for ImageNet and replace its convolutions with flowers. The flowers should be bigger so that the total number of parameters in the resulting flower net is the same or comparable with the total number of parameters in the CNN. And the flower net should be trained and tested with the same procedure that was used for the CNN so that the comparison is fair. For example, if parameter sweep was used for the CNN, the same should be used for the flower net.

We will leave the above as an exercise for our readers. We should note that two popular networks for the ImageNet data set are ResNet and ResNext. ResNet is a CNN while ResNext is a CNN with grouped convolutions. ResNext clearly outperforms ResNet. We can offer a hint to explain this difference in performance if we consider the fact that grouped convolutions are sunflower corollas (or, equivalently, sunflowers with empty cores). In other words, ResNext is a sunflower net.

As we explained in section 8.2, sunflowers tend to be so sparse that they force oligopetalous designs with large petals. They also correspond to stepwise rather than gradual structural differences. As a result, they are at a clear disadvantage when compared to flowers and general flower nets. However, the full connectivity among convolution filters used in CNNs is the worst possible configuration. In our ranking of synaptic structures for the filters, CNNs are at the very bottom. Sunflowers may not be much better but they could not possibly fail to outperform the bottom.

8.10 The N Programming Language

The issues of neural network frameworks that were mentioned in section 8.6 are neither a coincidence nor an isolated incident. Pytorch and TensorFlow have many such issues. They are largely the outcome of a sprawling growth

and proliferation of libraries, classes, interfaces and overall functionality without a clear sense of direction. They are also a consequence of compromised or incremental design, considerable redundancy, multiple overlapping interfaces, poor integration with distributed processing as well as a haphazard combination of programming languages and paradigms.

The latter has resulted in situations that are so unfortunate they actually have educational value as examples of counterproductive system design. Our latest example is a *memory leak* that we found in the SEF ensemble code. The entire code has been written in Python, which is a language with *automatic memory management*,* and yet there was a memory leak! Searching the internet for a solution, we spent hours reading various posts from frustrated users before we finally found a recommendation to clear the Keras backend session and call the garbage collector manually. The recommendation resolved the memory leak but that is not the real issue.

How is it possible to have a memory leak in Python code, if Python has automatic memory management? The answer is that TensorFlow and Keras combine code that is written in C and Python. Python is not an efficient programming language and C is used for many performance-critical components. However, this combination can create an opportunity for a memory leak in C code that manifests in Python code.

Computer science contains multiple programming paradigms. There are good reasons that those paradigms are distinct. We may do research and find ways to combine them; we need appropriate compiler methods and technology; but we cannot possibly pile those paradigms on top of each other and hope for the best. The latter is not merely poor design. It is no design at all.

Yet, this is only a high-level view. The issues of Pytorch and TensorFlow are just too many to list here and they are beyond the scope of this section. When we compare the two frameworks, Pytorch is the clear winner but that is only because TensorFlow is also overwrought, disoriented, and fraught with many other problems and design issues. In our view, Pytorch requires reorganization, reduction, and a rewrite of large parts of its functionality while TensorFlow requires redesign and almost a complete rewrite. As they are now, neither framework is appropriate for true neural network development.

In the history of computer science, programming languages have been introduced and implemented upon considerably less demand for them. They have also played a crucial role in the development and progress of related

* *Automatic memory management* is a type of management of computer memory as a resource of the system, which is usually the operating system or an application. It supports dynamic allocation of memory to programs as needed and release of memory back to the system when the allocated memory is no longer used. Memory release is usually handled by *garbage collection* – which releases memory when it is no longer referenced. *Manual memory management*, which requires that the programmer manages the memory, can lead to a number of problems such as forgetting to release allocated memory, causing a *memory leak*, or trying to access memory that is already released. Automatic memory management eliminates such problems but with a small performance cost to maintain a memory reference record and run the garbage collection process.

areas and fields.

For example, the *C programming language* was developed by D. Ritchie between 1972 and 1973 in order to construct utilities for the *Unix* operating system. It was also used to re-implement the kernel of Unix. Yet, the applications of C did not end there. C not only became the standard in systems programming during the 1980s but it also propelled system design and implementation to new levels. This is not a coincidence. Good languages make engineers more productive and enable better solutions and progress. They are not merely a desirable tool. They are an essential tool.

C++ represents a similar case. B. Stroustrup developed C++ in the 1980s as a language that could support the development of large software applications. The result was an efficient object-oriented language that not only became a widespread standard in the computer industry in the 1990s but also enhanced and facilitated software engineering at large scale.

Similarly, the advent of the internet came with or brought forward important new languages such as HTML, Java, and Javascript. HTML is a markup language, not a programming language, but this only underlines the fact that the necessity for new languages or a new formalism is not native to software engineering or computer science. Major scientific developments have generated, derived, or subsequently emerged from such new frameworks. From Euclidean geometry to quantum physics, science is replete with such examples.

Neural networks cannot and should not be an exception. It is irrational and counterproductive to use existing languages for neural networks – especially inefficient or thread-challenged languages like Python. The reason for this oversight is a disconnect and education deficit within computer science that we explained in more detail in *Neural Age*. Researchers who have a good understanding of neural networks do not usually have as much experience or a background in systems programming, compilers, or programming languages and vice versa. The result is Pytorch and TensorFlow.

A new programming language dedicated to neural networks is long overdue. We call this the *N programming language* in direct analogy to the C programming language. The language must meet a set of minimum criteria – and nothing more.

N has to be an extremely efficient language like C so that we do not have to mix up code from multiple languages. Yet, it has to be flexible, supporting manual memory management for crucial applications and automatic memory management for cases where speed and economy are not as important or manual management fails.

N must act as a single layer between hardware and neural networks. It should provide the substrate for all other code. New hardware should only have to support N. Data should only have to conform to N. Then everything should work with no further interfacing or implementation. If some new hardware is invented and existing applications are ported to it, the first step

would be to implement N. For most new hardware solutions, this should be the only step, too.

As a result, N should be strictly minimal, free, and open. Additional functionality can be encoded in software libraries that should be implemented in N.

N should also be extensible. It should have a flexible syntax that includes the definition of new operators and provides an operator for element-wise functions. It should also include split notation and not only for arguments or other parameters but also for types. Types should be optional and inferred.

N should support procedural and structural programming as well as object-oriented programming and various elements of functional programming. In general, it should be a decision of software engineers whether they would write code that looks like C code, Python code, or anything in between. N should handle all those cases and try to minimize the computational overhead of Python-like blocks.

N should also provide for automatic differentiation as well as a few other computational elements that will be introduced by Mathesis.

Neural network development should be based on design layers that will be software libraries implemented in N (like Python modules that are implemented in Python.)

The *neural library* should contain numerical functions and operations that will be used to compose more complex elements. It should use such elements to construct units, neural layers or blocks, neural partitions, and eventually arbitrary neural networks. It will be the first part of a neural algebra.

The *synthesis library* should extend the neural algebra and the neural library to include neural synthesis and synergy. It should provide support for synergetic networks. It has to be co-developed with the *learning library*, which would be dedicated to learning and learning methods and paradigms.

The *data library* should be about data management and processing. This can be anything between a minimal relational algebra and a distributed SQL-like or Hadoop-like library. It should also contain lexical or other textual tools as well as functionality for signal, image, or video processing such as affine image transformations, Fourier transforms, mel-frequency cepstral coefficients etc.*

The *distributed library* should be about distributed processing and custom synchronization of parallel modules. In general, N should allow for both parallel and sequential execution without requiring intervention from the user except when a custom form of synchronization is needed. Yet, all synchronization should be optional. It should be easy to turn it off and let the network operate in an unconstrained parallel mode.

As we develop Mathesis further, we will specify additional libraries, such

*The data library can be extensive. It should be partitioned in constituent libraries. However, all of those libraries are not part of the N language but rather additions to it. Consequently, they can be developed independently by third parties and added to N.

as libraries for plasticity, heredity, evolution etc. In many ways, N will grow and expand with Mathesis and follow its development.

We will provide a more detailed and formal description of N in volume II. We will most likely publish the specification and start developing the language long before volume II is published. Our intention is to develop N and Mathesis in parallel the same way that C and Unix were developed together. However, we would like to think that we would not be alone in this effort. We look forward to collaborating with the neural network community and computer industry in order to create a better language and a better framework for neural network research and development.

8.11 Historical and Bibliographical Remarks

The term Δ -system first appeared in 1960 [105] but it has largely been replaced by the more graphical term sunflower, which appeared about 20 years later [80]. We have likewise adopted the latter term.

MNIST is an acronym that stands for Modified National Institute of Standards and Technology in reference to the NIST laboratory and agency. Information about the MNIST data set and references to empirical studies and models, including state-of-the-art models, can be found on the MNIST Wikipedia page [386]. Unfortunately, most of those studies do not include measures of statistical significance. It is thus unclear whether their claims are accurate or how inaccurate they are.

The MCDNN paper on MNIST was published in 2012 [55] while the dropconnect paper was published about a year later [357]. The MNIST study that uses randomized deep learning was published in 2018 [195]. Finally, the code for the MNIST SE ensemble was uploaded on GitHub in 2020 [126].

The ImageNet data set was announced in 2009 [79]. ResNet and ResNext were published in 2015 and 2017, respectively [152, 397].

Information about Pytorch and TensorFlow as well as tutorials and documentation of their APIs can be found in their websites [264, 170].

The classic textbook about the C programming language was published by B. Kernighan and D. Ritchie in 1972 [183]. An account of the development of C was published by D. Ritchie in 1993 [283].

B. Stroustrup published the first edition of his C++ textbook in 1985 [330].

It is only prudent never to place complete confidence in that by which we have even once been deceived.

— René Descartes, *Meditations on First Philosophy*, 1641.

9.1	The Perils of Empiricism	313
9.2	Metascience	315
9.3	Replication Crisis	318
9.4	Ulterior Motives	321
9.5	Medical Dark Age	324
9.6	Preparing for the Future	330
9.7	Historical and Bibliographical Remarks	338

9

Science in Crisis

In this chapter, we take a step back from our theory and try to explain the perils of empiricism, especially in the context of human traits, flaws, and fallibility. We attempt to understand the problem and its unfortunate consequences that have already materialized in medicine and social science. Drawing parallels with the conditions in those disciplines, we may infer that neural networks can suffer a similar degradation. This is a developing issue that should be addressed before it causes further damage to the field.

As we advance and expand Mathesis further, we will propose specific platforms, protocols, and methods that can serve as safeguards against the weaknesses of the empirical doctrine within the context of common human flaws and fallibility. There is no system that can prevent unconstrained ambition and greed, fraud and misconduct, or corruption. Rather, our intention is a set of principles and a process that can quickly and effectively detect such behavior and alert the community to it.

Most of this chapter is based on our book *Medical Dark Age*. Sections 9.2, 9.3, and 9.4 are excerpted from that book with only minor changes. Section 9.5 is a summary of the book while section 9.6 is largely based on the last chapter of *Medical Dark Age*. We have chosen to include all this material instead of simply referencing it so that this chapter is self-contained. Readers who would like to consider the evidence and various details are encouraged to read *Medical Dark Age* and some of its key references.

9.1 The Perils of Empiricism

Empiricism and *rationalism* are two doctrines of *epistemology*, which is the branch of philosophy that studies knowledge.

Empiricism states that knowledge derives primarily or exclusively from sensory experience. It relies on observations or evidence derived from experiments or phenomena in the natural world or a specific subject for the formation of ideas, arguments, and theories. Mathematically, it represents an approach to knowledge that is based on information and induction instead of

deduction or intuition. Empiricism was first formulated into a doctrine by Aristotle, and it has played an essential role in the formation of the *scientific method* and the development of natural science.

Rationalism, on the other hand, represents a different view and doctrine where reason is the main source and only true authority for knowledge. The emphasis is on the intellect rather than perception and sensory experience. The underlying assumption is that the world and reality (as well as any possible reality) have a structure that is determined by rational principles – primarily logic and mathematics – that can be grasped by the human mind. The origins of rationalism can be traced to Pythagoras and his statement “all is number” that expressed the idea that mathematics represent the ultimate nature of reality. The term ‘rationalism’ was introduced much later, and it derives from the word *ratio*, which is Latin for “reason”.

Rationalism is often associated with the introduction of mathematics in philosophy and natural science as well as with the intellectual, scientific, political, and social movement that is known as the *Age of Reason*. Mathematically, rationalism is based primarily on deduction and possibly intuition to drive deduction.

There is no question that deduction should take precedence over induction. Deductive conclusions are certain when the premises are established but inductive ones are only probable using evidence as a premise. Empiricism and induction are inherently probabilistic and uncertain – and fallible, too. Consequently, rationalism is a higher authority of truth.

In general, empiricism and induction are as reliable (or unreliable) as our perception and interpretation of reality can be. They depend as much on the observer as they depend on the observed phenomenon.

Aristotle generally considered the senses to be reliable but he also emphasized that “not everything that appears is true.” He specifically indicated that our sensations (αἰσθήσεις) are relatively reliable but phenomena or appearances (φαινόμενα) are not as reliable and human judgement that is based on such perceptions can often lead to errors. Kurt Gödel expressed a similar sentiment. “I don’t believe in empirical science,” he famously said once. “I only believe in a priori truth.”

Ironically, Gödel’s work on incompleteness is a rationalist argument for the necessity of empiricism. Deduction is hard. Its computational complexity increases quickly with the complexity of the underlying system. Even systems as simple as the natural numbers are incomplete. If we consider neural networks or any universal learning system, deduction is usually impossible. One cannot expect to develop theorems that can precisely predict how such a learning system would perform and which forms or variations of the system would be best for any given environment. Induction is the only true option. Learning is an inductive process by definition; and so is life, nature, and evolution. As we explained in section 2.6, “nature and humanity have opted for complexity, not provability. The latter is a desirable property but it is

often a luxury with a prohibitively high price.”

If deduction and provability are indeed luxuries in learning, we have no other choice but to rely on induction and empiricism, which usually translate to experiments, data, statistical analyses, and interpretations. However, exclusive use of empiricism comes with great risks, significant errors, and potentially broad distortions. Statistical studies and measurements can be sufficiently biased to distort reality and blend the lines between facts and fiction, thus promoting fallacies or fabrications. There can be intention or error behind such studies and experiments; and if the errors or biases are not pointed out by reviewers or caught by readers, the erroneous studies can be cited by other researchers. Fallacies can then become widely disseminated and accepted within a research community while scientific facts can appear as if they are debatable. Various undesirable effects or consequences can likewise be hidden by statistical manipulation.

The above is not an extreme or unrealistic concern. It is a reality that has affected science in the last few decades. The term *replication crisis* was coined in the early 2010s to refer to the fact that many scientific studies are difficult or impossible to replicate or reproduce.

We will discuss this unfortunate development in the next section. We will repeatedly return to this topic throughout Mathesis and we will progressively provide guidelines and protocols that can mitigate empirical errors and reduce bias in the field of neural networks and learning in general.

9.2 Metascience

Metascience is an emerging scientific field that seeks to improve the quality and reliability of scientific research and empirical results.

A first indication of the field’s necessity appeared in 1966 with the publication of a research paper that examined the statistical methods of 295 papers published in ten esteemed and peer-reviewed medical journals and found that about 73% of the papers derived conclusions from invalid justifications.

In 2005, John Ioannidis published a foundational paper for metascience where he argued that a majority of medical research publications produce false conclusions. His paper, “Why most published research findings are false”, is currently the most downloaded paper in the *Public Library of Science*.

Let us examine the issue mathematically using Ioannidis’ approach.

We assume an experiment that tests a hypothesis H for validity and generates or contains supporting evidence E . The *positive predictive value* (PPV) or *precision* of the experiment is defined as

$$\text{PPV} = \frac{\text{Number of true positives}}{\text{Number of positives}}.$$

In our case, it is $\text{PPV} = P(H/E)$, assuming that P stands for a probability

measure. Using Bayes' theorem, we can rewrite the above as

$$\text{PPV} = P(H/E) = \frac{P(E/H)P(H)}{P(E/H)P(H) + P(E/\bar{H})P(\bar{H})} = \frac{1 - \beta}{1 - \beta + \alpha r} \quad (9.1)$$

where $\bar{H}$ indicates the logical negation of H , $\beta = 1 - P(E/H)$ is probability of error when the hypothesis H is true (occurring by incidental evidence that does not support H), $\alpha = P(E/\bar{H})$ is the probability of error when the hypothesis H is false (occurring by incidental evidence that supports H), and $r = P(\bar{H})/P(H)$ are the odds of the hypothesis H being false.

In most scientific research, it is a common convention that $\alpha = 0.05$ and $\beta = 0.2$. Hence, $1 - \beta = 16\alpha$ and equation (9.1) becomes $\text{PPV} = 16/(16 + r)$. If we assume that, on average, $P(H) = 0.1$ for a given scientific field, then the expected odds of false hypotheses are $r = 9$ and $\text{PPV} = P(H/E) = 64\%$. In other words, there is a 36% chance that we would make an error if we assert H given the evidence E .

Moreover, equation (9.1) does not account for bias in the experiment. We should note that equation (9.1) is a decreasing function with respect to both α and β . (For the latter, we can simply observe that $1/\text{PPV} = 1 + \alpha r/(1 - \beta)$.) As a result, when either error α or error β (or both) decrease, the PPV value increases. Experimental bias tends to underestimate those errors and thus artificially increase the PPV value.* It can thus aggravate the problem and increase the probability that the experiment will produce an erroneous conclusion about the validity of the hypothesis H .

Ioannidis furthermore identifies a number of factors that contribute in underestimating errors α and β or otherwise inflate the PPV value. All the following factors make research findings less likely to be true.

- a. Small studies, which can more easily lead to biased versions of the error probabilities.
- b. Small *effect size* (this is the a priori probability of a condition under study), which results in small numbers of positive samples and thus less reliable estimates of the corresponding probabilities.
- c. Large number and limited selection of tested hypotheses. When numerous hypotheses are being tested, it is more likely that some correlation may emerge out of statistical chance (especially, when the sample size or effect size are small). This phenomenon is aggravated by the fact that positive results are reported but negative ones are often dismissed.
- d. Flexibility in designs, definitions, outcomes, and analytical modes in studies. For the same reasons as before, increased experimental 'creativity' can lead to erroneous correlations.

*For simplicity, Ioannidis assumes a uniform bias u , which results in error probabilities $(1 - u)\alpha$ and $(1 - u)\beta$, respectively. However, this is not necessary. Any bias that reduces the errors will increase PPV.

- e. Financial and other interests, conflicts of interest, and prejudice in a scientific field. Financial interests create pressure for positive results and novelty derived in a timely manner. They create conflicts of interest that introduce bias. Prejudice can be the result of such conflicts but it can also emerge from beliefs (in certain theories or approaches) and career objectives and constraints (such as tenure-track positions or institutional appointments). In either case, prejudice can produce similar forms of bias and lead to erroneous conclusions.
- f. Increased popularity of a scientific field (with more scientific teams involved). The more teams that test a set of hypotheses, the more likely that some will find positive correlations. The more popular a field, the greater the pressure to arrive at positive results earlier. Negative results are often dismissed and not disseminated except when they refute earlier published claims. (This has led to the *Proteus phenomenon*, which represents a tendency to refute early or original claims through failed or contradictory replications of a work and often replace them with similarly erroneous and refutable claims.)

In recent years, and particularly after Ioannidis' seminal publication, the field of metascience grew more rapidly and developed further in terms of complexity, scope, and initiatives. We briefly describe some of its main areas below as well as the primary sources of bias in scientific studies.

- A. The replication crisis (a.k.a. *reproducibility crisis*) is a current condition in science where many or most of the published studies are difficult or impossible to reproduce. It is a complex and multifaceted topic and it will be discussed in more detail in the next section.
- B. Conflicts of interest, corruption, and fraud are not uncommon in science. This is an extensive topic that will be discussed in latter sections, especially within medicine and the pharmaceutical industry where it is most common.
- C. A scientific foundation and a well-defined platform for peer review and evaluation of research is imperative. It can prevent bias, eliminate ad hoc procedures, reduce subjectivity, identify errors or omissions, and expose conflicts of interest and attempts to diffuse certain scientific facts and effects (to make them appear as if they are debatable because they are financially or otherwise undesirable). Overall, such a platform can protect science from the many perils of empiricism and contain the effects of human fallibility and corruption.
- D. Poor design of studies and experiments and improper use of statistics (especially p-values and statistical significance) can introduce significant bias in the data and lead to incorrect interpretations of the data and unjustifiable conclusions.

- E. Poor practices, reduced transparency, and omissions in documentation, explanation, and dissemination of research can make it difficult to interpret the results, replicate them, and identify biases and conflicts of interest in the authors.
- F. *Publication bias* is a bias in published research that occurs when the outcome of a research study affects the decision to publish or disseminate the results. It usually produces a bias in favor of significant positive results. However, publishing only positive results creates a bias that distorts the underlying distribution regardless of how statistically significant the results are. This is effectively an aversion to null or negative results. Research indicates that studies with statistically significant positive results are about three times more likely to be published than papers with null results and despite comparable design quality in the experiments.
- G. A current *publish-or-perish* environment in academia promotes low quality research, false positives, and questionable outcomes. It emphasizes the number of publications and ‘interesting’ plausible results rather than high-quality research and true discoveries. Metascience attempts to promote higher-quality research through improved incentive systems.
- H. *Observer bias* (a.k.a. *experimenter bias*) occurs when the beliefs and expectations of a researcher about the results of their research study bias the research outcome or its interpretation. *Blind experiments* are experiments where information that may affect or influence the participants is withheld until after the experiments are complete. Such experiments are often used to limit or eliminate observer bias. However, blinding is not always properly reported or implemented, especially in medical literature, while failure of blinding is rarely measured or reported. Ideally, all research should assess and report the quality of blinding.
- I. Career considerations and a desire for media attention can introduce a bias for misleading claims, exaggerations, and reduced epistemological modesty.

