

Undergraduate Topics in Computer Science


Sandro Skansi

Introduction to Deep Learning

From Logical Calculus to Artificial
Intelligence



 Springer

Sandro Skansi 
University of Zagreb
Zagreb
Croatia

ISSN 1863-7310 ISSN 2197-1781 (electronic)
Undergraduate Topics in Computer Science
ISBN 978-3-319-73003-5 ISBN 978-3-319-73004-2 (eBook)
<https://doi.org/10.1007/978-3-319-73004-2>

Library of Congress Control Number: 2017963994

© Springer International Publishing AG, part of Springer Nature 2018

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Printed on acid-free paper

This Springer imprint is published by the registered company Springer International Publishing AG part of Springer Nature

The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Contents

1	From Logic to Cognitive Science	1
1.1	The Beginnings of Artificial Neural Networks	1
1.2	The XOR Problem	5
1.3	From Cognitive Science to Deep Learning	8
1.4	Neural Networks in the General AI Landscape	11
1.5	Philosophical and Cognitive Aspects	12
	References.	15
2	Mathematical and Computational Prerequisites	17
2.1	Derivations and Function Minimization	17
2.2	Vectors, Matrices and Linear Programming	25
2.3	Probability Distributions	32
2.4	Logic and Turing Machines	39
2.5	Writing Python Code	41
2.6	A Brief Overview of Python Programming.	43
	References.	49
3	Machine Learning Basics	51
3.1	Elementary Classification Problem	51
3.2	Evaluating Classification Results	57
3.3	A Simple Classifier: Naive Bayes.	59
3.4	A Simple Neural Network: Logistic Regression	61
3.5	Introducing the MNIST Dataset	68
3.6	Learning Without Labels: K-Means	70
3.7	Learning Different Representations: PCA	72
3.8	Learning Language: The Bag of Words Representation	75
	References.	77
4	Feedforward Neural Networks	79
4.1	Basic Concepts and Terminology for Neural Networks	79
4.2	Representing Network Components with Vectors and Matrices	82
4.3	The Perceptron Rule	84
4.4	The Delta Rule	87

4.5	From the Logistic Neuron to Backpropagation	89
4.6	Backpropagation	93
4.7	A Complete Feedforward Neural Network	102
	References.	105
5	Modifications and Extensions to a Feed-Forward Neural Network	107
5.1	The Idea of Regularization	107
5.2	L_1 and L_2 Regularization	109
5.3	Learning Rate, Momentum and Dropout.	111
5.4	Stochastic Gradient Descent and Online Learning	116
5.5	Problems for Multiple Hidden Layers: Vanishing and Exploding Gradients	118
	References.	119
6	Convolutional Neural Networks	121
6.1	A Third Visit to Logistic Regression	121
6.2	Feature Maps and Pooling	125
6.3	A Complete Convolutional Network.	127
6.4	Using a Convolutional Network to Classify Text	130
	References.	132
7	Recurrent Neural Networks	135
7.1	Sequences of Unequal Length	135
7.2	The Three Settings of Learning with Recurrent Neural Networks	136
7.3	Adding Feedback Loops and Unfolding a Neural Network	139
7.4	Elman Networks	140
7.5	Long Short-Term Memory	142
7.6	Using a Recurrent Neural Network for Predicting Following Words	145
	References.	152
8	Autoencoders	153
8.1	Learning Representations	153
8.2	Different Autoencoder Architectures.	156
8.3	Stacking Autoencoders	158
8.4	Recreating the Cat Paper	161
	References.	163
9	Neural Language Models	165
9.1	Word Embeddings and Word Analogies.	165
9.2	CBOw and Word2vec	166
9.3	Word2vec in Code	168

9.4	Walking Through the Word-Space: An Idea That Has Eluded Symbolic AI	171
	References.	173
10	An Overview of Different Neural Network Architectures.	175
10.1	Energy-Based Models.	175
10.2	Memory-Based Models.	178
10.3	The Kernel of General Connectionist Intelligence: The bAbI Dataset	181
	References.	182
11	Conclusion	185
11.1	An Incomplete Overview of Open Research Questions	185
11.2	The Spirit of Connectionism and Philosophical Ties	186
	Reference	187
Index	189