9.3 Replication Crisis

Replication is an essential and indispensable component of empiricism, empirical research, and the *scientific method*. It serves multiple purposes. It can limit and expose sampling errors, publication biases, questionable research practices, malpractice, and fraud; it can test whether results generalize to different or larger samples; and it can support or reject an underlying theory or hypothesis and determine its true scope. When it is successful, replication can increase credibility within a scientific field.

On the other hand, it is a crisis in a field when replication efforts broadly fail in that field; and it is a crisis in science when such failures occur across multiple fields. Non-replicable studies can diffuse facts and taint the scientific record with various fallacies and fabrications. Such studies can become a source of misinformation, false marketing, propaganda, and fraud when there are financial or other interests behind the related claims. In the health sector, the replication crisis has actually become a tool that is used to achieve unprecedented levels of fraud (as we shall see in section 9.5).

In general, the replication crisis seems to have started in the second half of the 20th century. It primarily affects medicine and social science but also natural science to some extent. It is a complex and extensive topic. We will only discuss some of its aspects in this section in order to understand how it can affect learning and neural networks and how to safeguard the field against such misfortune or deceit.

We can get an idea of the extent of the replication crisis with some concrete examples. Those examples represent only a tiny fraction of all cases in approximate chronological order, starting with the earliest known reference. Overall, the literature on the replication crisis is extensive. The cases are so widespread and their total number so large that is hard to produce a precise estimate.

In a study published in 1959, psychology articles that were published in 1955-56 were examined and 97% of them were found to reject the null hypothesis. Given that insignificant results are not published, significant results are bound to occur by mere chance after a number of experiments. It is possible then, as the author stated, that “the literature of such a field consists in substantial part of false conclusions resulting from errors of the first kind in statistical tests of significance.” The results of this study also suggest that the replication crisis started in the 1950s or earlier.

In 1966, another study examined 295 papers published in ten highly-esteemed and peer-reviewed medical journals and found that “conclusions were drawn when the justification for these conclusions was invalid” in about 73% of the reports. This implies that almost 3 out of 4 conclusions were effectively noise; and if we assume that the estimate is accurate across most medical research, it implies that most of the conclusions in published medical research are noise. The results also indicate that the replication crisis had already reached a critical mass by the late 1960s. However, and despite its severity, the replication crisis received very little attention until Ioannidis’ seminal paper in 2005.

In his paper, Ioannidis went further and actually claimed that the majority of published medical research is false. His claim paints a bleak picture of medicine; and if we assume it to be true, it brings medical research dangerously close to superstition.

In 2009, a meta-analysis combined 39 research surveys and found that 1.97% of scientists admitted that they have had fabricated, falsified or modified

data or results at least once, and up to 33.7% admitted other questionable research practices. The rates for such behavior among colleagues were 14.12% and up to 72%, respectively. As the study indicated, given the sensitive nature of such questions and limitations of surveys, the above estimates are probably conservative estimates of the true prevalence of scientific misconduct.

In 2012, an analysis of 53 pre-clinical cancer studies found that only 11% of them could be replicated. The non-replicable studies shared a number of common features and flaws in their execution or design. The overall result represents one of the highest rates of replication failures in literature. It arguably demonstrates the extent of the replication crisis in medicine.

In 2013, B. Nosek and J. Spies founded the *Center for Open Science*, which is a non-profit organization with the mission to “increase the openness, integrity, and reproducibility of scientific research.” Likewise, recognizing the replication crisis and its extent and implications, the *Defense Advanced Research Projects Agency* (DARPA) initiated the *Systematizing Confidence in Open Research and Evidence* (SCORE) program with the goal to develop automated tools that assign “confidence scores” to different social and behavioral science research results and claims. The overall goal of such institutions and programs is not to eliminate human bias but rather to expose and contain it.

In 2015, the *Open Science Collaboration*, which is a program by the Center for Open Science, evaluated 100 experimental and correlational studies that were published in three psychology journals. 97% of the original studies had significant results ($p\text{-value} < 0.05$) but only 36% of the replications had significant results. It found that 47% of original effect sizes were in the 95% confidence interval of the replication effect size; 39% of effects were subjectively rated to have replicated the original result; and if no bias in original results is assumed, combining original and replication results left 68% with statistically significant effects.

In 2016, the journal *Nature* conducted a poll of 1576 scientists and reported that more than 70% of researchers had failed to reproduce one or more of another scientist’s experiments, more than 50% had failed to reproduce one or more of their own experiments, 24% had been able to publish a successful replication, and 13% had published a failed replication.

In an article published in 2016, Ioannidis explained that clinical research should be useful in the sense that “it should make a difference for health and disease outcomes or should be undertaken with that as a realistic prospect”; but he estimated that 85% of the funding for all medical research is not useful.

Economists tried to measure the extent of the waste in a study that was published in 2018. They combined prior studies and estimated that non-reproducible preclinical research represents a majority that exceeds 50% and it consumes approximately \$28 billion per year in the United States alone. They furthermore indicated that such studies “undermine cumulative knowledge production and contribute to both delays and costs of therapeutic

drug development.”

In our view, this is the most detrimental effect of the replication crisis. It undermines science and our knowledge, which we have painstakingly acquired and accumulated over thousands of years, and it diffuses the scientific record with fallacies and falsities. We should also note that the above estimates of waste are consistent with Ioannidis estimates that the majority of published medical research is simply false.

In two studies published in 2018 and 2020, respectively, it was found that replication relates to plausibility. Using surveys and a prediction market, the first study found that “peer beliefs of replicability are strongly related to replicability, suggesting that the research community could predict which results would replicate and that failures to replicate were not the result of chance alone.” The second study found similar results among non-experts, thus establishing a relation between intuitive plausibility and replication. The study considered 27 high-profile social-science findings, used a sample of 233 people without a Ph.D. in psychology, and found that the participants predicted replication success with 59% accuracy – or 67% accuracy when they were informed about the strength of evidence from the original studies. In other words, even non-experts can predict which results would probably not hold up to scrutiny, and yet the reviewers, who should be experts or at least more informed about the relevant areas, allow them to be published.

In 2021, a study showed that published papers in top psychology, economics, and general interest journals that fail to replicate are cited more than those that replicate; that the difference in citation does not change after the failure to replicate is published; and that only 12% of citations acknowledge the replication failure. Given the evidence that experts can predict which papers can be replicated, the authors postulated that reviewers applied lower standards to papers that seemed more “interesting” even when they were unlikely to replicate.

There are many more replication studies. Failure rates vary across scientific fields and journals but the studies generally support the premise that the replication failures represent a widespread crisis that is more common and severe in medicine and social science.

9.4 Ulterior Motives

In this section, we will focus more on the causes of the replication crisis rather than its various effects and forms – which can misdirect from the true nature of the crisis. We will attempt to create a binary classification for the causes but we will argue that this is primarily a singular problem where a single class defines the vast majority of all cases.

Let us start with a blog, *Fantastic Anachronism*. In a related entry, the author, A. De Menard, states that he participated in DARPA’s SCORE program about replication markets and explains that prediction markets are

accurate because predicting replication is easy. Papers that fail have “obvious, surface-level problems.” He also explains that it is impossible to accept the popular belief that weak studies are the result of unconscious biases, subtle statistical errors, or questionable theories.

We can only echo the sentiment – for we indeed find such a belief “impossible to accept.” De Menard demonstrates its incredibility in unequivocal terms: “even if the authors really are misled by the forking paths, what are the editors and reviewers doing? Are we supposed to believe they are all gullible rubes?”

De Menard furthermore explains that the problem is aggravated by the fact that students or researchers do not always read the publications they cite or they merely skim through them. (In a prior publication, it has been estimated that only about 20% of cited papers are read before citation.) We believe that this factor adds to the proliferation of citations of weak or non-replicable studies. It may partly explain why citations continue even after replication fails but it does not explain how and why the weak studies came to be on they first place, or why the reviewers and the editors failed to detect the flaws or weaknesses and block the papers from publication. The very existence of such publications, their alarming frequency, and the associated editorial negligence suggest a different explanation.

It is almost trivial to argue that scientific research and publications are not as much related to skimmed papers or incidental negligence as they are to a complex of financial interests, conflicts of interest, and career objectives. Most researchers cannot afford to oppose such interests. They have to secure grants, publish in journals, and attend conferences and faculty meetings. They greatly dependent on their sources of funding and their peers too. It is more convenient to attribute poor research to the intricacies of the subject rather than the ethics of the researchers and the interests behind them. Fraud will not always be exposed; and when it is, sometimes many years or decades later, “the fraudster can admit, without much of a hit to their reputation, that indeed they were misled by that dastardly garden”, if we may quote A. De Menard again.

The relationship between academia and various industry sectors is a delicate affair. When academia depends heavily on a sector for funding, corruption can spread quickly from the sector to the corresponding academic fields.

Professor A. Relman concisely and accurately described the situation in medicine: “The medical profession is being bought by the pharmaceutical industry, not only in terms of the practice of medicine, but also in terms of teaching and research. The academic institutions of this country are allowing themselves to be the paid agents of the pharmaceutical industry. I think it’s disgraceful.”

There have been a number of publications and books that make the same argument and demonstrate the intricate web of conflicts of interests in

medicine and the extent of the corruption – and with many specific examples. However, this is not the bulk of the problem but only part of it as we explain later in this chapter and demonstrate in greater detail in our book *Medical Dark Age*.

Over the years, a considerable number of researchers and politicians or governments and institutions have made various erroneous or extreme statements. In the same manner, business leaders and executive boards have seemingly committed strategic errors. We can potentially attribute those statements and failures to various factors such as ignorance, shortsightedness, incompetence, naivety, or even coincidence and bad luck. We call the above the *naive interpretation* and we consider it to be a highly unlikely explanation.

In most cases, there are undeclared motives behind such statements or actions. When those motives are not taken into account, the statements may appear as fallacies and the actions as miscalculations. They are effectively taken out of context and they appear as errors.

However, when the true motives are considered, the statements and actions become more deliberate and calculated. They are often strategic or tactical responses with specific purposes. Those purposes vary. The intent is often to hide the truth and make facts appear as debatable, promote careers or certain beliefs, manipulate public opinion and scientific consensus, and more commonly protect financial interests by influencing markets, maximizing revenue or profits, and minimizing damage or losses. Sometimes there are combinations of such motives but financial motives are present in most cases.

We refer to the above as the *ulterior motive hypothesis* and we consider it to be the true explanation for the vast majority of cases.

Business leaders are neither incompetent nor naive. Politicians are not driven by ignorance or inexperience but rather by ambition. Institutions are likewise neither gullible nor shortsighted. Researchers are not ignorant fools who make outrageous statements. When such events occur, there is always an ulterior motive. Arguably, there may be some exceptions, such as politicians who are not astute enough to comprehend all the factors sometimes, but their actions are usually the result of their ambitions, desire for power, or plain greed. Their potential ignorance or incompetence is irrelevant.

Hanlon's razor states that we should “never attribute to malice that which is adequately explained by stupidity.” It is an adage that may apply to certain human affairs but has no application whatsoever when there are financial interests and billions to be made, power to be grabbed, and careers to advance or make. In such cases, the main factors are not stupidity or ignorance but rather profit, authority, and ambition as well as greed, corruption, and fraud. If we choose to call all this malice, the *opposite* of Hanlon's razor would be true. In the end, it does not matter what term we apply for it, or whether a naive and simplistic razor reverses. The result is the same and it is detrimental – or disastrous. (We can think of climate change as an example.)

It is always instructive to look at specific cases in order to understand

an argument. We provide a number of examples in *Medical Dark Age* and apply the ulterior motive hypothesis. For comparison, we also present the naive interpretation. Readers can read the book and decide for themselves which one is more likely to be true. There is also a much larger collection of examples and supporting evidence in some of the key references of *Medical Dark Age*.

9.5 Medical Dark Age

In this section, we provide a summary of *Medical Dark Age*. Readers who are strictly interested in learning and have no interest in understanding the current situation in medicine can skip or skim through it. However, we consider the situation in medicine and social science to be a cautionary tale that is relevant to all empirical science. We encourage all readers to read this material.

Here are the main facts and claims of *Medical Dark Age*.

The health sector has undergone a gradual and profound transformation since the 1970s. It is now the most fraudulent sector in American history, accounting for more than two thirds of all fraud against the federal government as of 2020. Statistics from the US Department of Justice show that the federal government has recovered more than \$64.450 billion under the *False Claims Act* (FCA)* from 1986 to 2020. They also show that \$43.375 billion, or approximately 67.3% of the recovered amounts, are in health and human services, \$6.211 billion, or 9.6%, cover matters related to the Department of Defense, while the remaining \$14.864 billion, or 23.1% relates to other categories, agencies, and departments. A chart with the above numbers is shown in figure 9.1. The bulk of fraud is no longer in the military-industrial complex rather in the *medical-industrial complex*.

Overall, this is an unprecedented level of fraud, far beyond any other sector or industry in the US, and it has caused extensive human suffering, a total death toll of millions, and an economic cost of trillions US dollars. (The *opioid epidemic* cost the U.S. \$504 billion in the year 2015 alone.) Moreover, the fraud statistics from the Department of Justice probably represent a fraction of the total amount of fraud and the total number of cases.

Yet, the transformation did not end with fraud. The pharmaceutical sector effectively created a new form of organized crime. If this seems like an exaggeration, let us consider the evidence and related facts.

Big pharmaceutical companies (aka *big pharma*) have corrupted the government and science. They have shaped policy about medicine and health care, and they have dictated decisions and laws that gave them unjustifiable privilege and protection. The result of all this is that health care costs in

*The False Claims Act is an American federal law that imposes liability on persons and companies who defraud the government. It is the federal government's primary litigation tool against fraud that affects the government and its programs or institutions.

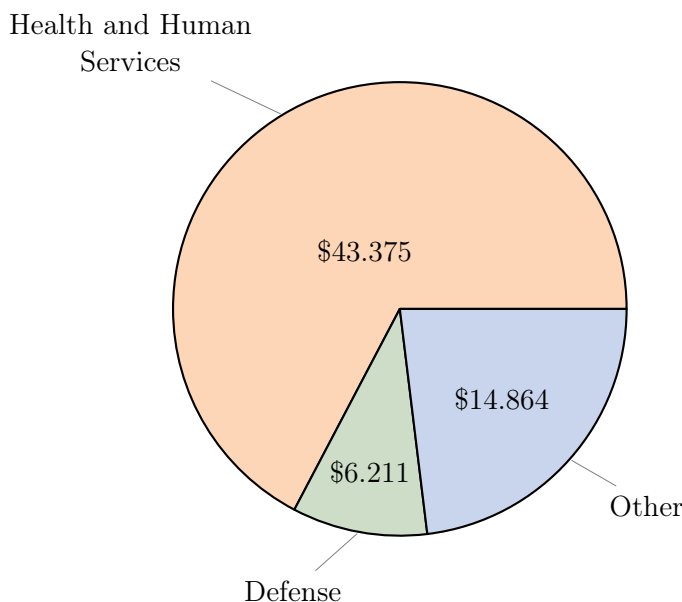


Figure 9.1: FCA recovered amounts (in billions US dollars), October 1, 1986 – September 30, 2020.

the US have increased from 5% of the GDP in the 1960s to nearly 20% in 2020. They represent about one fifth of the GDP and they are a major burden on the nation's budget. The cost of health care in the United States is more than double the average of most developed nations as a percentage of GDP (adjusted for differences in cost of living); and the gap follows a clear increasing trend over time since 1980 as shown in figure 9.2.

A. S. Relman and M. Angel have published two critical articles and a book about the pharmaceutical industry. They point out that American expenditures on prescription drugs were virtually steady as a percent of U.S. GDP from 1960 to 1980. However, from 1980 to 2000, they tripled. More recently, spending increased 4.3% in 2019 and 3.0% in 2020, amounting to \$348.4 billion. Over the next decade, the Centers for Medicare and Medicaid Services project that spending for retail prescription drugs will be the fastest-growing health care category and will consistently outpace that of other health care spending. Yet, no action has been taken by Congress against big pharma. On the contrary, big pharma has been offered legal immunity (for vaccine injuries) and a government granted monopoly – through patents and FDA-approved exclusive marketing rights – so that they can inflate drug prices almost at will.

R. F. Kennedy, Jr. has described big pharma as a racketeering syndicate. E. McCarthy, assistant professor of business law, has authored a call to prosecute drug company fraud as organized crime under the *Racketeering Influenced and Corrupt Organizations Act* (RICO) of 1970. He explains that

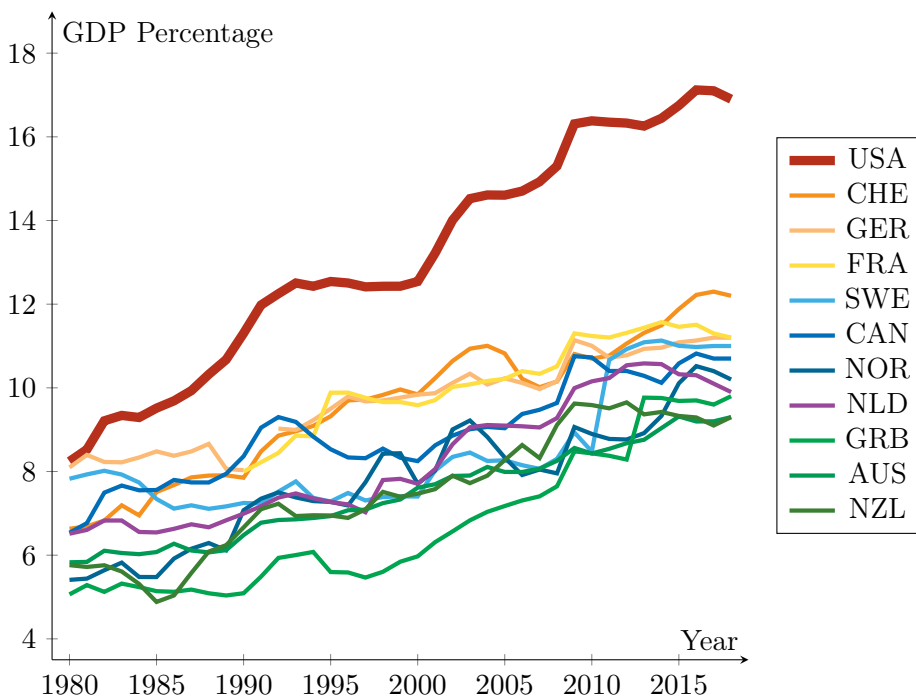


Figure 9.2: Health care costs from 1980 to 2018 as a percentage of GDP, adjusted for differences in cost of living across different nations. Source: Organization for Economic Cooperation and Development (OECD) Health Statistics 2019.

prescription drug fraud “contributes to more than 100,000 American fatalities every year” while “few (if any) drug company executives or other complicit parties face criminal charges for fraudulently developing and marketing these drugs.”

A most detailed and informative source about big pharma as organized crime is P. C. Gøtzsche’s book *Deadly Medicines and Organised Crime: How Big Pharma has Corrupted Healthcare*. It is a remarkable and authoritative book. It contains an extraordinary number of examples, details, and evidence that the pharmaceutical sector is organized crime. The book also includes an insightful quote from P. Rost, a former Pfizer marketing vice president:

“It is scary how many similarities there are between this industry and the mob. The mob makes obscene amounts of money, as does this industry. The side effects of organized crime are killings and deaths, and the side effects are the same in this industry. The mob bribes politicians and others, and so does the drug industry [...] The difference is, all these people in the drug industry look upon themselves – well, I’d say 99 percent, anyway – look upon themselves as law-abiding citizens, not as citizens who would ever rob a bank [...] However, when they get together as a group and

manage these corporations, something seems to happen [...] to otherwise good citizens when they are part of a corporation. It's almost like when you have war atrocities; people do things they don't think they're capable of. ”

R. Smith wrote the book's foreword and pointed out that the characteristics of organized crime include extortion, fraud, federal drug offenses, bribery, embezzlement, obstruction of justice, obstruction of law enforcement, tampering with witnesses and political corruption. He explained that the author “produces evidence, most of it detailed, to support his case that pharmaceutical companies are guilty of most of these offenses.”

We can only support the above claims and arguments. Every counterargument or effort to portray big pharma in a less liable form is relatively easy to refute with evidence and numerous examples across multiple decades.

The pharmaceutical sector has largely become a sector of organized crime where the various big pharmaceutical corporations are the equivalent of mafia families or factions within the mob, while their actions are loosely coordinated by their trade association, the *Pharmaceutical Research and Manufacturers of America* (PhRMA). The association includes virtually all American manufacturers of brand-name drugs, and many foreign manufacturers, too.

Big pharma uses lobbying and corruption to ensure that Congress will not pass related laws and the government will not interfere and disrupt their criminal activities. PhRMA employs hundreds of full-time staff and lobbyists, and conducts an extensive and nonstop campaign on behalf of their clients. In 2017, they had revenue of \$455 million, \$128 million of which was spent on direct lobbying activities. This does not include lobbying or other spending that is taken up by individual pharmaceutical companies. The total expenses are probably more than a billion US dollars every year. Big pharma spends so much and has such extensive influence and control of Congress that it has been argued that “Congress is owned by pharma” and that “it really doesn't matter who the president is.” As a result, the government has not taken any action against big pharma. Most likely, they never will.

Yet, organized crime and corruption are not the end of it. This unfortunate situation in medicine and health care extends even further. Pharmaceutical fraud and misinformation require falsification of science and scientific facts. They require control or decisive influence of the medical establishment and medical knowledge and beliefs. They require fabrications, statistical manipulation, bias, and various other forms of deceit – as well as an overall environment where there is limited criticism and fallacies are treated as established facts.

The scientific record of medicine and medical publications are now replete with such fallacies and fabrications. Big pharma has used the replication crisis as a tool. They contributed to it. They proliferated and enhanced it. In order to evade criticism and suppress resistance, they created an avalanche of conflicts of interest in medicine.

As Relman and Angell pointed out in 2002, “virtually every research-

intensive medical center in the country now has contractual ties with one or more drug firms.” Clinical trials have already shifted from academic medical centers and teaching hospitals to *contract research organizations* (CROs), which are hired by pharmaceutical companies. The design and execution of those trials are now completely under the control of the pharmaceutical sector and subject to numerous conflicts of interest. Such trials are combined with financial ties and a revolving door system between big pharma and regulatory institutions in order to manipulate those institutions and evade regulation.

In general, conflicts of interest have become one of the most common affairs in medicine. In her article, “Drug companies & doctors: a story of corruption,” Angell provides prominent examples and demonstrates that even the most prestigious universities and research institutions in the country are not immune, or even resistant, to corruption and manipulation by the pharmaceutical sector. Relman accurately and succinctly described this condition: “The medical profession is being bought by the pharmaceutical industry, not only in terms of the practice of medicine, but also in terms of teaching and research. The academic institutions of this country are allowing themselves to be the paid agents of the pharmaceutical industry. I think it’s disgraceful.”

Medicine and medical research are slowly and persistently degrading, admitting more falsities and fabrications. There is a new *religion of false medicine* that has spread in them. It consists of convenient beliefs and harmful but profitable fallacies and fabrications that are disguised as science and scientific facts. And it is under the authority of big pharma – along with the rest of medicine.

Big pharma is effectively the church of this new religion. In a manner similar to the medieval church, they operate under a status of universal authority – across everything that relates even remotely to medicine or health care. They did not declare or advertise their authority but they apply it to all affairs consistently and continuously.

Overall, big pharma and the medical-industrial complex represent a power structure that fits the description not only of organized crime but also of a new religion and medieval church within medicine and the medical establishment. We can thus argue that big pharma and their web of power represent a partial return to the Dark Ages. It is a far more complex and sophisticated version of the Dark Ages; it is modern, evolved, and more restrained; and it is primarily about medicine, medical research, and medical policies and legislation. We have called it the *Medical Dark Age*.

If our term seems like hyperbole, let us consider some facts and put all this in perspective. It is not only ballooned health expenses, an unprecedented level of fraud, or excessive human, social, and economic costs. It is not a usual type of crime. In the last few decades, there have been various health crises, including the opioid epidemic, the coronavirus pandemic, the autoimmune disorder epidemic, and the autism and diabetes epidemics. Humanity has

suffered new forms of oppression, such as lockdowns and vaccine mandates, while science has witnessed a substantial degradation of medicine and medical research with a majority of questionable or patently false medical publications. It is not hyperbole but an undeniable failure to be well into the 21st century and a single virus locks down much of the planet. The current situation with Covid-19, vaccines and vaccine mandates, lockdowns, and the associated social and economic hardship and decline is a long-term effect of greed, corruption, and shortsightedness. It is the unfortunate consequence and the legacy of big pharma, which has effectively forced their mercantile and reckless vaccine-centered approach to infectious diseases in medicine and medical research for decades now because vaccines are their most lucrative product. And they did so with the aid of complicit governments and an obedient medical establishment. The damage to democracy, society and science has been significant, and continues to increase over time. The trend is clear. The only question is whether the harm will be irreparable in the end, when big pharma is finally displaced.

In general, a skeptical viewpoint is advisable, especially upon extraordinary claims like the above. However, we would like to invite the readers to consider the evidence that we present in our book *Medical Dark Age* and make a more informed decision about our claims and their consequences. Disbelief is not an uncommon reaction to disturbing information; but we believe that the evidence leaves little room for doubt.

We have included all this material in *Mathesis* because the *Medical Dark Age* and the associated conditions in medicine and the pharmaceutical sector are relevant to neural networks and their future technological innovations and applications. There are two reasons for the latter fact.

First, neural networks are growing fast. They will become a lucrative field in this century and they will dwarf all other areas of technology and human activity as we have explained in our book *Neural Age*. The popularity of the field is growing and it will only accelerate over time. This will create extraordinary pressure for results and in a timely manner. Greed and corruption may become factors. We do not believe that the technology sector can ever become as corrupt and criminal as the pharmaceutical sector. However, even a limited amount of replication issues, misconduct, and fraud can be quite detrimental.

Second, neural networks will pave the way of the technology sector into medicine and health care. They will reduce health care costs significantly and diminish pharmaceutical revenue in most cases. They are likely to displace most of the pharmaceutical industry. This will prompt increasing resistance and direct attacks on the field by the pharmaceutical industry. As their profit margins and revenue are challenged, big pharma's attacks will become stronger, more frequent, and more vociferous. Actually, some minor attacking elements are already in place.

By drawing attention to this resistance before it occurs, we intend to

provide an early warning so that the neural network community can prepare for it and shield the field from harm. We expect big pharma's attacks to be unparalleled in scope and magnitude.

We discuss the above two issues in more detail in the next section. In the long run, it does not matter what the pharmaceutical industry will do or attempt to do. They may manage to delay or impede progress for a while but the final outcome is inevitable. Their very existence and their threat to neural networks and neural technology are strictly ephemeral.

9.6 Preparing for the Future

Critical Assessment of protein Structure Prediction (CASP) is a worldwide biennial competition for *protein structure prediction* that started in 1994. It uses metrics between actual and predicted alpha carbon positions in target protein molecules and derives an aggregate score as a measure of protein structure prediction accuracy.

In 2020, DeepMind's neural network system for protein structure prediction, known as *AlphaFold2*, achieved a record median score of 92.4, which is on par with experimental techniques. The result has been hailed as an "astounding" and "transformational" breakthrough. In 2021, DeepMind announced that they had solved the structure for 350,000 proteins found in the human body, which represent 44% of all known human proteins. The DeepMind team furthermore predicted that their database will soon grow to 100 million proteins across all species, which is about half of all proteins that are believed to exist. Subsequently, AlphaFold2 was named the 2021 Breakthrough of the Year by *Science* magazine.

The importance of protein structure prediction was indicated by biochemist C. Anfinsen in his 1972 Nobel Prize acceptance speech. Anfinsen explained that such an advance would have vast applications, offer insights into basic biology, and reveal promising new drugs.

We believe that the above result by DeepMind is a first among a long sequence of breakthroughs that will take place in the next 2-3 decades and will lead to the development of a neural antiviral expert system that will provide protein-based treatment for virtually all or most viruses known to humankind. Such an approach can also be effective against bacteria, because proteins can be used to signal the immune system. For example, the protein NOD2 can fight bacterial infections by sensing the presence of bacteria and alerting immune cells. Proteins are the true answer to infectious diseases because they are building blocks of life. Their essential role in such defense is not a coincidence but a direct consequence of the very structure of life.

The process of developing new drugs with neural networks has already started. In January 2022, DeepMind announced a first new drug discovered by AlphaFold. Their system effectively marks the beginning of a new era in pharmaceutical research and development.

AlphaFold could have been developed earlier. Data and sufficient computing power have been available for at least two decades now. If development had started in the early 2000s or earlier, we would probably have some early version of a neural antiviral expert system by now. If that system was sophisticated enough by 2020 to provide protein-based treatment for the Covid-19 virus, the entire pandemic and its unfortunate social and economic impact could have been avoided.

However, there has been a major delay in this line of research – for three main reasons.

The first is a computer science prejudice against neural networks that persisted until the early 2010s. It was the result of misinformation and propaganda that was deployed by artificial intelligence researchers in their effort to undermine neural network research and divert funding into artificial intelligence. We briefly explained this situation in section 2.7 and described it in more detail in *Neural Age*. The prejudice effectively removed neural networks from consideration and dramatically slowed down progress.

The second reason for the delay is a disconnect between artificial neural networks on one side and biology and medicine on the other side that largely persists to this day. Systems like AlphaFold2 will certainly draw more attention to neural approaches, but the gap and the disconnect may persist for a while.

The third and perhaps most important reason for the delay is the pharmaceutical industry. An expert antiviral and antibacterial system will make vaccines for infectious diseases unnecessary; it will replace them with cheaper, safer, and more effective protein-based treatments; and it will most likely mark the beginning of a decline in the autoimmune disorder epidemic, which is a great source of revenue for the pharmaceutical industry.* Consequently, such a system will represent a financially catastrophic development for big pharma as it can reduce their revenue by about 70% to 80%, per our estimates. Big pharma will never truly support, fund, or promote such research. On the contrary, they will use all means possible to divert attention from it and retain an emphasis on a vaccine-centered approach – as detrimental as this may be for everybody else.

However, there will not be any more delays. The technology sector has always been bound to expand into medicine and health care. It is a relatively slow and gradual process that requires considerably more time but it is inevitable. Neural antiviral and antibacterial expert systems will be one important component of the expansion. Diagnostic systems and robotic or computer-assisted surgical systems will be two others.

From a logical standpoint, medical inference tends to be relatively simple. Doctors are not talented mathematicians who apply complex logic and ingenious or extraordinary new methods to derive spectacular diagnoses. Most of

*More details can be found in *Medical Dark Age* and particularly its chapter about vaccines.

the time, they detect patterns and apply rules. The main complexity in this process is the large number and diversity of all possible symptoms, diseases, diagnostic tests, and treatments – as well as the medical history of the patient, which may affect certain decisions. This diversity is so great and with such enormous underlying detail that it is impossible for a human to be a medical expert across all medicine. However, this is not necessarily the case with neural systems dedicated to medical diagnoses.

As neural systems improve their natural language processing capabilities, they will eventually be able to process millions or even billions of medical textbooks, publications, studies, and even individual clinical trials. They will be able to detect and ignore or discount small studies and trial, studies that are unlikely to replicate, and studies that are likely to be biased or false. In the end, they will develop medical expertise that dwarfs the expertise of any human doctor. Most likely, they will reject certain medical practices such as prescribing expensive new brand name drugs that offer little or no advantage over older and cheaper drugs while they may introduce higher risks of adverse effect (as is the case with third and fourth generation contraceptives that are discussed in a dedicated chapter in *Medical Dark Age*). And they will certainly reject over-prescription and over-medication, which is a problem in some countries and particularly the United States.

Finally, computer-assisted or robotic surgical systems are already a reality, albeit very limited in their functions and capabilities. Those systems will only become more sophisticated over time and they will eventually be fully autonomous. Even a partially autonomous system that operates under the supervision of a doctor can have a tremendous effect on the accuracy, efficiency, and cost of surgical operations. Such systems will most likely be developed and become widely available within this century.

All those medical systems (for diagnosis, treatment, and surgery) will be guided and controlled by advanced neural networks and neural technology. In a sense, neural networks will pave the way of the technology sector into medicine and the pharmaceutical sector and they will displace the pharmaceutical industry.

Such medical systems will significantly decrease the cost of health care and diminish pharmaceutical revenue. We estimate the total reduction to be close to 90%. Furthermore, the majority of whatever revenue is left will be taken up by technology companies. Those companies will create health platforms that provide diagnosis and treatment; and people will subscribe to those platforms the same way they subscribe to media streaming or e-commerce services today. Technology companies do not need large profit margins for drugs – and certainly not the bloated margins that pharmaceutical companies currently enjoy. The main financial goal of those platforms will be to maximize their number of subscribers because they could then steer them to other products, have additional advertising revenue, and obtain data that would allow them to create more accurate systems and services. As a result, drugs and health

services will be offered at a discount – like everything else.

The pharmaceutical sector cannot possibly compete with all that. The most fraudulent sector on earth cannot hope to compete with the most innovative and fastest-growing sector, which also happens to be much larger. However, big pharma will try to delay their displacement and obsolescence as much as possible and with all means possible, both legal and illegal. Their reckless disregard for human life and knowledge demonstrates that they will stop at nothing to secure profits and access to power.

We believe that big pharma will launch a coordinated attack against neural network research and related medical innovations. We think that their attack will utilize three main elements: public opinion, the government, and scientific literature.

Their strategy on public opinion will most likely take the form of a fearmongering campaign to create public alarm or concern about neural networks (or artificial intelligence, if the latter term persists). They will use the media and the internet to promote propaganda that neural networks and automation will lead to excessive unemployment and poverty; that medical services and treatment will be under the control of a new and possibly evil intelligence; that privacy will be eroded further by big technology corporations and their medical neural networks; and that those corporations will use all those systems to control the population.

The second element of big pharma's attack on neural networks is likely their strongest weapon. The pharmaceutical trade association has a large budget and spends exorbitant amounts on lobbying each year. As we explained in *Medical Dark Age* and section 9.5, big pharma has such extensive influence and control of Congress that it has been argued that "Congress is owned by pharma" and that "it really doesn't matter who the president is."

The pharmaceutical web of power and influence will be used against neural network research and the technology sector. Their main line of attack will be to mobilize the government against large technology corporations on various pretexts – accusing them of monopolies and violations of antitrust laws, tax evasion, unfair trade practices, privacy issues and violations, discrimination etc. Their second line of attack will be on neural network research for medical purposes. They will again use the government and they will try to pass laws that prevent or limit such applications on a pretext that they represent a liability and a threat to public health, a supposed destabilizing factor for the pharmaceutical market, and an unconscionable effort to replace life-saving vaccines with various unproven, questionable, and less effective methods.

The third area and target of the pharmaceutical industry's attack will be scientific literature and the scientific record. They have already introduced an extraordinary level of fallacies and fabrications into the medical literature and they will do the same for neural network research that relates to medicine and medical applications. For every publication that demonstrates an effective protein-based treatment for an infectious disease (for which there

are vaccines or expensive newly patented drugs) they will publish various biased or fabricated articles that supposedly show the opposite; they will claim that vaccine-based prevention is better than treatment; and they will conjure up all sorts of studies that ‘demonstrate’ the advantages of prevention and the potential adverse effects of protein treatments. For every diagnostic system that rejects over-prescription, over-medication, or over-vaccination, they will employ third parties to develop similar systems that supposedly support the opposite. They will furthermore attack the credibility of neural research, digging up questionable results as examples; they will claim that neural networks are black boxes that have no place in science; and they will argue that they introduce unethical biases and discrimination.

In this book, we will not discuss or refute fearmongering campaigns or other methods to manipulate public opinion. We believe that other researchers can provide a detailed and effective strategy for defending neural-network based medicine and medical research.

For the technology sector, we would like to point out a simple fact – and issue a warning. Innovation and strategy are the only things that matter. For technology companies, all other factors are negligible and irrelevant in the long run. This has been repeatedly demonstrated by various companies such as Apple, Google, Amazon, and Tesla. Technology companies do not need to evade taxes, employ unfair trade practices, sell their users’ data or compromise their privacy; and they certainly do not need to associate themselves with big pharma. Such actions will always be strategic blunders. They may have some short-term benefits and increase the bonuses of some executives, but in the long run they can have no other effect than to tarnish the public image of a corporation. The technology sector will be paramount in the 21st century. Individual corporations must rise to the occasion. Those that do not will simply be left behind.

The case with the government, and Congress in particular, seems hopeless to us. The level of corruption and influence from big pharma may already be beyond remedy. It is not merely hypocritical for various senators and representatives to be so focused and determined on various issues and alleged violations by technology companies while turning a blind eye to the pharmaceutical sector. They have permitted big pharma to settle one case of fraud after another, for negligible amounts compared to their profits from fraud, and they have tolerated an economic cost to society that is measured in trillions. They have allowed health care expenses to balloon to one fifth of the GDP of the nation. Worst of all, they have accepted an extensive record of human suffering and a death toll of millions. Those actions are not merely hypocritical. They constitute a breach of duty of elected officials to their constituents, and they are often the result of corruption and submission to big pharma.

The technology sector can easily outspend the pharmaceutical on lobbying activities, if they choose to do so. However, we believe that a better use

of funds will be to demonstrate the extent of influence by big pharma and the corruption in Congress with a media campaign, especially given that the pharmaceutical sector is bound to attack neural networks and the technology sector.

We can always invite the government to disassociate themselves from big pharma, remove protections that they have given to the pharmaceutical sector, implement new laws and prosecute pharmaceutical fraud as organized crime*, limit drug pricing and health expenses and bring them down to a level that is comparable with other nations etc. We could also invite the government to maintain a neutral stance in the conflict between the technology and pharmaceutical sector that will take place in the next few decades.

However, such invitations are naive. The government is too deep in pharmaceutical pockets, they will take sides, and they will lose credibility and power. The final impact to the nation and democracy is hard to estimate.[†]

In either case, it is imperative that we address a number of weaknesses in the field of neural networks not only because those will be used in an effort to attack and discredit the field but also because our field and its central role in the technology of the 21st century demands and deserves better.

We can argue that the attack on neural networks by the medical establishment and the pharmaceutical sector has already started – albeit in a limited and rather academic manner. For example, in 2020, Google researchers published a paper in the journal *Nature* about breast cancer screening using a neural network – which they called an AI system. They claimed that their system was “capable of surpassing human experts in breast cancer prediction.” A few months later, a group of more than 20 medical researchers, who were frustrated by the lack of details of the methods and access to the algorithm and the code, published a rebuttal in *Nature* that was centered around “transparency and reproducibility in artificial intelligence.” They argued that the absence of such important information undermined the “scientific value” of the publication, made replication virtually impossible, and restricted the ability for a proper peer review that could identify potential issues with the methodology.

*In section 9.5, we mentioned a proposal by business law professor E. McCarthy to prosecute drug company fraud as organized crime under the Racketeering Influenced and Corrupt Organizations Act of 1970.

[†]We anticipate that the U.S. military will rise as a fourth branch of government in this century. Historically, the military has played an important role in shaping the U.S. democracy; and it will do so once more in the coming decades. Climate change and an annual stream of environmental challenges, disasters, and crises will bring the military to a more prominent position as it is the main element of government and society that can adapt to those challenges effectively. Furthermore, there is a significant overlap and various opportunities between the military and technology but a clear antithesis between the military and the pharmaceutical sector – not only because the latter drains 20% of the GDP and thus leaves less room for military expenditures but also because the military does not welcome a future where most of its personnel require daily doses of autoimmune disorder drugs.

We largely agree with the rebuttal and we will not dispute it. We will not question the motivation of the authors either. On the contrary, we would like to believe that we all share a genuine interest in scientific discovery, truth, and the scientific method, and that the purpose of the rebuttal is to point out significant omissions in the paper that should be addressed.

However, we cannot help but notice the double standards. With the exception perhaps of J. Ioannidis, who is a coauthor of the rebuttal but has published extensively on the replication crisis, where are the publications, rebuttals, and letters of protests of the rest of the authors? Why are they silent and tacitly complicit with the falsities and fallacies that are published regularly in medical journals? Small or statistically negligible studies are common in medical literature. Clinical trials are largely a waste of resources and funds. Conflicts of interests and creative designs abound in medical research. Vaccine studies and publications are abysmal. Blind placebo tests have sometimes been replaced with tests with a reduced amount of an immunologic adjuvant. Children too young to be diagnosed with autism or too ill to be vaccinated are not always excluded from studies. Individual vaccines are targeted in studies but the claims are ‘magically’ transferred to all vaccines. Over-generalization takes place among cases that are not established as equivalent or compatible. Despite its extraordinary complexity and learning sophistication, the immune system is regularly reduced to a finite-state automaton in peer-reviewed and published vaccine articles. Even outrageous and utterly unsupported claims such that “healthy infants could safely get up to 100,000 vaccines at once” are published and promoted. When all those ‘scientific’ indiscretions take place, where are the rebuttals and protests of those academics and researchers? Where is their otherwise genuine interest for scientific truth? Are they hypocritical, corrupt, or misinformed? We understand that the medical literature and clinical studies are now so replete with bias, fallacies, and misinformation that it is virtually impossible to counter or refute all or most of them. We sympathize with the predicament of those researchers. However, a claim of safety for 10,000 or 100,000 vaccines simultaneously administered to an infant and not a single article of protest from the medical establishment? How can one subsequently trust such an establishment with anything? When researchers and academics have demonstrated such levels of complicity, can they still be trusted and be taken seriously? Can they still criticize others or have they lost such privilege – permanently?

We examined this unfortunate condition in more detail in *Medical Dark Age*. We provided various examples that demonstrated its extent and severity. Many more examples and related information can be found in other referenced publications.

Overall, we cannot do much about this invalid state of medicine, the new religion of false medicine, and the lax standards of various medical journals that permit an endless stream of fallacies and falsities to be published. Most of those conditions are a direct product of big pharma and they will probably

disappear with it. Neural networks and the technology sector will displace and diminish the entire pharmaceutical industry and thus put an end to all this. There will be great resistance, it will be a spectacular display, but the final outcome is inevitable.

On the other hand, we must put our own house in order. The popularity of neural networks and their number of publications have increased dramatically in the last decade. Papers are being published by researchers who have limited or even superficial understanding of the field. The quality of certain neural network publications is abysmal – especially publications in peripheral conferences on fields such as artificial intelligence or data mining. The replication crisis has developed a considerable footprint in the field. Measures of statistical significance are absent from many empirical studies. Various artifacts that are a consequence of specific conventions have been elevated to the level of principles or best practices. Even extreme claims (such as superhuman performance) have surfaced with little or no subsequent criticism. Furthermore, as we explained in section 8.8, there is an extensive publication bias that is independent of replication. All those issues must be addressed, if we do not wish neural networks to drift into an invalid state like medical research.

Another issue with neural networks is their *opacity*, which usually refers to the argument that the internal logic of a neural network is opaque or hard to understand for humans. One cannot look into the structure and parameters of a neural network and understand or reliably predict the response of the network across a set of inputs. The parameters of a neural network and their values have no obvious meaning – and they often have no meaning at all, if they are considered individually. It is their collective and statistical existence that gives meaning to them, but that is considerably harder to decipher.

As a result, neural networks have sometimes been labeled ‘black boxes’. There has been some associated criticism of the field. We can dismiss potential propaganda, but as it has been pointed out by other researchers, lack of transparency is a real issue. Rational causal explanations of the decisions of a neural system are essential for such a system to be accepted and trusted by users. Furthermore, there can be social consequences when a neural network is used for classification or ranking in a range of applications such as fraud detection, search engines, news and product recommendations, insurance or loan qualification, credit scoring, job applications etc.

The neural network community clearly understands the problem and there have been significant efforts to reduce or resolve it. Those fall under the generic terms of *transparency* and *accountability*. There is a considerable degree of literature on the topic. We anticipate that there will be significant progress in the coming years and decades. We will make specific contributions to this research in volume III.

However, we would like to point out a different aspect of this condition of opacity or limited transparency. Such a condition effectively refers to partial

or limited understanding of a system or subject. It is not unique to neural networks. On the contrary, it is quite common in natural science. For example, we have a very limited understanding of our brains and our nervous system; we do not fully understand the functions of cells; we cannot even tell how nature can fold proteins so effectively when there is an astronomical number of possibilities. Yet, we do not reject or even criticize neuroscience, medicine, and biology for those limitations. On the contrary, we accept those disciplines in their current condition and we try to make progress. Our treatment of neural networks cannot and should not be different in that regard. On the other hand, as a research community, we should not be groping in the dark indefinitely, driven exclusively or primarily by empiricism. We need a guiding theory. Mathesis represents our efforts in that direction.

9.7 Historical and Bibliographical Remarks

The origins of empiricism as an attitude can be traced to *Presocratics* such as Thales of Miletus (ca. 585 BCE), Heraclitus (ca. 500 BCE), or Empedocles (ca. 495–435 BCE) [136, 20] as well as the *Lokayata school of Indian philosophy* (6th century BCE) and the Buddha (ca. 485–405 BCE) [49, 175]. Aristotle formulated empiricism into a doctrine [10, 11, 12] and proposed a *tabula rasa* version of it denying innate ideas altogether. In his treatise *On the Soul* [11, 3.4.430], he stated:

“What it [the mind] thinks must be in it just as characters may be said to be on a writing tablet on which as yet nothing actually stands written: this is exactly what happens with mind. ”

Most of the above information and a thorough presentation of the history of empiricism appears in [72].

Pythagoras was the first Western philosopher who held a rationalistic view, assuming that certain mathematical relationships represented the ultimate nature of reality. Socrates and Plato supported the view that knowledge is innate using arguments from geometry. Aristotle introduced *sylogistic logic*, which asserts that humans truly understand a phenomenon or fact only when they can describe it as an instance of a broader and self-evident principle. A review of rationalism and its history appears in [29].

The first known use of the term “rationalism” appeared circa 1732 while the term “rationalist” as “one who follows reason and not authority in thought or speculation” appeared in the 1620s [83, 396].

Rationalism eventually led to the intellectual, philosophical, scientific, social, and political movement that is known as the *Age of Reason* (a.k.a. *Age of Enlightenment*) that dominated Europe during the 17th and 18th centuries [193].

R. Descartes was the first modern rationalist. He attempted to introduce the clarity and certainty of mathematics (and geometry in particular) into

philosophy and reasoning [29]. G. Leibniz adopted a similar view.

Aristotle expressed his view about the limitations of perception (“not everything that appears is true”) in *Metaphysics* [10, 1010b2–3]. He distinguished between sensations and human perception and judgement and he indicated the fallible aspects of human nature.

The earliest publication that identified the issue of false conclusions from errors in statistical tests of significance appeared in 1959 [327].

One of the earliest publications that indicated the replication crisis in science – and indirectly implied the necessity of metascience – appeared in 1966 [305]. The study examined 295 papers published in ten peer-reviewed medical journals and found that “conclusions were drawn when the justification for these conclusions was invalid” in about 73% of the reports.

Ioannidis’ seminal paper on the replication crisis and falsities in medical research appeared in 2005 [171]. A subsequent study about reducing the waste in biomedical research with Ioannidis as a coauthor appeared in 2014 [224]. Ioannidis published a more extensive paper about wasteful or useless research – which he generously called “not useful” – on the same topic two years later [172]. The economic study on the cost of such wasteful and non-reproducible research was published in 2018 [116].

The term Proteus phenomenon was coined by J. Ioannidis and T. Trikalinos [173]. It was named after the Hellenic deity Proteus who could rapidly change his appearance.

Publication bias is discussed in detail in [263]. The report intends to identify and appraise empirical studies on publication and related biases published since 1998. The bias for statistically significant positive results and the aversion to null results is discussed in [107]. Design quality is often comparable between studies with significant results and studies with a null result [101]. However, the former studies have been shown to be about three times more likely to be published [81]. A publication bottleneck has been attributed to the limited number of publication outputs compared to the much larger number of research studies [125]. It has similarly been argued that journals increasingly derive their authority from selectivity, create an *artificial scarcity** of publication opportunities, an illusion of exclusivity, and more competition among scientists [400].

Selective reporting regarded as a questionable research practice is discussed in [177]. Flexibility in data analysis and the associated biases are discussed in [317] and the productivity and novelty pressure in [223]. The desire for media attention and a plea for more epistemological modesty appear in [31]. A data analysis that demonstrates a general preference for non-replicable results (being cited more often than replicable ones) is presented [309].

An early analysis of the effects of publication bias in rejecting the null

*In [400], artificial scarcity is defined as “any situation where, even though a commodity exists in abundance, restrictions of access, distribution, or availability make it seem rare, and thus overpriced.”

hypothesis appears in [327].

Scientific misconduct rates have been estimated using a meta-analysis of 39 research surveys containing relevant misconduct questions [106]. The estimated rates are assumed to be conservative.

An analysis of 53 pre-clinical cancer studies with a replication rate of just 11% appears in [127].

An analysis of a survey of the journal *Nature* about reproducibility appears in [17].

Information about the Center of Open Science can be found in [114, 382]. Their study about reproducibility in psychology appears in [60].

Information about DARPA's SCORE program can be found in [390]. The announcement that the Center for Open Science joined the SCORE program is available in [113].

A. De Menard's blog entry about the replication crisis in social science can be found in [232]. As early as 2002, it was argued that papers are cited without being read [316]. The authors applied stochastic modeling to the citation process and misprints in citations (arguing that those are similar to Freudian slips) and they estimated that cited papers were read prior to citation in only about 20% of the cases.

Hanlon's razor, its origin (as a submission to a joke book), and its variations are discussed in a Wikipedia article [383]. In the same article, it is stated that earlier versions of the razor were provided by W. James and R. Heinlein.

Information about the FCA can be found at the Department of Justice and on Wikipedia [255, 384]. The website www.whistleblowingprotection.org also contains information on the topic as well more information about various cases of fraud, whistleblowers, qui tam action, and a qui tam timeline [364].

Fraud recovery statistics under the FCA are provided by the Department of Justice [58]. The term medical-industrial complex first appeared in 1969 [102]. It was used in various publication during the 1970s and it became more common after it was used by A. S. Relman in 1980 [278].

Detailed information and statistics about the opioid epidemic can be found in government sites and on Wikipedia, too [248, 348, 387, 388].

Data and an analysis of health care costs as percentage of GDP across developed nations can be found in [340]. Our estimates about US health care spending in 2020 are from the Centers for Medicare and Medicaid Services, which is an official website of the United States [111]. A comparative analysis of quality of the U.S. health care system using various indicators can be found in [202].

In 2002, A. S. Relman and M. Angell published a paper about unscrupulous practices, fraud, and political influence or corruption in the pharmaceutical industry that earned them the prestigious George Polk award in journalism [279]. M. Angell published a more detailed book about the same topic in 2005 [7]. In both cases, the authors proposed various measures to address or reduce the problem but their recommendations have had no effect on the

government.

In 2009, M. Angel published an article that explained and demonstrated in detail the virtual control of the pharmaceutical industry over medicine and medical research [8].

Four years later, P. C. Götzsche published a remarkable book. In *Deadly Medicines and Organised Crime: How Big Pharma has Corrupted Healthcare*, he makes the case that the pharmaceutical industry is organized crime and it operates like organized crime [133]. The arguments are supported by numerous examples and detailed in-depth evidence.

In 2019, E. McCarthy, assistant professor on business law at James Madison University, published a call to prosecute drug company fraud as organized crime under the Racketeering Influenced and Corrupt Organizations Act of 1970 [227].

Information about PhRMA and our numbers of their 2017 revenue and lobbying expenditures are from Wikipedia [385]. Information and statistics about prescription drug spending in the United States and national health expenditures can be found in [254, 112]

Information about DeepMind's AlphaFold and its distinction as 2021 Breakthrough of the Year by Science magazine can be found in [336, 343, 310]. AlphaFold's first discovery of a new drug was announced in January 2022 [280].

The paper by Google researchers for breast cancer screening using a neural network and its rebuttal were published in 2020 [229, 143].

List of Figures

1.1	Generic definition of neural networks.	11
1.2	Cross-connections for expressions of varying depths.	13
1.3	Layers with alternative binding.	16
1.4	Examples of sigmoids and their derivatives.	19
1.5	A convolutional neural network for vision.	28
1.6	Distribution of cones and rods along a line passing through the fovea and the blind spot of a human eye.	32
2.1	Potential dependency paths between the loss ℓ and outputs Y_k	46
2.2	Neural network stochastic gradient descent.	49
2.3	π -set construction.	50
2.4	Memory efficient neural gradient descent.	50
3.1	Anatomy of a neuron.	70
3.2	Synapse types.	71
3.3	Structure of a neurons and its axon.	79
4.1	Human brain structure and function.	88
4.2	Graph cuts and peripheries.	92
4.3	The synergy conjecture.	128
4.4	The diversity selection algorithm.	136
4.5	The incremental diversity algorithm.	137
4.6	The dynamic diversity algorithm.	137
4.7	The evolutionary diversity algorithm.	138
5.1	The sine function and an approximating parabola.	169
5.2	Polynomial fit detects the parabola in the data.	173
5.3	A single offset introduced functional distinction and poor ap- proximation outside the training interval.	174
5.4	Runge's phenomenon. The function $1/(1+x^2)$ is approximated by two polynomials of degree six and ten, respectively. The polynomials are derived from equally spaced samples in $[-5, 5]$	176
5.5	Chebyshev nodes eliminate Runge's phenomenon. They guar- antee that the approximation error converges to zero as the degree of the polynomial approximation goes to infinity.	177

5.6	Chebyshev nodes do not address approximation error outside the interpolation interval.	177
5.7	Approximating diagonal data with sinusoid displacement or 'noise' using 11 Chebyshev nodes in the interval $[-5, 5]$ and polynomials p_5 and p_9 of degree 5 and 9, respectively.	182
5.8	Extending the plot of the approximating polynomials p_5 and p_9 (of degree 5 and 9, respectively) to the range $[-6, 6]$	182
5.9	An image of gray scale color gradient on the left and a random permutation of pixels blocks in it.	187
6.1	The coefficient c_0 is disproportionately large in exponential smoothing (left plot). The number of coefficients that are less or equal to c_0 increases quickly as γ tends to 1 (right plot). .	222
6.2	The unbiased exponential smoothing algorithm.	224
7.1	Synaptic cross-gradient coherence.	252
7.2	The Sync algorithm.	253
7.3	First version of the Sync algorithm.	258
7.4	The Sync algorithm.	261
8.1	A sunflower with five petals $C \cup P_i$, for $i < 5$	267
8.2	A flower with five petals $C \cup P_i$, for $i < 5$	267
8.3	Flower structure of three successive layers defined by two linear sunflowers.	270
8.4	Flower structure of three successive layers defined by two flowers.	271
8.5	A Venn diagram and filter connectivity of the first three layers of a flower net. Only a single filter for each segment of the third flower is shown in the graph to make the structure more visible.	272
8.6	Average test error and confidence intervals of the SEF ensemble on MNIST.	304
8.7	Average test error rates and confidence intervals of the SE and SEF networks and ensembles on MNIST.	305
9.1	FCA recovered amounts (in billions US dollars), October 1, 1986 – September 30, 2020.	325
9.2	Health care costs from 1980 to 2018 as a percentage of GDP, adjusted for differences in cost of living across different nations. Source: Organization for Economic Cooperation and Development (OECD) Health Statistics 2019.	326

List of Tables

1.1	Typical activation functions.	20
2.1	Common loss functions.	41
2.2	Common f -divergence functions.	54
2.3	Common Bregman divergence functions.	55
4.1	Examples of taxonomic biological classification.	120
4.2	Taxonomic ranks of biological classification.	121
5.1	Percentage of the MNIST test set in its training convex hull with reduced images.	179
5.2	Various forms of coherence in computer graphics.	186
6.1	Examples of cumulative correlations and the equivalent first- order recursions.	227
8.1	Output shapes and number of parameters for the layers of the MNIST SE model.	285
8.2	Number of parameters of the SEF networks along with their β values, their initial and final flowers, and their multiplier arrays.	298
8.3	Flower parameters for each flower in the SEF ensemble.	299
8.4	Samples and pairs with identical minimum bounding boxes for each class in the MNIST training set.	301
8.5	Learning schedule for the SEF ensemble. S+A indicates a combination of superposition data and data augmentation.	303

Bibliography

- [1] Top 500. The green 500. <https://www.top500.org/green500/lists/2019/11/>, November 2019. Online, accessed 17-June-2020.
- [2] A. Abdulkader, J. A. Drakopoulos, and Q. Zhang. Comparative classifier aggregation. In *18th International Conference on Pattern Recognition (ICPR'06)*, volume 3, pages 156–159, 2006.
- [3] S. Acharyya, A. Banerjee, and D. Boley. Bregman divergences and triangle inequality. In S. Parthasarathy, J. Ghosh, Z.-H. Zhou, J. Dy, Z. Obradovic, and C. Kamath, editors, *Proceedings of the 2013 SIAM International Conference on Data Mining*, Proceedings of the 2013 SIAM International Conference on Data Mining,, pages 476–484. Siam Society, January 2013.
- [4] W. Ackermann. Zum hilbertschen aufbau der reellen zahlen. *Mathematische Annalen*, 99(1):118–133, 1928.
- [5] J. J. G. Adeva, U. C. Beresi, and R. A. Calvo. Accuracy and diversity in ensembles of text categorisers. *CLEI Electronic Journal*, 8, 2005.
- [6] J.-M. Alonso and Y. Chen. Receptive field. http://www.scholarpedia.org/article/Receptive_field, 2009. Online, accessed 3-February-2020.
- [7] M. Angell. *The Truth about the Drug Companies: How They Deceive Us and What to Do about It*. Random House, New York, 2005.
- [8] M. Angell. Drug companies & doctors: a story of corruption. *The New York Review of Books*, 56(1):8–12, 2009. Also available online at <http://www.fondazionediabella.org/cms-web/upl/doc/Documenti-inseriti-dal-2-11-2007/Truth%20About%20The%20Drug%20Companies.pdf>.
- [9] Clay Mathematics Institute Historical Archive. The thirteen books of Euclid’s Elements. <https://www.claymath.org/library/historical/euclid/>, May 8, 2008. Online, accessed 8-August-2020.
- [10] Aristotle. Metaphysics, c. 350 BCE.
- [11] Aristotle. On the Soul, c. 350 BCE.
- [12] Aristotle. On the Soul. Parva Naturalia. On Breath, c. 350 BCE. Translated by W. S. Hett. Harvard University Press. Cambridge, MA, 1957.

- [13] Aristotle. Physics, c. 350 BCE.
- [14] Aristotle. Posterior analytics, c. 350 BCE.
- [15] D. Attwell and S. B. Laughlin. An energy budget for signaling in the grey matter of the brain. *Journal of cerebral blood flow and metabolism*, 21(10):1133–1145, October 2001.
- [16] H. G. Ayad and M. S. Kamel. Cluster-based cumulative ensembles. In N. C. Oza, R. Polikar, F. Roli, and J. Kittler, editors, *6th International Workshop on Multiple Classifier Systems, Lecture Notes in Computer Science*, volume 3541, pages 236–245. Springer, 2005.
- [17] M. Baker. 1,500 scientists lift the lid on reproducibility. *Nature News*, 533(7604):452, 2016.
- [18] G. Balavoine and A. Adoutte. The segmented urbilateria: a testable scenario. *Integrative and comparative biology*, 43(1):137–47, February 2003. doi:10.1093/icb/43.1.137.
- [19] R. A. Barco, G. M. Garrity, J. J. Scott, J. P. Amend, K. H. Nealson, and D. Emerson. A genus definition for bacteria and archaea based on a standard genome relatedness index. *mBio*, 11(1), 2020. <https://mbio.asm.org/content/11/1/e02475-19>. arXiv:<https://mbio.asm.org/content/11/1/e02475-19.full.pdf>, doi:10.1128/mBio.02475-19.
- [20] J. Barnes. *The Presocratic Philosophers, The Arguments of the Philosophers*. Routledge, 1982.
- [21] R. Bartels, J. Beatty, and B. Barsky. *An Introduction to Splines for Use in Computer Graphics and Geometric Modeling*. Morgan Kaufmann, 1st edition, 1996.
- [22] J. S. Bendat and A. G. Piersol. *Random Data: Analysis and Measurement Procedures*. Wiley-Interscience, 3rd edition, 200.
- [23] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin. A neural probabilistic language model. *The Journal of Machine Learning Research*, 3:1137–1155, 2003.
- [24] A. K. Bhattacharyya. On a measure of divergence between two statistical populations defined by their probability distributions. *Bulletin of the Calcutta Mathematical Society*, 35:99–109, 1943.
- [25] A. K. Bhattacharyya. On a measure of divergence between two multinomial populations. *Sankhya: The Indian Journal of Statistics (1933-1960)*, 7(4):401–406, 1946. URL: <http://www.jstor.org/stable/25047882>.

- [26] E. Bianconi, A. Piovesan, F. Facchin, A. Beraudi, R. Casadei, F. Frabetti, L. Vitale, M. C. Pelleri, S. Tassani, F. Piva, S. Perez-Amodio, P. Strippoli, and S. Canaider. An estimation of the number of cells in the human body. *Annals of human biology*, 40(6):463–471, 2013.
- [27] G. Birkhoff. Integration of functions with values in a Banach space. *Transactions of the American Mathematical Society*, 38(2):357–378, 1935. doi:10.2307/1989687.
- [28] Christopher Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.
- [29] B. Blanshard. Rationalism. <https://www.britannica.com/topic/rationalism>, 2020. Online, accessed 13-May-2021.
- [30] A. L. Blum and R. L. Rivest. Training a 3-node neural network is NP-complete. *Neural Networks*, 5(1):117–127, 1992. doi:10.1016/S0893-6080(05)80010-3.
- [31] P. Boffetta, J. K. McLaughlin, C. La Vecchia, R.R Tarone, Loren L. Lipworth, and W. J. Blot. False-positive results in cancer epidemiology: a plea for epistemological modesty. *Journal of the National Cancer Institute*, 100(14):988–995, 2008.
- [32] Brahmagupta. Brahmasphutasiddhanta. http://grettil.sub.uni-goettingen.de/grettil/1_sanskr/6_sastra/8_jyot/brsphutu.htm, 628 CE. Online, accessed 8-August-2020.
- [33] L. M. Bregman. The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming. *USSR Computational Mathematics and Mathematical Physics*, 7(3):200–217, 1967. URL: <http://www.sciencedirect.com/science/article/pii/0041555367900407>, doi:10.1016/0041-5553(67)90040-7.
- [34] L. Breiman. Bagging predictors. Technical Report 421, Department of Statistics, University of California Berkeley, September 1994.
- [35] Gavin Brown. *Diversity in neural network ensembles*. PhD thesis, University of Birmingham, Birmingham, United Kingdom, 2004.
- [36] R. G. Brown. Exponential smoothing for predicting demand. In *Operations Research*, volume 5, pages 145–145, 1957.
- [37] J. Browne. *Charles Darwin: Voyaging*. Princeton University Press, Princeton, April 1996.
- [38] J. Browne. *Charles Darwin: The Power of Place*. Princeton University Press, Princeton, October 2003.

- [39] A. Bryson. A gradient method for optimizing multi-stage allocation processes. In *Proceedings of the Harvard University Symposium on digital computers and their applications*, 1961.
- [40] S. Bubeck and M. Sellke. A universal law of robustness via isoperimetry. *Advances in Neural Information Processing Systems*, 34, 2021.
- [41] S. Butterworth. On the theory of filter amplifiers. *Wireless Engineer*, 7(6):536–541, 1930.
- [42] I. Bárány and C. Buchta. Random polytopes in a convex polytope, independence of shape, and concentration of vertices. *Mathematische Annalen*, 297:467–497, 1993. doi:10.1007/BF01459511.
- [43] W. B. Cannon. Physiological regulation of normal states: some tentative postulates concerning biological homeostatics. In A. Pettit, editor, *A Charles Riches amis, ses collègues, ses élèves*, page 91. Les Éditions Médicales, Paris, 1926.
- [44] Georg Cantor. Ueber eine eigenschaft des inbegriffs aller reellen algebraischen zahlen. *Journal für die reine und angewandte Mathematik*, 77:258–262, 1874.
- [45] Augustin-Louis Cauchy. Methode generale pour la resolution des systemes d’equations simultanees. *Comptes Rendus Hebdomadaires des Seances de l’Academie des Sciences*, 25:536–538, 1847.
- [46] T. Cavalier-Smith. Kingdom chromista and its eight phyla: a new synthesis emphasising periplastid protein targeting, cytoskeletal and periplastid evolution, and ancient divergences. *Protoplasma*, 255:297–357, January 2018. <https://doi.org/10.1007/s00709-017-1147-3>. doi:10.1007/s00709-017-1147-3.
- [47] P. Charbonnier, L. Blanc-Feraud, G. Aubert, and M. Barlaud. Deterministic edge-preserving regularization in computed imaging. *IEEE Transactions on Image Processing*, 6(2):298–311, 1997.
- [48] M. J. Charlesworth. Aristotle’s razor. *Philosophical Studies*, 6:105–112, 1956. doi:10.5840/philstudies1956606.
- [49] S. Chatterjee and D. M. Datta. *An introduction to Indian philosophy*. University of Calcutta, 1954.
- [50] P. Chebyshev. *Théorie des mécanismes connus sous le nom de parallélogrammes*. Imprimerie de l’Académie impériale des sciences, 1853.
- [51] K. Chellapilla, P. Simard, and A. Abdulkader. Allograph based writer adaptation for handwritten character recognition. In G. Lorette, editor, *Tenth International Workshop on Frontiers in Handwriting Recognition*,

- La Baule, France, October 2006. Université de Rennes 1, Suvisoft. <https://hal.inria.fr/inria-00112632>.
- [52] Noam Chomsky. Approaching UG from below. In H.-M. Gärtner and U. Sauerland, editors, *Interfaces + Recursion = Language? Chomsky's Minimalism and the View from Syntax-Semantics*, Studies in Generative Grammar, Berlin, 2007. Mouton de Gruyter.
- [53] Noam Chomsky. The galilean challenge: Architecture and evolution of language. *Journal of Physics: Conference Series*, 880:12–16, August 2017. doi:10.1088/1742-6596/880/1/012015.
- [54] A. Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58:345–363, 1936.
- [55] D. Ciregan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3642–3649. IEEE, 2012.
- [56] D. Ciresan, U. Meier, L. Gambardella, and J. Schmidhuber. Deep big simple neural nets for handwritten digit recognition. *Neural Computation*, 12(22):3207–3220, 2010. arXiv:1003.0358, <https://arxiv.org/abs/1003.0358>. doi:10.1162/NECO_a_00052.
- [57] D. Ciresan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*, volume 2, pages 1237–1242, 2011.
- [58] Civil Division, U.S. Department of Justice. Fraud statistics. <https://www.justice.gov/opa/press-release/file/1354316/download>, 2021. Online, accessed 8-January-2022.
- [59] R. N. Clark. Notes on the resolution and other details of the human eye. <https://clarkvision.com/imagetdetail/eye-resolution.html>, January 2005. Online, updated 28-December-2018, accessed 28-March-2021.
- [60] Open Science Collaboration et al. Estimating the reproducibility of psychological science. *Science*, 349(6251), 2015. Also available in https://ppw.kuleuven.be/okp/_pdf/Nosek2015ETROP.pdf.
- [61] R. Collobert, S. Bengio, and J. Mariethoz. Torch. <http://torch.ch>, October 2002. Online, accessed 11-January-2020.
- [62] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12:2493–2537, November 2011.

- [63] O. F. Cook. Factors of species-formation. *Science*, 23(587):506–507, 1906.
- [64] H. Cramér. Sur un nouveau théorème-limite de la théorie des probabilités. *Act. Sci. et Ind.*, 736, 1938.
- [65] A. C. Crombie. *Medieval and Early Modern Philosophy. Vol. II*. Doubleday Anchor Books, 1959.
- [66] Y. Le Cun, J. S. Denker, and S. A. Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems 2*, page 598–605. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990.
- [67] P. Cunningham and J. Carney. Diversity versus quality in classification ensembles based on feature selection. In R. López de Mántaras and E. Plaza, editors, *Machine Learning: ECML 2000*, pages 109–116, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.
- [68] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei. Deformable convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 764–773, 2017.
- [69] G. B. Dantzig and D. R. Fulkerson. On the max flow min cut theorem of networks. RAND Corporation, Santa Monica, CA, 1955.
- [70] Charles Darwin. *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*. John Murray, November 24, 1859.
- [71] B. V. Dasarathy and B. V. Sheela. A composite classifier system design: concepts and methodology. *Proceedings of the IEEE*, 67(5):708–713, 1979.
- [72] G. W. Dawes. Ancient and medieval empiricism. In E. N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Winter 2017 edition, 2017.
- [73] R. Dawkins. *The Blind Watchmaker: Why the Evidence of Evolution Reveals a Universe Without Design*. Norton & Company, Inc., 1986.
- [74] Marquis de Condorcet. Essai sur l’application de l’analyse à la probabilité des décisions rendues à la pluralité des voix. <https://gallica.bnf.fr/ark:/12148/bpt6k417181>, 1785. Online, accessed 1-August-2020.
- [75] Marquis de Condorcet. The plant list. <http://www.theplantlist.org/>, 2013. Online, accessed 5-August-2020.
- [76] D. Petri de Fermat. *Varia Opera Mathematica*, 1679. results derived in 1629; circulated in manuscript form in 1636.

- [77] W. Dean. Recursive functions. In E. N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Spring 2021 edition, 2021. Online, accessed 23-February-2021.
- [78] R. Dedekind. *Was sind und was sollen die Zahlen?* Braunschweig. Vieweg, 1888. English translation in W. W. Beman's *Dedekind, Essays on the Theory of Numbers*, Chicago, Open Court, 1901, 29–115.
- [79] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. IEEE, 2009.
- [80] M. Deza and P. Frankl. Every large set of equidistant $(0, +1, -1)$ -vectors forms a sunflower. *Combinatorica*, 1(3):225–231, 1981.
- [81] K. Dickersin, S. S. Chan, T. C. Chalmers, H.S. Sacks, and Jr. H. Smith. Publication bias and clinical trials. *Controlled Clinical Trials*, 8(4):343–353, 1987.
- [82] Merriam-Webster.com Dictionary. Sigmoid. <https://www.merriam-webster.com/dictionary/sigmoid>, 2020. Online, accessed 11-January-2020.
- [83] Merriam-Webster.com Dictionary. Rationalism. <https://www.merriam-webster.com/dictionary/rationalism>, 2021. Online, accessed 17-May-2021.
- [84] E. Dimitriadou, A. Weingessel, and K. Hornik. A cluster ensembles framework. In *Design and application of hybrid intelligent systems*. IOS Press, Amsterdam, 2003.
- [85] L. L. Dines. On convexity. *American Mathematical Monthly*, 45(4):199–209, 1938. doi:10.2307/2302604.
- [86] Diophantus. *Arithmetica*, 3rd Century CE. Latin translation: *Diophanti Alexandrini, Arithmeticonum Liber V*. Claude-Gaspar Bachet de Méziriac, 1621.
- [87] A. Dominguez-Torres. The origin and history of convolution i: continuous and discrete convolution operations. *Applied Mathematics and Computing Group. Cranfield Institute of Technology, Bedford, UK*, 2010.
- [88] D. H. Douglas and T. K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10(2):112–122, 1973.
- [89] R. J. Douglas, K. A. C. Martin, and D. Whitteridge. A canonical microcircuit for neocortex. *Neural Computation*, 1:480–488, 1989.

- [90] J. A. Drakopoulos and A. Abdulkader. Training neural net-works with heterogeneous data. *Neural Networks*, 18(5):595–601, 2005. doi: 10.1016/j.neunet.2005.06.011.
- [91] John A. Drakopoulos. *Theory and design of a hybrid pattern recognition system*. PhD thesis, Stanford University, Stanford, CA, May 1995.
- [92] John A. Drakopoulos. *Neural Age*. Kindle Direct Publishing, September 2017. Online at http://fairkryn.com/cetera/neural_age.html, accessed 19-December-2019.
- [93] S. E. Dreyfus. The numerical solution of variational problems. *Journal of Mathematical Analysis and Applications*, 5(1):30–45, 1962.
- [94] S. E. Dreyfus. The computational solution of optimal control problems with time lag. *IEEE Transactions on Automatic Control*, 18(4):383–385, 1973.
- [95] P. Dubey. Inefficiency of Nash equilibria. Technical Report WP-83-074, International Institute for Applied Systems Analysis, July 1983. URL: <https://core.ac.uk/download/pdf/33893709.pdf>.
- [96] P. Dubey. Inefficiency of Nash equilibria. *Mathematics of Operations Research*, 11(1):1–8, 1986.
- [97] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(7), 2011.
- [98] Richard O. Duda and Peter E. Hart. *Pattern Recognition and Scene Analysis*. Wiley Interscience, 1973.
- [99] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern classification*. Wiley, New York, 2nd edition, 2001.
- [100] H. M. Dunsworth. Origin of the genus homo. *Evolution: Education and Outreach*, 3(3):353–366, 2010. doi:10.1007/s12052-010-0247-8.
- [101] P. J. Easterbrook, R. Gopalan, J. A. Berlin, and D. R. Matthews. Publication bias in clinical research. *The Lancet*, 337(8746):867–872, 1991.
- [102] B. Ehrenreich and J. Ehrenreich. The medical industrial complex. Bulletin of the Health Policy Advisory Center, <http://www.healthpacbulletin.org/CompleteBulletinRun/Health%20PAC%20Bulletin%201969%20Nov.pdf>, November 1969. Online, accessed 6-January-2022.
- [103] H. B. Enderton. *A Mathematical Introduction to Logic*. Elsevier, 2nd edition, 2001.

- [104] H. B. Enderton. *Computability Theory: An Introduction to Recursion Theory*. Academic Press, 2010.
- [105] P. Erdős and R. Rado. Intersection theorems for systems of sets. *Journal of the London Mathematical Society*, 1(1):85–90, 1960.
- [106] D. Fanelli. How many scientists fabricate and falsify research? a systematic review and meta-analysis of survey data. *PLOS One*, 4(5):1–11, May 2009. <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7241762/>. doi:10.1371/journal.pone.0005738.
- [107] C. J. Ferguson and M. Heene. A vast graveyard of undead theories: Publication bias and psychological science’s aversion to the null. *Perspectives on Psychological Science*, 7(6):555–561, 2012.
- [108] J. R. Firth. A synopsis of linguistic theory 1930-1955. *Studies in Linguistic Analysis (special volume of the Philological Society)*, 1952-59:1–32, 1957.
- [109] John R. Firth. The technique of semantics. *Transactions of the Philological Society*, 34(1):36–72, 1935. doi:10.1111/j.1467-968X.1935.tb01254.x.
- [110] R. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2):179–188, 1936.
- [111] Centers for Medicare and Medicaid Services. National health expenditure data/historical. <https://www.cms.gov/Research-Statistics-Data-and-Systems/Statistics-Trends-and-Reports/NationalHealthExpendData/NationalHealthAccountsHistorical>, December 15, 2021. Online, accessed 11-January-2022.
- [112] Centers for Medicare & Medicaid Services. National health expenditure fact sheet, December 15, 2021. Online, accessed 14-January-2022.
- [113] Center for Open Science. Can machines determine the credibility of research claims? the center for open science joins a new darpa program to find out. <https://www.cos.io/about/news/can-machines-determine-credibility-research-claims-center-open-science-joins-new-darpa-program-find-out>.
- [114] Center for Open Science. Center for open science. <https://www.cos.io/>, 2013.
- [115] J. Franklin. *The Science of Conjecture: Evidence and Probability before Pascal*. The Johns Hopkins University Press, 2001.
- [116] L. P. Freedman, I. M. Cockburn, and T. S. Simcoe. The economics of reproducibility in preclinical research. *PLOS Biology*, 13(6):1–9, 06 2015. doi:10.1371/journal.pbio.1002165.

- [117] E. Freese. On the evolution of the base composition of dna. *Journal of Theoretical Biology*, 3(1):82–101, July 1962. doi:doi:10.1016/S0022-5193(62)80005-8.
- [118] Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *International Conference on Machine Learning*, volume 96, pages 148–156, 1996.
- [119] C. A. Frye. Cells of the nervous system. <https://www.albany.edu/faculty/cafrye/apsy601/Ch.02cellsofthenervoussystem.html>, 1999. Online, accessed 3-June-2020.
- [120] K. Fukushima. Neocognitron: a self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, 1980. doi:10.1007/BF00344251.
- [121] P. Földiák. Sparse coding in the primate cortex. In M. A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*, pages 1064–1068. MIT Press, second edition, 2002.
- [122] Carl F. Gauss. *Disquisitiones arithmeticae*, 1801. English translation by A. Clarke. Yale University Press, 1965.
- [123] J. D. Gergonne. The application of the method of least squares to the interpolation of sequences. *Historia Mathematica*, 1(4):439–447, 1815. doi:10.1016/0315-0860(74)90034-2.
- [124] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal on Robotics and Automation*, 4(2):193–203, 1988. doi:10.1109/56.2083.
- [125] R. Giner-Sorolla. Science or art? how aesthetic standards grease the way through the publication bottleneck but undermine science. *Perspectives on Psychological Science*, 7(6):562–571, 2012.
- [126] GitHub-Matuzas77. Mnist-0.17: MNIST classifier with average 0.17% error. <https://github.com/Matuzas77/MNIST-0.17>, 2020. Online, accessed 8-August-2021.
- [127] B. C. Glenn and L. M. Ellis. Drug development: Raise standards for preclinical cancer research. *Nature*, 483(7391):531–33, 2012.
- [128] Mark A. Gluck, Eduardo Mercado, and Catherine E. Myers. *Learning and Memory. From Brain to Behavior*. Worth Publishers, New York, 3rd edition, 2016.
- [129] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-wesley, 1989.

- [130] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680, December 2014.
- [131] L. A. Goodman and H. O. Hartley. The precision of unbiased ratio-type estimators. *Journal of the American Statistical Association*, 53(282):491–508, 1958.
- [132] S. Goonatilake. *Toward a Global Science: Mining Civilizational Knowledge*. Indiana University Press, 1998.
- [133] P. C. Gøtzsche. *Deadly Medicines and Organised Crime: How Big Pharma has Corrupted Healthcare*. Taylor & Francis group, 2013.
- [134] S. J. Gould. Opus 200. *Natural History*, 100(8):12–18, August 1991.
- [135] S. J. Gould and N. Eldredge. Punctuated equilibria: an alternative to phyletic gradualism. *Models in Paleobiology*, pages 82–115, 1972.
- [136] D. W. Graham. *The Texts of Early Greek Philosophy: The Complete Fragments and Selected Testimonies of the Major Presocratics*. Cambridge University Press, Cambridge, 2010.
- [137] M. E. Gröller and W. Purgathofer. Coherence in computer graphics. Technical Report TR-186-2-95-04, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/E193-02, A-1040 Vienna, Austria, March 1995. URL: <https://www.cg.tuwien.ac.at/research/publications/1995/Groeller-1995-CCG/>.
- [138] C. G. Gross. Claude bernard and the constancy of the internal environment. *Neuroscientist*, 4:380–385, 1988.
- [139] A. Guha, P. M. Haluptzokk, and J. A. Pittman. Handwriting and speech recognizer using neural network with separate start and continuation output scores. US patent 6393395, May 21, 2002.
- [140] Kurt Gödel. Über formal unentscheidbare sätze der Principia Mathematica und verwandter systeme, i. *Monatshefte für Mathematik und Physik*, 38(1):173–198, 1931.
- [141] Kurt Gödel. On undecidable propositions of formal mathematical systems. Lecture notes taken by Kleene and Rosser at the Institute for Advanced Study; reprinted in Davis 1965: 39–74, 1934.
- [142] J. Hadamard. Sur les problèmes aux dérivées partielles et leur signification physique. In *Princeton University Bulletin*, pages 49–52. Princeton University, 1902.

- [143] B. Haibe-Kains, G. A. Adam, A. Hosny, F. Khodakarami, Massive Analysis Quality Control (MAQC) Society Board of Directors, L. Waldron, B. Wang, C. McIntosh, A. Goldenberg, A. Kundaje, C. S. Greene, T. Broderick, M. M. Hoffman, J. T. Leek, K. Korthauer, W. Huber, A. Brazma, J. Pineau, R. Tibshirani, T. Hastie, J. P. A. Ioannidis, J. Quackenbush, and H. J. W. L. Aerts. Transparency and reproducibility in artificial intelligence. *Nature*, 586(7829):E14–E16, 2020.
- [144] P. Halmos. *Measure Theory*. Van Nostrand and Co., 1950.
- [145] P. Haluptzok, M. Revow, and A. Abdulkader. Personalization of an online handwriting recognition system. In *Tenth International Workshop on Frontiers in Handwriting Recognition*, La Baule (France), October 2006. Suvisoft.
- [146] L. K. Hansen and P. Salamon. Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):993–1001, 1990.
- [147] Z. S. Harris. Distributional structure. *WORD*, 10(2-3):146–162, 1954. <https://doi.org/10.1080/0437956.1954.11659520>. doi:10.1080/00437956.1954.11659520.
- [148] H. Hartley and A. Ross. Unbiased ratio estimators. *Nature*, 174:270–271, 1954. doi:10.1038/174270a0.
- [149] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, New York, NY, USA, 2nd edition, 2003.
- [150] H. K. Hartline. The response of single optic nerve fibers of the vertebrate eye to illumination of the retina. *American Journal of Physiology*, 121:400–415, 1938.
- [151] B. Hassibi, D. G. Stork, and G. J. Wolff. Optimal brain surgeon and general network pruning. In *IEEE International Conference on Neural Networks*, volume 1, pages 293–299, 1993. doi:10.1109/ICNN.1993.298572.
- [152] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. arXiv: 1512.03385, <https://arxiv.org/abs/1512.03385>, 2015.
- [153] J. L. Heiberg. *Euclid’s Elements of Geometry*. Teubner, 1883. <http://farside.ph.utexas.edu/Books/Euclid/Elements.pdf>.
- [154] J. Herbrand. Sur la non-contradiction de l’arithmétique. *Journal für die reine und angewandte Mathematik*, 166:1–8, 1932.

- [155] S. Herculano-Houzel. Scaling of brain metabolism with a fixed energy budget per neuron: implications for neuronal activity, plasticity and evolution. *Public Library of Science One*, 6(3):e17514, 2011.
- [156] L. R. Herrmann. Laplacian-isoparametric grid generation scheme. *Journal of the Engineering Mechanics Division*, 102(5):749–907, 1976.
- [157] J. F. W. Herschel. On a remarkable application of Cotes’s theorem. *Philosophical Transactions*, 103(1):8, 1813. doi:10.1098/rstl.1813.0005.
- [158] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.
- [159] T. K. Ho. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):832–844, 1998.
- [160] K. P. Hoffmann and J. Stone. Conduction velocity of afferents to cat visual cortex: a correlation with cortical receptive field properties. *Brain Research*, 32:460–466, 1971.
- [161] C. C. Holt. Forecasting trends and seasonal by exponentially weighted averages. In *Office of Naval Research Memorandum 52*, 1957. Reprinted in C. C. Holt, 2004. “Forecasting trends and seasonal by exponentially weighted averages.” *International Journal of Forecasting*, 20(1):5–10.
- [162] E. Hovers. Chapter 5 - invention, reinvention and innovation: The makings of Oldowan lithic technology. In S. Elias, editor, *Origins of Human Innovation and Creativity*, volume 16 of *Developments in Quaternary Sciences*, pages 51–68. Elsevier, 2012. doi:10.1016/B978-0-444-53821-5.00005-1.
- [163] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger. Deep networks with stochastic depth. In B. Leibe, J. Matas, N. Sebe, and M. Welling, editors, *Computer Vision – ECCV 2016*, pages 646–661, Cham, 2016. Springer International Publishing.
- [164] David H. Hubel and Torsten N. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat’s visual cortex. *Journal of Physiology*, 160:106–154, 1962.
- [165] P. J. Huber. Robust estimation of a location parameter. *The Annals of Mathematical Statistics*, 35(1):73–101, 1964. doi:10.1214/aoms/1177703732.

- [166] F. Hyder, D. Rothman, and M. Bennett. Cortical energy demand of signaling and nonsignaling components in brain are conserved across mammalian species and activity levels. *Proceedings of the National Academy of Sciences of the USA*, 110(9):3549–3554, February 2013. doi:10.1073/pnas.1214912110.
- [167] H. Iba and N. Noman. *Deep Neural Evolution: Deep Learning with Evolutionary Computation*. Springer Singapore, 2020.
- [168] AI Impacts. Brain performance in flops). <https://aiimpacts.org/brain-performance-in-flops/>, 2020. Online, accessed 17-June-2020.
- [169] AI Impacts. Neuron firing rates in humans. <https://aiimpacts.org/rate-of-neuron-firing/>, 2020. Online, accessed 17-June-2020.
- [170] Google Inc. Tensorflow. <https://www.tensorflow.org>, November 9, 2015. Online, accessed 11-January-2020.
- [171] J. P. A. Ioannidis. Why most published research findings are false. *PLOS Medicine*, 2(8), 08 2005. doi:10.1371/journal.pmed.0020124.
- [172] J. P. A. Ioannidis. Why most clinical research is not useful. *PLOS Medicine*, 13(6):1–10, 06 2016. doi:10.1371/journal.pmed.1002049.
- [173] J. P. A. Ioannidis and T. A. Trikalinos. Early extreme contradictory estimates may appear in published research: the proteus phenomenon in molecular genetics research and randomized trials. *Journal of Clinical Epidemiology*, 58(6):543–549, 2005.
- [174] S. Ioffe and C. Szegedy. Batch normalization: accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456. PMLR, 2015.
- [175] K. N. Jayatilleke. *Early Buddhist Theory of Knowledge*. Allen and Unwin, London, 1963.
- [176] Thomas Jech. *Set Theory*. Springer, 3rd edition, 2006. Online at <https://b-ok.org/book/1053554/00ed84>.
- [177] L. K. John, G. Loewenstein, and D. Prelec. Measuring the prevalence of questionable research practices with incentives for truth telling. *Psychological science*, 23(5):524–532, 2012.
- [178] J. S. Judd. Learning in networks is hard. In *Proceedings of the First International Conference on Neural Networks*, pages 685–692, San Diego, California, June 1987. IEEE.
- [179] N. Kalchbrenner, E. Grefenstette, and P. Blunsom. A convolutional neural network for modelling sentences. In *Proceedings of the 52nd Annual*

- Meeting of the Association for Computational Linguistics*, volume 1, pages 655–665, Baltimore, Maryland, June 2014. Association for Computational Linguistics. <https://www.aclweb.org/anthology/P14-1062>. doi:10.3115/v1/P14-1062.
- [180] R. E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82:35–45, 1960.
 - [181] H. Kelley. Gradient theory of optimal flight paths. *Ars Journal*, 30(10):947–954, 1960.
 - [182] M. G. Kendall and S. Alan. *The advanced theory of statistics. Vols. II and III*. Hafner, 1961.
 - [183] B. Kernighan and D. Ritchie. *The C Programming Language*. Prentice Hall, 1st edition, 1978.
 - [184] S. S. Kety. The general metabolism of the brain in vivo. In D. Richter, editor, *Metabolism of the nervous system*, pages 221–237. Pergamon, London, 1957.
 - [185] J. Kiefer and J. Wolfowitz. Stochastic estimation of the maximum of a regression function. *Annals of Mathematical Statistics*, 23(3):462–466, September 1952. doi:10.1214/aoms/1177729392.
 - [186] M. Kimura. Evolutionary rate at the molecular level. *Nature*, 217(5129):624–626, February 1968. doi:10.1038/217624a0.
 - [187] D. P. Kingma and J. Ba. Adam: a method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
 - [188] S. C. Kleene. Lambda-definability and recursiveness. *Duke Mathematical Journal*, 2:340–353, 1936.
 - [189] S. C. Kleene. Representation of events in nerve nets and finite automata. In C. E. Shannon and J. McCarthy, editors, *Automata Studies*, pages 3–42. Princeton University Press, 1951. https://www.rand.org/content/dam/rand/pubs/research_memoranda/2008/RM704.pdf.
 - [190] D. E. Knuth. *The Art of Computer Programming*, volume 4. Addison–Wesley, 2013.
 - [191] A. N. Kolmogorov. *Foundations of the Theory of Probability*. Chelsea Publishing Company, New York, USA, 1950. Initially published in 1933.
 - [192] Andrey N. Kolmogorov. On the representation of continuous functions of several variables by superpositions of continuous functions of a smaller number of variables. *Proceedings of the USSR Academy of Sciences*, 108:179–182, 1956. English translation: American Mathematical Society Translations, 17 (1961), pp. 369–373.

- [193] A. C. Kors. *Encyclopedia of the Enlightenment*. Oxford University Press, 2003.
- [194] E. Koutsoupias and C. Papadimitriou. Worst-case equilibria. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 404–413. Springer, 1999.
- [195] K. Kowsari, M. Heidarysafa, D. E. Brown, K. J. Meimandi, and L. E. Barnes. RMDL: random multimodel deep learning for classification. In *Proceedings of the 2nd International Conference on Information System and Data Mining*, pages 19–28, 2018.
- [196] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. doi:10.1145/3065386.
- [197] A. Krogh and P. Sollich. Learning with ensembles: how over-fitting can be useful. In *Advances in Neural Information Processing Systems 8: Proceedings of the 1995 Conference*, volume 8, page 190, 1996.
- [198] A. Krogh and J. Vedelsby. Neural network ensembles, cross validation, and active learning. In *Advances in Neural Information Processing Systems*, pages 231–238. MIT Press, 1995.
- [199] S. Kullback and R. A. Liebler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, March 1951. doi:10.1214/aoms/1177729694.
- [200] L. I. Kuncheva. *Combining Pattern Classifiers: Methods and Algorithms*. Wiley Publishing, 2nd edition, 2014.
- [201] L. I. Kuncheva and C. J. Whitaker. Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine learning*, 51(2):181–207, 2003. doi:10.1023/A:1022859003006.
- [202] N. Kurani and E. W. Twitter. How does the quality of the U.S. health system compare to other countries? Peterson-KFF Health System Tracker, <https://www.healthsystemtracker.org/chart-collection/quality-u-s-healthcare-system-compare-countries/>, September 30, 2021. Online, accessed 11-January-2022.
- [203] D. König. Über konvexe körper. *Mathematische Zeitschrift*, 14(1):208–210, 1922. doi:10.1007/bf01215899.
- [204] D. B. Lahiri. A method of sample selection providing unbiased ratio estimates. *Bulletin of the International Statistical Institute*, 33(2):133–140, 1951.

- [205] Lumen Learning. The cerebrum. <https://courses.lumenlearning.com/boundless-ap/chapter/the-cerebrum/>, 2020. Online, accessed 3-February-2020.
- [206] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. Technical report, AT&T Bell Laboratories, 1989.
- [207] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. <https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.32.9552>.
- [208] Y. LeCun, C. Cortes, and C. J. C. Burges. The mnist database. <http://yann.lecun.com/exdb/mnist/>, 1999. Online, accessed 28-March-2021.
- [209] Adrien-Marie Legendre. *Nouvelles méthodes pour la détermination des orbites des comètes [New Methods for the Determination of the Orbits of Comets]*. F. Didot, Paris, 1805.
- [210] E. L. Lehmann and G. Casella. *Theory of Point Estimation*, volume 31. Springer, 1998.
- [211] Gottfried W. Leibniz. Methodi tangentium inversae exempla, November 11, 1673. English translation in *The Early Mathematical Manuscripts of Leibniz*. Open Court Publishing, p. 93, 1920.
- [212] Gottfried W. Leibniz. Nouveaux essais sur l’entendement humain [new essays on human understanding], 1704. English translation by Peter Remnant and Jonathan Bennett. Cambridge University Press, New York, 1996.
- [213] C. J. Lepre, H. Roche, D. V. Kent, S. Harmand, R. L. Quinn, J.-P. Brugal, P.-J. Texier, A. Lenoble, and C. S. Feibel. An earlier origin for the Acheulian. *Nature*, 477(7362):82–85, 2011.
- [214] J. Leskovec, A. Rajaraman, and J. D. Ullman. *Mining of Massive Datasets*. Cambridge University Press, 3rd edition, 2020.
- [215] W. B. Levy and R. A. Baxter. Energy-efficient neuronal computation via quantal synaptic failures. *The Journal of neuroscience: the official journal of the Society for Neuroscience*, 22(11):4746–4755, June 2002.
- [216] R. Li, A. Montpetit, M. Rousseau, S. Y. M. Wu, C. M. T. Greenwood, T. D. Spector, M. Pollak, C. Polychronakos, and J. B. Richards. Somatic point mutations occurring early in development: a monozygotic twin study. *Journal of medical genetics*, 51(1):28–34, January 2014. doi: 10.1136/jmedgenet-2013-101712.

- [217] Carolus Linnaeus. *Philosophia botanica*, 1751.
- [218] S. Linnainmaa. The representation of the cumulative rounding error of an algorithm as a taylor expansion of the local rounding errors. Master's thesis, University of Helsinki, 1970.
- [219] K. J. Locey and J. T. Lennon. Scaling laws predict global microbial diversity. *Proceedings of the National Academy of Sciences*, 113(21):5970–5975, 2016. URL: <https://www.pnas.org/content/113/21/5970>, arXiv:<https://www.pnas.org/content/113/21/5970.full.pdf>, doi:10.1073/pnas.1521291113.
- [220] John Locke. *An Essay Concerning Human Understanding*. Tho. Baffet, 1689.
- [221] F. Lopez-Munoz, J. Boya, and C. Alamo. Neuron theory, the cornerstone of neuroscience, on the centenary of the nobel prize award to santiago ramon y cajal. *Brain research bulletin*, 70(4-6):391–405, October 2006.
- [222] G. Lorentz. Metric entropy, widths, and superpositions of functions. *American Mathematical Monthly*, 69:469–485, 1962. doi:10.1080/00029890.1962.11989915.
- [223] Marco M. Bertamini and M. R. Munafò. Bite-size science and its undesired side effects. *Perspectives on Psychological Science*, 7(1):67–71, 2012.
- [224] M. R. Macleod, S. Michie, I. Roberts, U. Dirnagl, I. Chalmers, J. P. A. Ioannidis, R. A-S. Salman, A-W. Chan, and P.I Glasziou. Biomedical research: increasing value, reducing waste. *The Lancet*, 383(9912):101–104, 2014. URL: <https://www.sciencedirect.com/science/article/pii/S0140673613623296>, doi:[https://doi.org/10.1016/S0140-6736\(13\)62329-6](https://doi.org/10.1016/S0140-6736(13)62329-6).
- [225] Quanta Magazine. Theorists debate how ‘neutral’ evolution really is. <https://www.quantamagazine.org/neutral-theory-of-evolution-challenged-by-evidence-for-dna-selection-20181108/>, November 8, 2018. Online, accessed 5-August-2020.
- [226] L. M. Martinez and J.-M. Alonso. Complex receptive fields in primary visual cortex. *Neuroscientist*, 9:317–331, 2003. doi:10.1177/1073858403252732.
- [227] E. McCarthy. A call to prosecute drug company fraud as organized crime. *Syracuse Law Review*, 69:439–489, 2019.
- [228] R. McKeon. *The Basic Works of Aristotle*. Random House, New York, 1941.

- [229] S. M. McKinney, M. Sieniek, V. Godbole, J. Godwin, N. Antropova, H. Ashraffian, T. Back, M. Chesus, G. S. Corrado, A. Darzi and M. Etemadi, F. Garcia-Vicente, F. J. Gilbert, M. Halling-Brown, D. Hassabis, S. Jansen, A. Karthikesalingam, C. J. Kelly, D. King, J. R. Ledsam, D. Melnick, H. Mostofi, L. Peng, J. J. Reicher, B. Romera-Paredes, R. Sidebottom, M. Suleyman, D. Tse, K. C. Young, J. De Fauw, and S. Shetty. International evaluation of an ai system for breast cancer screening. *Nature*, 577(7788):89–94, 2020.
- [230] C. Mead. Neuromorphic electronic systems. *Proceedings of the IEEE*, 78(10):1629–1636, 1990.
- [231] P. Melville and R. J. Mooney. Creating diversity in ensembles using artificial data. *Information Fusion*, 6(1):99 – 111, 2005. Diversity in Multiple Classifier Systems. doi:10.1016/j.inffus.2004.04.001.
- [232] A. De Menard. What’s wrong with social science and how to fix it: reflections after reading 2578 papers. <https://fantasticanachronism.com/2020/09/11/whats-wrong-with-social-science-and-how-to-fix-it/>, September 11, 2020. Online, accessed 21-December-2021.
- [233] M. R. Mickey. Some finite population unbiased ratio and regression estimators. *Journal of the American Statistical Association*, 54(287):594–612, 1959.
- [234] H. Midzuno. On the sampling system with probability proportionate to sum of sizes. *Annals of Institute of Statistical Mathematics*, 3(2):99–108, 1952.
- [235] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *Computing Research Repository (CoRR)*, abs/1301.3:1–12, January 2013.
- [236] M. Minsky and S. Papert. *Perceptrons: an introduction to computational geometry*. M.I.T. Press, Cambridge, Massachusetts, 1969.
- [237] M. Mitchell. *An Introduction to Genetic Algorithms*. Bradford Books, 1998.
- [238] Tom Mitchell. *Machine Learning*. McGraw Hill, 1997.
- [239] S. Monti, P. Tamayo, J. Mesirov, and T. Golub. Consensus clustering: A resampling-based method for class discovery and visualization of gene expression microarray data. *Machine Learning*, 52(1–2):91–118, July 2003.
- [240] University of Montreal Montreal Institute for Learning Algorithms. Theano. <http://www.deeplearning.net/software/theano/>, 2007. Online, accessed 11-January-2020.

- [241] C. Mora, D. P. Tittensor, S. Adl, A. G. B. Simpson, and B. Worm. How many species are there on earth and in the ocean? *PLOS Biology*, 9(8):1–8, August 2011. <https://doi.org/10.1371/journal.pbio.1001127>. doi:10.1371/journal.pbio.1001127.
- [242] T. J. H. Morgan, N. T. Uomini, L. E. Rendell, L. Chouinard-Thuly, S. E. Street, H. M. Lewis, C. P. Cross, C. Evans, R. Kearney, I. de la Torre, A. Whiten, and K. N. Laland. Experimental evidence for the co-evolution of hominin tool-making teaching and language. *Nature Communications*, 6(6029), January 2015. doi:10.1038/ncomms7029.
- [243] O. Morgenstern and J. von Neumann. *Theory of Games and Economic Behavior*. Princeton University Press, 1944.
- [244] M. M. Murray and M. T. Wallace. *The Neural Bases of Multisensory Processes*. CRC Press, 2011.
- [245] M. N. Murthy and N. S. Nanjamma. Almost unbiased ratio estimates based on interpenetrating sub-sample estimates. *Sankhyā: The Indian Journal of Statistics (1933-1960)*, 21(3/4):381–392, 1959.
- [246] Vladimir Nabokov. *Pale Fire*. G. P. Putnam’s Sons, 1962.
- [247] N. S. Nanjamma, M. N. Murthy, and V. K. Sethi. Some sampling systems providing unbiased ratio estimators. *Sankhyā: The Indian Journal of Statistics (1933-1960)*, 21(3/4):299–314, 1959.
- [248] National Institute on Drug Abuse. Opioid overdose crisis.
- [249] nature.com. Cell energy and cell functions. <https://www.nature.com/scitable/topicpage/cell-energy-and-cell-functions-14024533/>, 2014. Online, accessed 22-June-2020.
- [250] M. Nei and W.-H. Li. Mathematical model for studying genetic variation in terms of restriction endonucleases. *Proceedings of the National Academy of Sciences*, 76(10):5269–5273, 1979.
- [251] Isaac Newton. Letter to Henry Oldenburg. The Newton Project, University of Oxford, 1676.
- [252] Nils J. Nilsson. *Learning Machines: Foundations of Trainable Pattern-Classifying Systems*. McGraw-Hill, New York, 1965.
- [253] J. E. Niven. Neuronal energy consumption: biophysics, efficiency and evolution. *Current Opinion in Neurobiology*, 41:129–135, 2016. doi:10.1016/j.conb.2016.09.004.
- [254] American Academy of Actuaries. Prescription drug spending in the u.s. health care system, March 2018. Online, accessed 14-January-2022.

- [255] Department of Justice. The False Claims Act. <https://www.justice.gov/civil/false-claims-act>, January 14, 2021. Online, accessed 8-January-2022.
- [256] B. A. Olshausen and D. J. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6883):607–609, 1996. doi:10.1038/381607a0.
- [257] Biology Online. Branches of biology. <https://www.biologyonline.com/dictionary/branches-of-biology>, 2020. Online, accessed 4-June-2020.
- [258] D. W. Opitz and J. W. Shavlik. Generating accurate and diverse members of a neural-network ensemble. In *Proceedings of the 8th International Conference on Neural Information Processing Systems*, number 7 in NIPS’95, page 535–541, Cambridge, MA, USA, 1995. MIT Press.
- [259] J. Ortega-Hernandez. Our 500 million-year-old nervous system fossil shines a light on animal evolution. The Conversation, US, <https://theconversation.com/our-500-million-year-old-nervous-system-fossil-shines-a-light-on-animal-evolution-55460>, 2016. Online, accessed 3-June-2020.
- [260] C. Papadimitriou. Algorithms, games, and the internet. In *Proceedings of the thirty-third annual ACM symposium on theory of computing (STOC)*, pages 749–753, 2001.
- [261] A. Papoulis. *The Fourier Integral and Its Applications*. McGraw-Hill, New York, 1962.
- [262] A. Papoulis and S. U. Pillai. *Probability, Random Variables, and Stochastic Processes*. McGraw-Hill, 2002.
- [263] F. Song and S. Parekh, L. Hooper, Y. K. Loke, J. Ryder, A. J. Sutton, C. Hing, C. S. Kwok, C. Pang, and I. Harvey. Dissemination and publication of research findings: an updated review of related biases. *Health Technology Assessment*, 14(8):1–220, 2010. doi:10.3310/hta14080.
- [264] A. Paszke, S. Gross, S. Chintala, and G. Chanan. Pytorch. <https://www.pytorch.org>, October 2016. Online, accessed 11-January-2020.
- [265] G. Peano. *Arithmetices Principia: Nova Methodo Exposita*. Fratres Bocca, 1889.
- [266] B. K. D. Pearce, A. S. Tupper, R. E. Pudritz, and P. G. Higgs. Constraining the time interval for the origin of life on earth. *Astrobiology*, 18(3):343–364, March 1, 2018. doi:10.1089/ast.2017.1674.

- [267] K. Pearson. On a form of spurious correlation which may arise when indices are used in the measurement of organs. In *Proceedings of the Royal Society*, volume 60, pages 489–502, London, 1897.
- [268] J. A. Perge, J. E. Niven, E. Mugnaini, V. Balasubramanian, and P. Sterling. Why do axons differ in caliber? *Journal of Neuroscience*, 32(2):626–638, 2012. URL: <https://www.jneurosci.org/content/32/2/626>, arXiv:<https://www.jneurosci.org/content/32/2/626.full.pdf>, doi:10.1523/JNEUROSCI.4254-11.2012.
- [269] H. Pham and Q. V. Le. Autodropout: learning dropout patterns to regularize deep networks. <https://arxiv.org/abs/2101.01761>, 2021.
- [270] PhysiologyWeb.com. Neuronal action potential – refractory periods. http://www.physiologyweb.com/lecture_notes/neuronal_action_potential/neuronal_action_potential_refractory_periods.html, 2012. Online, accessed 17-June-2020.
- [271] Leonardo Pisano. *Liber Abaci*, 1202. English translation by L. Sigler. *Fibonacci’s Liber Abaci: a translation into modern English of Leonardo Pisano’s book of calculation*. Springer Science & Business Media, 2003.
- [272] J. A. Pittman. Handwriting recognition: tablet PC text input. *IEEE Computer*, 40(9):49–54, September 2007.
- [273] Plato. Cratylus. <http://classics.mit.edu/Plato/cratylus.html>, c. 360 BCE. Online, accessed 6-March-2020.
- [274] B. T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.
- [275] J. G. Proakis and D. G. Manolakis. *Digital Signal Processing: Principles, Algorithms, and Applications*. Prentice-Hall, Inc., 3rd edition, 1996.
- [276] D. Raj. Ratio estimation in sampling with equal and unequal probabilities. *Journal of the Indian Society of Agricultural Statistics*, 6(2):127–138, 1954.
- [277] U. Ramer. An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing*, 1(3):244–256, 1972.
- [278] A. S. Relman. The new medical-industrial complex. *New England journal of medicine*, 303(17):963–970, 1980.
- [279] A. S. Relman and M. Angell. How the drug industry distorts medicine and politics. america’s other drug problem. *The New Republic*, 227(25):27–41, 2002.

- [280] F. Ren, X. Ding, M. Zheng, M. Korzinkin, X. Cai, W. Zhu, A. Mantsyzov, A. Aliper, V. Aladinskiy, Z. Cao, S. Kong, X. Long, B. H. M. Liu, Y. Liu, V. Naumov, A. Shneyderman, I. V. Ozerov, J. Wang, F. W. Pun, A. Aspuru-Guzik, M. Levitt, and A. Zhavoronkov. Alphafold accelerates artificial intelligence powered drug discovery: efficient discovery of a novel cyclin-dependent kinase 20 (cdk20) small molecule inhibitor. *arXiv preprint arXiv:2201.09647*, 2022.
- [281] B. B. Rieger. On distributed representation in word semantics. Technical report, Forschungsbericht TR-91-012, International Computer Science Institute (ICSI, 1991.
- [282] J. Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978. doi:10.1016/0005-1098(78)90005-5.
- [283] D. M. Ritchie. The development of the C language. *ACM Sigplan Notices*, 28(3):201–208, 1993.
- [284] H. Robbins and S. Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, pages 400–407, 1951.
- [285] L. Rokach. Ensemble-based classifiers. *Artificial Intelligence Review*, 33(1-2):1–39, 2010.
- [286] D. F. Rolfe and G. C. Brown. Cellular energy utilization and molecular origin of standard metabolic rate in mammals. *Physiological Reviews*, 77:731–758, 1997.
- [287] F. Rosenblatt. The perceptron – a perceiving and recognizing automaton. Technical Report 85-460-1, Cornell Aeronautical Laboratory, 1957.
- [288] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. Technical Report ICS-8506, University of California, San Diego, La Jolla Institute for Cognitive Science, 1985.
- [289] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing, volume 1*, pages 318–362. MIT Press, 1986.
- [290] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.
- [291] David Rumelhart and James McClelland. *Parallel Distributed Processing, Volume 1*. MIT Press, 1986.
- [292] David Rumelhart and James McClelland. *Parallel Distributed Processing, Volume 2*. MIT Press, 1986.

- [293] C. Runge. Über empirische funktionen und die interpolation zwischen äquidistanten ordinaten. *Zeitschrift für Mathematik und Physik*, 46:224–243, 1901.
- [294] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.
- [295] A. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, 1959. CiteSeerX:10.1.1.368.2254, <https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.368.2254>. doi:10.1147/rd.33.0210.
- [296] A. Savitzky and M. J. E. Golay. Smoothing and differentiation of data by simplified least squares procedures. *Analytical Chemistry*, 36(8):1627–1639, 1964.
- [297] R. E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.
- [298] J. Schmidhuber. Learning factorial codes by predictability minimization. Technical Report CU-CS-565-91, Department of Computer Science, University of Colorado at Boulder, December 1991.
- [299] J. Schmidhuber. A possibility for implementing curiosity and boredom in model-building neural controllers. In J. A. Meyer and S. W. Wilson, editors, *Proceedings of the International Conference on Simulation of Adaptive Behavior: From Animals to Animats*, pages 222–227. MIT Press/Bradford Books, 1991.
- [300] J. Schmidhuber. Deep learning. http://www.scholarpedia.org/article/Deep_Learning, 2015. Online, accessed 4-May-2020.
- [301] J. Schmidhuber. Deep learning in neural networks: an overview. *Neural Networks*, 61:85–117, 2015. Available at: "<http://people.idsia.ch/~juergen/DeepLearning80ct2014.pdf>".
- [302] J. Schmidhuber. History of computer vision contests won by deep convolutional nets on GPU. <http://people.idsia.ch/~juergen/computer-vision-contests-won-by-gpu-cnns.html>, 2017. Online, accessed 9-February-2020.
- [303] J. Schmidhuber. Critique of Honda prize for Dr. Hinton. <http://people.idsia.ch/~juergen/critique-honda-prize-hinton.html>, 2020. Online, accessed 11-May-2020.
- [304] I. J. Schoenberg. Contributions to the problem of approximation of equidistant data by analytic functions. *Quarterly of Applied Mathematics*, 4(1):45–99, April 1946. URL: <https://www.jstor.org/stable/43633538>.

- [305] S. Schor and I. Karten. Statistical evaluation of medical journal manuscripts. *The Journal of the American Medical Association*, 195(13):1123–1128, March 1966. doi:10.1001/jama.1966.03100130097026.
- [306] D. R. Schrider and A. D. Kern. Soft sweeps are the dominant mode of adaptation in the human genome. *Molecular Biology and Evolution*, 34(8):1863–1877, May 2017. <https://academic.oup.com/mbe/article-pdf/34/8/1863/24367882/msx154.pdf>. doi:10.1093/molbev/msx154.
- [307] A. R. Sen. Present status of probability sampling and its use in estimation of characteristics. *Econometrica*, 20:103, 1952.
- [308] B. Sengupta and M. B. Stemmler. Power consumption during neuronal computation. *Proceedings of the IEEE*, 102(5):738–750, 2014.
- [309] M. Serra-Garcia and U. Gneezy. Nonreplicable publications are cited more than replicable ones. *Science Advances*, 7(21), 2021. URL: <https://advances.sciencemag.org/content/7/21/eabd1705>, doi:10.1126/sciadv.abd1705.
- [310] R. Service. 2021 breakthrough of the year. protein structures for all. *Science*, <https://www.science.org/content/article/breakthrough-2021>, December 16, 2021. Online, accessed 28-January-2022.
- [311] L. S. Shapley. Notes on the n-person game — ii: The value of an n-person game. RAND Corporation, Santa Monica, CA, 1951. https://www.rand.org/pubs/research_memoranda/RM0670.html.
- [312] P. Sheeler and D. E. Bianchi. *Cell and Molecular Biology*. Wiley, United Kingdom, 1987.
- [313] G. M. Shepherd. *Foundations of the Neuron Doctrine*. Oxford University Press, 1991.
- [314] C. S. Sherrington. Observations on the scratch-reflex in the spinal dog. *Journal of Physiology*, 34(1–2):1–50, 1906. doi:10.1113/jphysiol.1906.sp001139.
- [315] K. Shin and J. Hammond. *Fundamentals of Signal Processing for Sound and Vibration Engineers*. John Wiley & Sons, 2008.
- [316] M. Simkin and V. P. Roychowdhury. Read before you cite! *arXiv preprint cond-mat/0212043*, 2002.
- [317] J. P. Simmons, L. D. Nelson, and U. Simonsohn. False-positive psychology: Undisclosed flexibility in data collection and analysis allows presenting anything as significant. *Psychological science*, 22(11):1359–1366, 2011.

- [318] P. Singh. The so-called fibonacci numbers in ancient and medieval india. *Historia Mathematica*, 12(3):229–244, 1985. doi:10.1016/0315-0860(85)90021-7.
- [319] D. R. So, C. Liang, and Q. V. Le. The evolved transformer. *CoRR*, abs/1901.11117, 2019. URL: <http://arxiv.org/abs/1901.11117>.
- [320] L. Sokoloff. The metabolism of the central nervous system in vivo. In J. Field, H. W. Magoun, and V. E. Hall, editors, *Handbook of Physiology, Section I, Neurophysiology, vol. 3*, pages 1843–1864. American Physiological Society, Washington D. C., 1960.
- [321] R. M. Solovay. Real-valued measurable cardinals. In *Axiomatic set theory (Proceedings of the Symposium on Pure Mathematics, Vol. XIII, Part I, University of California, Los Angeles, California, 1967)*, pages 397–428, Providence, R. I., 1971. American Mathematical Society.
- [322] Mayfield Brain & Spine. Anatomy of the brain. <https://mayfieldclinic.com/pe-anatbrain.htm>, 2018. Online, accessed 2-July-2020.
- [323] D. A. Sprecher. On the structure of continuous functions of several variables. *Transactions of the American Mathematical Society*, 115:340–355, 1965.
- [324] R. K. Srivastava, K. Greff, and J. Schmidhuber. Training very deep networks. arXiv: 1505.00387, <https://arxiv.org/abs/1505.00387>, 2015.
- [325] Blausen.com staff. Medical gallery of Blausen medical 2014. https://en.wikiversity.org/wiki/WikiJournal_of_Medicine/Medical_gallery_of_Blausen_Medical_2014, 2014. Online, accessed 15-June-2020. doi:10.15347/wjm/2014.010.
- [326] Peter Sterling and Simon Laughlin. *Principles of Neural Design*. MIT Press, 2015. <https://muse.jhu.edu/book/40220>.
- [327] T. D. Sterling. Publication decisions and their possible effects on inferences drawn from tests of significance—or vice versa. *Journal of the American statistical association*, 54(285):30–34, 1959.
- [328] D. G. Stork, G. Wolff, and E. Levine. Neural network lipreading system for improved speech recognition. In *Proceedings of the International Joint Conference on Neural Networks*, volume 2, pages 289–295, Baltimore, MD, 1992. IEEE. doi:10.1109/IJCNN.1992.226994.
- [329] A. Strehl and J. Ghosh. Cluster ensembles—a knowledge reuse framework for combining multiple partitions. *The Journal of Machine Learning Research*, 3:583–617, 2002.

- [330] B. Stroustrup. *The C++ Programming Language*. Addison Wesley, 1985.
- [331] E. Strubell, A. Ganesh, and A. McCallum. Energy and policy considerations for deep learning in NLP. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3645–3650, Florence, Italy, July 2019. Association for Computational Linguistics. URL: <https://www.aclweb.org/anthology/P19-1355>, doi:10.18653/v1/P19-1355.
- [332] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune. Deep neuroevolution: genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv preprint arXiv:1712.06567*, 2017.
- [333] J. Sylvester and N. V. Chawla. Evolutionary ensembles: combining learning agents using genetic algorithms. In *AAAI Workshop on Multi-agent Learning*, pages 46–51, 2005.
- [334] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1701–1708, 2013.
- [335] A. Tarski. Der wahrheitsbegriff in den formalisierten sprachen. *Studia Philosophica*, 1:261–405, 1936.
- [336] The AlphaFold team. Alphafold: a solution to a 50-year-old grand challenge in biology. DeepMind, <https://deepmind.com/blog/article/alphafold-a-solution-to-a-50-year-old-grand-challenge-in-biology>, November 30, 2020. Online, accessed 28-January-2022.
- [337] L. Tedersoo, S. Sánchez-Ramírez, U. Kõljalg, M. Bahram, M. Döring, D. Schigel, T. May, M. Ryberg, and K. Abarenkov. Ti high-level classification of the fungi and a tool for evolutionary ecological analyses. *Fungal Diversity*, 90(1):135–159, May 2018. <https://doi.org/10.1007/s13225-018-0401-0>. doi:10.1007/s13225-018-0401-0.
- [338] A. N. Tikhonov. On the stability of inverse problems. *Doklady Akademii Nauk SSSR*, 39(5):195–198, 1943.
- [339] A. N. Tikhonov and V. Y. Arsenin. *Solution of Ill-posed Problems*. Winston & Sons, Washington, 1977.
- [340] R. Tikkanen and M. K. Abrams. U.s. health care from a global perspective, 2019: higher spending, worse outcomes? <https://www.commonwealthfund.org/publications/issue-briefs/2020/jan/us-health-care-global-perspective-2019>, January 30, 2020. Online, accessed 11-January-2022.

- [341] J. Tompson, R. Goroshin, A. Jain, Y. LeCun, and C. Bregler. Efficient object localization using convolutional networks. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 648–656, 2015. doi:10.1109/CVPR.2015.7298664.
- [342] J. W. Tukey. *Exploratory Data Analysis*. Addison-Wesley Publishing Company, Reading, MA, 1977.
- [343] K. Tunyasuvunakool, J. Adler, Z. Wu, T. Green, M. Zielinski, A. Židek, A. Bridgland, A. Cowie, C. Meyer, A. Laydon, S. Velankar, G. J. Kleywegt, A. Bateman, R. Evans, A. Pritzel, M. Figurnov, O. Ronneberger, R. Bates, S. A. A. Kohl, A. Potapenko, A. J. Ballard, B. Romera-Paredes, S. Nikolov, R. Jain, E. Clancy, D. Reiman, S. Petersen, A. W. Senior, K. Kavukcuoglu, E. Birney, P. Kohli, J. Jumper, and D. Hassabis. Highly accurate protein structure prediction for the human proteome. *Nature*, 596(7873):590–596, 2021.
- [344] Alan M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42(2):230–265, 1936.
- [345] Alan M. Turing. Computability and λ -definability. *The Journal of Symbolic Logic*, 2(4):153–163, 1937.
- [346] Alan M. Turing. Intelligent machinery. Technical report, National Physical Laboratory, 1948. URL: <https://www.npl.co.uk/getattachment/about-us/History/Famous-faces/Alan-Turing/80916595-Intelligent-Machinery.pdf>.
- [347] S. Ulam. Zur masstheorie in der allgemeinen mengenlehre. *Fundamenta Mathematicae*, 16(1):140–150, 1930.
- [348] U.S. Department of Health and Human Services. What is the u.s. opioid epidemic?
- [349] G. M. P. van Kempen and L. J. van Vliet. Mean and variance of ratio estimators used in fluorescence ratio imaging. *Cytometry*, 39(4):300–305, 2000.
- [350] R. C. Vergara, S. J.-R. A. Luarte, C. Moënné-Loccoz, R. Fuentes, A. Couve, and P. E. Maldonado. The energy homeostasis principle: neuronal energy regulation drives local network dynamics generating behavior. *Frontiers in Computational Neuroscience*, 13:49, July 2019. doi:10.3389/fncom.2019.00049.
- [351] P.-F. Verhulst. Notice sue la loi que la population suit dans son accroissement. *Correspondance mathématique et Physique, publiée par A. Quetelet*, 10:113–120, 1838.

- [352] P.-F. Verhulst. Recherches mathematiques sur la loi d'accroissement de la population. *Nouveaux Memoires de l'Academie Royale des Sciences, des Lettres et des Beaux-Arts de Belgique*, 18:1–38, 1845.
- [353] P. S. Verma and V. K. Agarwal. *Cell Biology, Genetics, Molecular Biology: Evolution and Ecology*. S. Chand & Co Ltd, 2005.
- [354] John von Neumann. Zur theorie der gesellschaftsspiele. *Mathematische Annalen*, 100(1):295–320, 1928. doi:10.1007/BF01448847.
- [355] John von Neumann. On the theory of games of strategy. In A. W. Tucker and R. D. Luce, editors, *Contributions to the Theory of Games. Volume IV*, pages 13–42. Princeton University Press, 1959.
- [356] A. Waibel. Phoneme recognition using time-delay neural networks. In *Meeting of the Institute of Electrical Information and Communication Engineers (IEICE)*, Tokyo, Japan, 1987.
- [357] L. Wan, M. Zeiler, S. Zhang, Y. Le Cun, and R. Fergus. Regularization of neural networks using dropconnect. In *International conference on machine learning*, pages 1058–1066. PMLR, 2013.
- [358] L. Wan, M. Zeiler, S. Zhang, Y. LeCun, and R. Fergus. Regularization of neural networks using dropconnect. In S. Dasgupta and D. McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1058–1066, Atlanta, Georgia, USA, June 2013. PMLR. URL: <http://proceedings.mlr.press/v28/wan13.html>.
- [359] X. Wang, K. Chan, K. Yu, C. Dong, and C. Change Loy. Edvr: Video restoration with enhanced deformable convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, June 2019.
- [360] K. Weierstrass. Über die analytische darstellbarkeit sogenannter willkürlicher functionen einer reellen veränderlichen. *Sitzungsberichte der Königlich Preußischen Akademie der Wissenschaften zu Berlin*, 1885 (II), 1885.
- [361] P. J. Werbos. Applications of advances in nonlinear sensitivity analysis. In *Proceedings of the 10th International Federation for Information Processing (IFIP) Conference*, pages 762–770, New York, 1982.
- [362] Paul J. Werbos. *Beyond regression: New tools for prediction and analysis in the behavioral sciences*. PhD thesis, Harvard University, 1974.
- [363] R. S. Westfall. *The Life of Isaac Newton*. Canto original series. Cambridge University Press, Cambridge, 1994. doi:10.1017/CB09781107050334.

- [364] whistleblowingprotection.org. Whistleblower info. <http://www.whistleblowingprotection.org/>, 2022. Online, accessed 8-January-2022.
- [365] B. Widrow and M. E. Hoff. Adaptive switching circuits. In *IRE WESCON Convention Record*, pages 96–104, 1960.
- [366] Wikipedia. Mathesis universalis. https://en.wikipedia.org/wiki/Mathesis_universalis, May 9, 2018. Online, accessed 6-March-2020.
- [367] Wikipedia. Biodiversity. <https://en.wikipedia.org/wiki/Biodiversity>, August 3, 2020. Online, accessed 5-August-2020.
- [368] Wikipedia. Biology. <https://en.wikipedia.org/wiki/Biology>, May 17, 2020. Online, accessed 4-June-2020.
- [369] Wikipedia. Biology. https://en.wikipedia.org/wiki/Outline_of_biology, May 17, 2020. Online, accessed 4-June-2020.
- [370] Wikipedia. The compendious cook on calculation by completion and balancing. https://en.wikipedia.org/wiki/The_Compendious_Book_on_Calculation_by_Completion_and_Balancing, July 27, 2020. Online, accessed 8-August-2020.
- [371] Wikipedia. Cooperative game theory. https://en.wikipedia.org/wiki/Cooperative_game_theory, June 9, 2020. Online, accessed 30-July-2020.
- [372] Wikipedia. Evolutionary history of life. https://en.wikipedia.org/wiki/Evolutionary_history_of_life, June 2, 2020. Online, accessed 3-June-2020.
- [373] Wikipedia. Game theory. https://en.wikipedia.org/wiki/Game_theory, July 29, 2020. Online, accessed 30-July-2020.
- [374] Wikipedia. Generative adversarial network. https://en.wikipedia.org/wiki/Generative_adversarial_network, April 7, 2020. Online, accessed 7-April-2020.
- [375] Wikipedia. Generative adversarial network. https://en.wikipedia.org/wiki/Nervous_system, April 30, 2020. Online, accessed 3-June-2020.
- [376] Wikipedia. Genus. <https://en.wikipedia.org/wiki/Genus>, July 22, 2020. Online, accessed 5-August-2020.
- [377] Wikipedia. List of plant orders. https://en.wikipedia.org/wiki/List_of_plant_orders, March 15, 2020. Online, accessed 5-August-2020.

- [378] Wikipedia. Myelin. <https://en.wikipedia.org/wiki/Myelin>, June 20, 2020. Online, accessed 21-June-2020.
- [379] Wikipedia. Performance per watt. https://en.wikipedia.org/wiki/Performance_per_watt, June 10, 2020. Online, accessed 17-June-2020.
- [380] Wikipedia. Shapley value. https://en.wikipedia.org/wiki/Shapley_value, June 5, 2020. Online, accessed 30-July-2020.
- [381] Wikipedia. Summit (supercomputer). [https://en.wikipedia.org/wiki/Summit_\(supercomputer\)](https://en.wikipedia.org/wiki/Summit_(supercomputer)), June 1, 2020. Online, accessed 17-June-2020.
- [382] Wikipedia. Center for Open Science. https://en.wikipedia.org/wiki/Center_for_Open_Science, June 27, 2021. Online, accessed 13-September-202q.
- [383] Wikipedia. Center for Open Science. https://en.wikipedia.org/wiki/Hanlon%27s_razor, December 9, 2021. Online, accessed 4-January-2022.
- [384] Wikipedia. False Claims Act. https://en.wikipedia.org/wiki/False_Claims_Act, December 30, 2021. Online, accessed 8-January-2022.
- [385] Wikipedia. Pharmaceutical research and manufacturers of america. https://en.wikipedia.org/wiki/Pharmaceutical_Research_and_Manufacturers_of_America, Novemeber 1, 2021. Online, accessed 17-January-2022.
- [386] Wikipedia. Mnist database. https://en.wikipedia.org/wiki/MNIST_database, February 10, 2022. Online, accessed 22-February-2022.
- [387] Wikipedia. Opioid epidemic. https://en.wikipedia.org/wiki/Opioid_epidemic, January 10, 2022. Online, accessed 10-January-2022.
- [388] Wikipedia. Opioid epidemic in the United States. https://en.wikipedia.org/wiki/Opioid_epidemic_in_the_United_States, January 10, 2022. Online, accessed 10-January-2022.
- [389] A. B. Williams and F. J. Taylor. *Electronic Filter Design Handbook*. McGraw-Hill Education, 4th edition, 2006.
- [390] G. Witkop. Systematizing confidence in open research and evidence (score). <https://www.darpa.mil/program/systematizing-confidence-in-open-research-and-evidence>.
- [391] E. Wolf. *Introduction to the Theory of Coherence and Polarization of Light*. Cambridge University Press, Cambridge, 2007.

- [392] D. Wolpert. The lack of a priori distinctions between learning algorithms. *Neural Computation*, pages 1341–1390, 1996.
- [393] D. H. Wolpert. Stacked generalization. *Neural Networks*, 5:241–259, 1992.
- [394] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.
- [395] D. H. Wolpert and W. G. Macready. Coevolutionary free lunches. *Evolutionary Computation, IEEE Transactions on*, 9(6):721–735, 2005.
- [396] www.etymonline.com. Rationalist. <https://www.etymonline.com/word/rationalist>, 2021. Online, accessed 17-May-2021.
- [397] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017.
- [398] K. Yamaguchi, K. Sakamoto, T. Akabane, and Y. Fujimoto. A neural network for speaker-independent isolated word recognition. In *First International Conference on Spoken Language Processing (ICSLP 90)*, Kobe, Japan, 1990.
- [399] X. Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.
- [400] N. S. Young, J. P. A. Ioannidis, and O. Al-Ubaydli. Why current publication practices may distort science. *PLOS Medicine*, 5(10):e201, 2008.
- [401] P. Zarchan and H. Musoff. *Fundamentals of Kalman Filtering: A Practical Approach*. American Institute of Aeronautics and Astronautics, Incorporated, 4th edition, 2015.
- [402] D.-Q. Zhang. clcNet: Improving the efficiency of convolutional neural network using channel local convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7912–7919, 2018.
- [403] J. Zhu, L. Fang, and P. Ghamisi. Deformable convolutional neural networks for hyperspectral image classification. *IEEE Geoscience and Remote Sensing Letters*, 15(8):1254–1258, 2018.
- [404] I. G. Zurbenko. *The spectral analysis of time series*, volume 2. North-Holland Series in Statistics and Probability, 1986.

Alphabetical Index

Symbols

L_p regularization	166
Δ -system	267, 311
Θ -notation	183
α -divergence	54
χ^2 measure	54
λ -definable function	233, 233
μ -recursive	233
μ -recursive functions	214, 238
μ -recursive functions (or <i>general recursive functions</i>)	236
PPV	315
ψ -notation	45
σ -algebra	151
σ -field	151
f -divergence	54
k -nearest neighbor rule	180
k -order affine transformation	213
k -order recursion	211
k -order recursive transformation	212
n -orthotope	178
(	61, 154
παρισότης	59

A

absolute homogeneity	165
absolute refractory period	74
action potential	77
activation function	19
Adagrad	252, 263
Adam	252, 263
adaptation	38
adenosine triphosphate	. <i>see</i> ATP
adequality	59
adversarial approaches	42
adversarial curiosity	43, 61
adversarial learning	42
adversarial networks	112

affine invariant Riemannian metric
192

Age of Enlightenment	338
Age of Reason	314, 338
aggregating function	14
algebra	96
allele	120
allocortex	87
AlphaFold	341
AlphaFold2	330
ambiguity	141
angle	55
approximate dynamic	
programming	43
approximate equality	59
arctan activation function	20
area coherence	186
Aristotle-Ockham razor	161, 202
arithmetic algebra	97
Arithmetica	97
arrows	7
artificial scarcity	339
association function	189
association tracts	13
ATP	78
auto-encoder	42
autodropout	164, 202
automatic memory management	308

axiom of choice	149, 152
axiom of regularity	149
axon	70, 71
axon terminal	71

B

back-propagation	45, 48, 60, 148
back-propagation theorem	45, 148
bagging	96, 114
batch	224
batch normalization	167

batch training 167
 Bayes error rate 124
 best Lipschitz constant 197
 Bhattacharyya angle 56, 61
 Bhattacharyya coefficient 55
 Bhattacharyya distance ...56, 61
 bias 152
 bias layer 289
 biased estimator 152
 biased transformation 231
 big pharma 324
 big-theta notation 183
 bilaterian animal 83
 binary classification 57
 binary relation 157
 binding 16, 132
 binomial coefficient 113
 biodiversity 119
 biology 65, 83
 bipolar learner 109
 black reaction 83
 blind experiment 318
 boosting 96, 115
 bound index 144
 brainstem 87
 breast cancer prediction 335
 Bregman divergence 54, 61
 Butterworth filter 233

C

C programming language 309
 C++ 309
 Cantor's theorem 150
 cardinal 148
 cardinal arithmetic 149
 cardinal number 148
 cardinal of the continuum ...149
 cardinality 148
 CART 240
 CASP 330
 catastrophic interference 38
 cell theory 65, 68, 83
 celu activation function 20
 Center for Open Science 320
 Center of Open Science 340

cerebellum 87
 cerebral cortex 87
 cerebrum 13, 87
 characteristic 132
 Chebyshev filter 233
 Chebyshev nodes 176, 203
 Chebyshev polynomial 203
 Church-Turing thesis 58, 214,
 233
 classification 51
 classification function 51
 cluster coherence 255
 clustering 42
 CNN 27
 coalitions 112
 cognitive learning 40
 coherence 184, 193
 coherence convergence 248
 coherence of order k 193
 Colgi stain 83
 commissural tracts 13
 comparative neural network .140
 competition 89
 complete cut 91
 complete cut-set 91
 complex conjugate 207
 composite loss 108
 conditional statement 238
 cones 31
 configuration parameters 132
 congruence 143
 congruence modulo n 143
 congruence relation 143
 congruence subscript notation 243
 connections 12
 consistent function 157
 constant function ..233, **236**, 238
 context of the situation 61
 continuity 184
 contract research organizations
 328
 contrast function 53
 convergence 248
 convex closure 178

convex hull **178**, 203
convex hull expansion method **180**
convolution **207**
convolution aging **209**
convolution growth rate **209**
convolution layer **27**
convolution-squeeze-and-
 excitation block
 284
convolutional layer **27**
convolutional neural network . *see*
 CNN
cooperation **96**
cooperative game **112**
cooperative game theory **112**
cooperative learning **107**
cooperative neural networks .. **96**
coordination ratio **141**
corolla **268**
corolla strider **268**
corpus callosum **13**, **87**
correlation **126**
correlation aging **209**
correlation growth rate **209**
countable additivity **151**
countable set **150**
covariance **126**
Cramèr's theorem **123**
critic **116**
Critical Assessment of protein
 Structure Prediction *see* CASP
cross entropy **52**
cross entropy loss **41**, 51
cross validation **155**
cross-connections **12**
cross-correlation **207**
cumulative convolution **209**
cumulative correlation **209**
cut **91**
cut-set **91**
cycle **8**

D

DARPA **320**
data library **310**

data set **157**
data superposition method .. **276**
decision boundary **124**
decision problem **57**
decision tree learning **240**
Dedekind cut **149**
Defense Advanced Research
 Projects Agency .. *see* DARPA
deformable convolution **32**
deformable convolutional network
 34
delta rule **59**
dendrite **70**
dendritic coherence **255**
dendron **70**
depth **11**
depth coherence **186**
detachment segment **127**
deterministic Turing machine . **23**,
 57
diagonalization **150**
differentiation **70**
digraph **7**
diminishing gradient **164**
Dirac delta function **52**
direct full connectivity **18**
direct predecessor **7**
direct successor **7**
directed acyclic graph **8**
directed edges **7**
directed graph **7**
disagreement **129**
dissent **129**, 141
distance **41**, 132
distinction **172**
distributed library **310**
distribution **52**
distributional hypothesis .. **42**, 61
distributional semantics **60**
divergence function **53**
diveristy **132**
diversifying selection **120**
diversity **119**, **134**, 141
diversity algorithms **136**

diversity selection algorithm . **136**
 divide-and-conquer method .. **185**
 divide-and-conquer strategy ... **89**
 double slit experiment **204**
 drift **243**
 dropconnect **164**, 202
 dropout **162**, 202
 dynamic diversity **135**
 dynamic diversity algorithm . **136**
 dynamic inherent diversity .. **135**

E

edges **7**
 effect size **316**
 effective aggregation **127**
 electrostatic force **135**
 element-wise functions **18**
 element-wise superposition
 method **276**
 elemental duality **69**
 elementary neural theory *see* ENT
 elu activation function **20**
 empirical error of a function . **156**
 empirical error of a learning
 algorithm **156**
 empiricism **313**, 338
 empty array **5**
 empty set **5**
 empty tuple **5**
 encoder-decoder network **26**
 ensemble **113**
 ensemble members **114**
 ensemble models **96**
 ENT **72**
 entropy **125**
 epigenetic modification **81**
 epistemology **313**
 epoch **167**
 equilibrium inefficiency **129**
 equivalent indices **6**
 error function **41**
 estimand **152**
 estimate **152**
 estimator **152**
 estimator bias **152**

estimator error **152**
 evolution **66**, 82
 exaptation **120**
 experimenter bias **318**
 explicit split notation **145**
 exponential moving average . **222**,
 233
 Exponential smoothing **233**
 exponential smoothing **222**
 extrapolation **178**
 extrinsic diversity **129**
 extrinsic learning **40**

F

factorization method **256**
 factorized neural coherence .. **256**
 fair strider **273**
 False Claims Act *see* FCA
 Fantastic Anachronism **321**
 FCA **324**, 340
 feed-forward neural network .. **15**,
 94
 Fermat's theorem **59**
 Fermat's theorem for extrema . **59**
 Fibonacci sequence **144**, 211, 232
 filters **27**
 final state **23**
 finite cardinal **149**
 finite difference **183**
 finite different **183**
 finite metric difference **189**
 finite metric difference of order k
 193
 finite recursion **211**, 239
 finite sequence **220**
 first-order affine transformation
 213
 first-order coherence **193**
 first-order finite recursion **232**
 first-order incoherence **193**
 Fisher's linear discriminant ... **59**
 flip-flop **239**
 flow of computation **72**
 flower **268**
 flower core **268**

flower domain 268
 flower kernel 268
 flower map 269
 flower multiplication 273
 Flower net 271
 flower strider 268
 flower structure 269
 forward layers 10
 frame coherence 186
 frequency-dependent selection 120
 frontal lobe 87
 Fujitsu A64FX prototype 73
 fully aligned strider sequence 268
 fully connected layer 18
 fully connected neural network 26
 function composition 238
 function minimization 238
 functional data set 157
 functional distinction 172

G

game 112
 game players 112
 game theory 112
 gamma function 178
 garbage collection 308
 gene expression 66
 gene flow 120
 general recursive functions .. 236,
 238
 generalization 154
 generalization error 155
 generalized function 52
 generalized Hebbian learning .. 42
 generalized symmetrical Bregman
 divergence *see* GSB, 61
 generative adversarial network *see*
 (
 generative adversarial networks 43
 generative heterologous networks
 see GHN
 generative heterology 117
 generator 116
 genetic diversity 119
 genetic drift 66

genetic variability 119
 genetics 120
 geometric algebra 96, 139
 geometric coherence 196
 geometric incoherence 196
 GHN 117
 gigaFLOP 73
 Gilbert–Johnson–Keerthi distance
 algorithm 180, 203
 glia 66
 glial cell 66
 golden ratio 302
 gradient boosting 115
 gradient descent .. 44, 59, 60, 240
 gradient explosion 164
 gradient momentum 249
 graph peripheries 91
 graph theory 7, 91
 grouped convolution 28
 growth 69
 growth rate 183
 GSB 55

H

Hamlet 34
 handwriting personalization . 114
 Hanlon’s razor 323
 hardshrink activation function 20
 hardtanh activation function .. 20
 Hebbian learning 42
 Hellinger distance 55
 heredity 66
 Hessian matrix 165
 heterologous learners 110
 heterologous learning processes
 110
 heteropolar learners 110
 heteropolar learning processes 110
 heteropolar losses 110
 hidden layers 11
 hierarchical synergy 104
 highway network 33
 hinge loss 41
 homeostasis 66, 84
 homologous learners 109

homologous learning processes **109**
 homopolar learners **109**
 homopolar learning processes **109**
 homopolar losses **109**
 Huber loss **41**, 51
 hypercube volume **178**
 hyperrectangle **178**
 hypersphere **178**
 hypersphere volume **178**
 hypothesis **155**
 hypothesis space **58**, **156**
 hysteresis **243**, 244, 263

I

i.i.d. **76**, 122
 IBM Summit supercomputer . **73**,
 84
 ID3 **240**
 identity metric **198**
 identity of indiscernibles **41**
 iff **4**
 ill-posed problem **161**
 implicit cross-connections **13**
 implicit sea notation 144
 implicit split notation 144
 impulse **52**
 incoherence **193**
 incoherence of order k **193**
 incompleteness theorem ... **58**, 61
 incremental diversity algorithm
 136
 incremental processing **185**
 independent and identically
 distributed **76**, 122, 123
 indirect full connectivity **18**
 individual loss **108**
 induced **189**
 induced metric **189**
 inference **40**
 infinite cardinal **149**
 infinite sequence **220**
 information coherence **196**
 information incoherence **196**
 inherent diversity **134**
 inheritance **80**

initial state **23**
 inner connections **16**
 input neuron **72**
 input-output cut **91**
 interference **185**
 interpolation **178**
 intrinsic diveristy **132**
 intrinsic learning **40**
 inverse neural network **118**
 isoperimetry **197**
 Itakura–Saito distance **55**
 iteration 239

J

Jeffrey’s divergence **54**
 Jensen–Shannon divergence ... **55**

K

Kagan’s divergence **54**
 Kalman filter **233**
 kernel size **28**
 Kleene star **5**
 Kolmogorov complexity **162**
 Kolmogorov superposition theorem
 21
 Kolmogorov–Zurbenko filter . **233**
 Krylov subspace **101**
 Kullback–Leibler divergence .. **54**,
 61

L

lag **243**
 Laplacian smoothing **233**
 layer **10**, 11
 layer bias **18**
 layer locality ratio **24**, 198
 layer weights **18**
 leaky relu activation function . **20**
 learnable **44**
 learner **158**
 learner aggregation **114**
 learner distance **132**
 learning **37**
 learning algorithm **158**
 learning duality **69**, 107

learning ensemble 114
 learning library 310
 learning multiplicity 69
 learning polarity 109
 learning rate 45
 least mean squares method ... 59
 least squares 172, 203
 Lebesgue constant 175
 length of sequence 220
 lexicographic ordering 6
 linear estimator 194
 linear flower 269
 linear flower map 269
 linear flower structure 269
 linguistic nativism) 204
 linked selection 120
 Linpack 73
 Lipschitz constant 197
 Lipschitz continuous 197
 local neural network 26
 local superposition method .. 276
 local theory 22
 locality of reference . 22, 185, 197
 locality ratio 24, 198
 locality-sensitive 180
 locality-sensitive hashing 180, 203
 logistic activation function 20
 Lokayata school of Indian
 philosophy 338
 loss function 41, 50
 loss polarity 109
 low-pass filter 233
 lower convex hull 203

M

machine learning 38
 macroglia 70
 Mahalanobis distance 55
 manual memory management 308
 mathesis universalis 59
 maximal forward partition 10
 mean p -norm error 41
 mean absolute error 41, 51
 mean squared error ... 41, 51, *see*
 MSE

mean-field theory 162
 mean-of-ratios estimator 152
 measurable set 151
 measurable space 151
 measure 151
 measure space 151
 measure theory 148, 151, 200
 Medical Dark Age . 313, 324, 328
 medical-industrial complex .. 324,
 340
 memory coherence 185
 memory leak 308
 Metaphysics 339
 metascience 278, 315, 339
 metric 41
 metric difference 189
 metric difference of order k .. 193
 metric regularization ... 165, 165,
 202
 milieu intérieur 84
 mimesis segment 128
 mimimum description length 162
 mimimum hyperrectangle 178
 min-batch 167
 mini-batch 224
 mini-batch training 167
 minimum bounding box 178
 mitochondrion 78
 mixture of experts 114
 MNIST data set 179, 277
 MNIST SE ensemble 280
 model aggregation 114
 modular arithmetic 200
 modular arithmetic 143
 modulus 143
 momentum gradient descent 247,
 263
 momentum of exponential
 smoothing 222
 momentum of unbiased
 exponential smoothing ... 223
 MSE 130
 Multi Column Deep Neural
 Network 277

multi-index **5**
 multi-label classification . **51**, 157
 multidimensional array **5**
 mutation **38**
 mutual information **125**
 myelin 66, **78**, 85

N

N programming language **309**
 naive interpretation **323**
 Nan **75**
 narrow learning system **161**
 natural science **65**
 natural selection ..31, 82, **89**, 203
 nature **65**
 neighborhood **189**
 neocognitron **34**
 neocortex **87**
 nerve cell **70**
 nerve fiber **70**
 nerve terminal **71**
 nervous system **66**, 83
 nervous system cell 83
 nested split notation **145**
 network pruning **164**
 Neural Age 17, 37
 neural algebra **97**
 neural architecture search **75**
 neural coherence **255**
 neural coherence factorization **256**
 neural drift **122**
 neural equivalence **99**
 neural gradient descent **49**
 neural input **98**
 neural library **310**
 neural network accountability **337**
 neural network loading . *see* NNL,
 61
 neural network opacity **337**
 neural network stochastic gradient
 descent **49**
 neural network transparency .**337**
 neural partitioning **16**
 neural repetition **99**
 neural state **94**

neural stem cell **70**
 neural subset **266**
 neural synergy **96**, 109
 neuro-dynamic programming .. **43**
 neurogenesis **66**
 neuromorphic computing .. **80**, 85
 neuromorphic engineering **85**
 neuron **66**, 70, 72
 neuron doctrine **69**, 83
 neuron theory **69**, 83
 neurons **12**
 neutral theory of evolution .. **120**
 New Essays **203**
 Newton polygon **203**
 NNL **57**
 no free lunch theorem 201
 no free lunch theorems **159**
 node of Ranvier **78**
 nodes **7**
 non-deterministic polynomial time
 57
 non-deterministic Turing machines
 57
 non-local approximation **177**
 norm **165**, 183
 normalized convolution **208**
 normalized correlation **208**
 normalized mutual information
 126
 normed regularization **165**
 normed space **165**
 normed vector space 183
 NP **57**
 NP-complete **57**
 NP-complete problem 57
 NP-hard **57**
 nuclear coherence **255**
 nucleus **70**
 NumPy **172**

O

object **81**
 object coherence **186**
 object-oriented **81**
 observer bias **318**

occipital lobe **87**
 Ockham's razor **161**
 oligopetalous flower **288**
 On the Soul **338**
 online training **167**
 opacity **337**
 Open Science Collaboration . **320**
 opioid epidemic **324**, 340
 optimal brain damage .. **165**, 202
 optimal brain surgeon .. **165**, 202
 optimization convergence **248**
 ordered pair **4**
 ordered subtuple **99**
 ordinal arithmetic **149**
 ordinals **149**
 origin of language **171**
 Origin of Species **203**
 output neuron **72**
 overfitting 161, **168**

P

pair **4**
 Pale Fire **34**
 parallel flower implementation **287**
 parallel neural operation **98**
 parallel neural synergy **100**
 parallel synergetic neural networks
 100
 parametric coherence **243**
 parietal lobe **87**
 parisotes **59**
 partial function **236**
 partial functions 262
 path **7**
 payoff **112**
 perceptive learning **40**
 perceptron algorithm **59**
 peripheries **91**
 periphery **91**
 permissible indices **6**
 perspective **290**
 petaFLOP **73**
 petal **268**
 petal aligned strider sequence **268**
 petal size **269**

petal stride **269**
 petal strides **268**
 Pharmaceutical Research and
 Manufacturers of America . *see*
 PhRMA
 Philosophia Botanica **203**
 PhRMA **327**
 phyletic gradualism **184**
 planes **27**
 plasticity **30**
 player action **112**
 player information **112**
 point estimator **201**
 polarity **109**
 polarity operator **109**
 polynomial approximation ... **172**
 polynomial regression **172**
 Polynomial.fit **172**
 polypetalous flower **288**
 positive definite **55**, **165**
 positive definite matrix 55
 positive predictive value *see* PPV
 positive semi-definite **55**
 power set **51**, 149
 power set cardinal 149
 precision **315**
 predecessor function **237**
 predictability minimization **43**, 61
 prelu activation function **20**
 preprocessing-and-augmentation
 277
 Presocratics **338**
 price of anarchy **129**, 141
 primitive recursive functions . **232**
 principle of coherence **184**
 principle of locality **22**, 197
 Principles of Neural Design ... **84**
 probabilistic dropout **164**
 probability density function ... **52**
 probability distribution **52**
 probability space **151**
 progenitor cell **70**
 projection function 233, **237**, 238
 projection tracts **13**

protein structure prediction . **330**
 Proteus phenomenon ... **317**, 339
 pseudo-Huber loss **51**
 Public Library of Science **315**
 publication bias 278, **318**, 339
 publish-or-perish **318**
 punctuated equilibrium **184**
 puncutated equilibrium 204
 Python **172**
 Pytorch **33**

Q

quantum dropout **164**

R

Racketeering Influenced and
 Corrupt Organizations Act *see*
 RICO, 341
 radiosity coherence **186**
 Ramer–Douglas–Peucker
 algorithm **233**
 rank **199**
 rate of change **183**
 ratio-of-means estimator **153**
 rationalism **313**
 ray coherence **186**
 receptive field **24**
 recurrent network 239
 recurrent neural network **26**
 recursion **211**
 recursion of order k **211**
 recursive transformation **212**
 reflection **277**
 reflexive nerual network **104**
 regular event **84**
 Regular expression **67**
 regular language **67**, 84
 regularization **39**, **161**
 reinforcement learning **43**
 relational algebra **97**
 relational data set **157**
 relative entropy **54**, 61
 relative logical polarity **90**
 relative refractory time **74**
 religion of false medicine **328**

relu activation function **20**
 relu6 activation function **20**
 remote loss **44**
 replication crisis ... **315**, 317, 319,
 339
 reproducibility crisis **317**
 residual connections **13**
 residual error **115**
 residual network **33**
 reticular theory **83**
 RICO **325**
 rigid neural network **101**
 ring strider **268**
 robustness **196**
 rods **31**
 run-time stack **239**
 Runge's phenomenon ... **175**, 203

S

saliency **165**
 saltatory conduction **78**
 sample mean **152**
 sample width **276**
 sampling with replacement .. **114**
 Savitzky–Golay filter **233**
 scan-line algorithm **185**
 scan-line coherence **186**
 scanned symbol **23**
 scientific method **314**, **318**
 SCORE **320**, 340
 SE block **282**
 SEF model **292**
 SEF network **292**
 segmented flower map **269**
 segmented flower striders **268**
 segmented flower structure .. **269**
 segmented flowers **269**
 self-organized map **42**
 self-supervised **43**
 selu activation function **20**
 sequence convolution **208**
 sequence correlation **208**
 sequence cross-correlation ... **208**
 sequence length **220**

- sequential flower implementation
288
- sequential locality 22
- sequential neural operation ... 98
- set theory 148, 200
- sexual selection 82
- Shakespeare 34
- Shapley value 113
- sigma 33
- sigmoids 19
- signal-to-noise ratio *see* SNR
- simple moving average .. 224, 233
- simple moving average of length n
225
- single-label classification 51
- smooth interpolation 196
- smoothing 221
- smoothing factor of exponential
smoothing 222
- smoothing factor of unbiased
exponential smoothing ... 223
- smoothness 196
- SNR 75
- soft clustering 42
- soft sweeps 141
- softmax 52
- softmax function 20
- softmax function 20
- softsign activation function ... 20
- somatic cell 73
- span coherence 186
- sparse coding 77, 85
- sparsity 18
- spatial coherence 186
- spatial dropout 164, 202
- spatial locality 22
- speciation 184, 204
- splines 176, 203
- split notation 143, 213
- split wrapping 146
- split-and-merge algorithm ... 233
- squared Euclidean distance ... 55
- squared Hellinger distance 54
- squeeze-and-excitation block .280,
282
- stack 239
- stacked generalization 114
- stasis 184
- state feedback 100
- state insensitive neural network 96
- state sensitive neural network .96
- state tuple 95
- stateless neural network 96
- statistical angle 56, 61
- stem 72
- stem cell 70
- stem neuron 72
- step function 20
- step size 45
- stochastic approximation 59
- stochastic depth 164, 202
- stochastic estimation 59
- stochastic gradient descent 45, 59,
240
- stochastic training 167
- straight flower strider 268
- straight strider 268
- strictly partial function 236
- stride aligned strider sequence 268
- strider 268
- strider multiplication 273
- strider segment size 268
- subdivision method 185
- subtuple 99
- successive connections 12
- successive neural subsets 266
- successor function . 233, 236, 238
- sunflower 267, 268, 311
- sunflower core 267
- sunflower kernel 267
- sunflower petal 267
- superposition dropout 164
- superposition function 276
- supervised learning 41
- swish activation function 20
- Syc 252
- sylogistic logic 338
- synaptic coherence 198, 243

synaptic cross-gradient coherence
251
 synaptic design **266**
 synaptic domain **266**
 synaptic gradient coherence . **247**
 synaptic incoherence **198**
 synaptic locality **198**
 synaptic map **266**
 synaptic range **266**
 synaptic structure **266**
 synaptic terminal **71**
 Sync **256**
 synergetic child **104**
 synergetic learning **107**
 synergetic neural networks **96**
 synergetic parent **104**
 synergy conjecture **127**
 synergy segment **128**
 synnapse **70**
 synthesis library **310**
 Systematizing Confidence in Open
 Research and Evidence *see*
 SCORE

T

tabula rasa **338**
 tanh activation function **20**
 taxonomy **184**
 Taylor series expansion **183**
 temporal coherence **186**
 temporal lobe **87**
 temporal locality **22**
 TensorFlow **33**
 tensors **16**
 test error of a function **156**
 test error of a learning algorithm
156
 The Blind Watchmaker **204**
 Theano **33**
 theory of coherence in optics **204**
 Tikhonov matrix **202**
 Tikhonov regularization **202**
 time delay neural network **34**
 Timon of Athens **34**
 topological sort **199**

Torch **33**
 total function **236**
 total variation norm **54**
 training convex hull **178**
 training epoch **167**
 training hyperrectangle **178**
 trait **132**
 trait operator **132**
 transfinite recursion **211**
 transformation bias **231**
 transition function **23**
 transpose **6**
 triangle inequality **41, 165**
 Truing machine **214**
 tuple **5**
 Turing machine **22, 33, 238**
 Turing-computable function . **233**

U

ulterior motive hypothesis ... **323**
 unbiased estimator **152**
 unbiased exponential smoothing
223, 263
 unbiased transformation **231**
 uncorrelated random variables **260**
 uncountable set **150**
 undefinability theorem **58**
 under-constrained problem .. **161**
 under-constrained system **161**
 underfitting **170**
 unfolding **287**
 uniform convergence **175**
 unit **72**
 unit activation function **19**
 unit bias **18**
 unit weights **18**
 units **12**
 universal **237**
 universal computing **22**
 universal grammar **171, 204**
 Unix **309**
 unordered pair **4**
 unsupervised learning **42**
 update momentum gradient
 descent **247**

V

validation set 169
 value function **112**
 Vandemonde determinant 203
 Vandemonde matrix 203
 Vandermonde determinant ... **175**
 Vandermonde matrix **175**
 variable number of arguments **211**
 vertices **7**
 vestigiality **120**
 volume of hypercube **178**
 volume of hypershpere **178**
 voting ensemble **122**

W

wave interference **185**

waveform 185

Weierstrass approximation

 theorem **174**, 203

weight decay **166**, 241

weight decay exponent **166**

well-posed problem **161**

word embedding **42**, 61

word2vec **61**

wrapping **146**

Z

Zermelo-Fraenkel axiomatic set

 theory **200**

zero-sum game **61**

ZF **200**

ZFC **152**, 200