1.1 The Beginnings of Artificial Neural Networks

Artificial intelligence has its roots in two philosophical ideas of Gottfried Leibniz, the great seventeenth-century philosopher and mathematician, viz. the *characteristica universalis* and the *calculus ratiocinator*. The *characteristica universalis* is an idealized language, in which all of science could in principle be translated. It would be a language in which every natural language would translate, and as such it would be the language of pure meaning, uncluttered by linguistic technicalities. This language can then serve as a background for explicating rational thinking, in a manner so precise, a machine could be made to replicate it. The *calculus ratiocinator* would be a name for such a machine. There is a debate among historians of philosophy whether this would mean making a software or a hardware, but this is in fact an insubstantial question since to get the distinction we must understand the concept of a universal machine accepting different instructions for different tasks, an idea that would come from Alan Turing in 1936 [1] (we will return to Turing shortly), but would become clear to a wider scientific community only in the late 1970s with the advent of the personal computer. The ideas of the *characteristica universalis* and the *calculus ratiocinator* are Leibniz' central ideas, and are scattered throughout his work, so there is no single point to reference them, but we point the reader to the paper [2], which is a good place to start exploring.

The journey towards deep learning continues with two classical nineteenth century works in logic. This is usually omitted since it is not clearly related to neural networks, there was a strong influence, which deserves a couple of sentences. The first is John Stuart Mill's *System of Logic* from 1843 [3], where for the first time in history, logic is explored in terms of a manifestation of a mental process. This approach,

called *logical psychologism*, is still researched only in philosophical logic,¹ but even in philosophical logic it is considered a fringe theory. Mill's book never became an important work, and his work in ethics overshadowed his contribution to logical psychologism, but fortunately there was a second book, which was highly influential. It was the *Laws of Thought* by George Boole, published in 1854 [4]. In his book, Boole systematically presented logic as a system of formal rules which turned out to be a major milestone in the reshaping of logic as a formal science. Quickly after, formal logic developed, and today it is considered a native branch of both philosophy and mathematics, with abundant applications to computer science. The difference in these 'logics' is not in the techniques and methodology, but rather in applications. The core results of logic such as De Morgan's laws, or deduction rules for first-order logic, remain the same across all sciences. But exploring formal logic beyond this would take us away from our journey. What is important here is that during the first half of the twentieth century, logic was still considered to be something connected with the laws of thinking. Since thinking was the epitome of intelligence, it was only natural that artificial intelligence started out with logic.

Alan Turing, the father of computing, marked the first step of the birth of artificial intelligence with his seminal 1950 paper [5] by introducing the Turing test to determine whether a computer can be regarded intelligent. A Turing test is a test in natural language administered to a human (who takes the role of the referee). The human communicates with a person and a computer for five minutes, and if the referee cannot tell the two apart, the computer has passed the Turing test and it may be regarded as intelligent. There are many modifications and criticism, but to this day the Turing test is one of the most widely used benchmarks in artificial intelligence.

The second event that is considered the birth of artificial intelligence was the Dartmouth Summer Research Project on Artificial Intelligence. The participants were John McCarthy, Marvin Minsky, Julian Bigelow, Donald MacKay, Ray Solomonoff, John Holland, Claude Shannon, Nathaniel Rochester, Oliver Selfridge, Allen Newell and Herbert Simon. Quoting the proposal, the conference was *to proceed on the basis of the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it.*² This premise made a substantial mark in the years to come, and mainstream AI would become logical AI. This logical AI would go unchallenged for years, and would eventually be overthrown only in the 21 millennium by a new tradition, known today as deep learning. This tradition was actually older, founded more than a decade earlier in 1943, in a paper written by a logician of a different kind, and his co-author, a philosopher and psychiatrist. But, before we continue, let us take a small step back. The interconnection between logical rules and thinking was seen as directed. The common knowledge is that the logical rules are grounded in thinking. Artificial intelligence asked whether we can impersonate thinking in a machine with

¹Today, this field of research can be found under a refreshing but very unusual name: 'logic in the wild'.

²The full text of the proposal is available at <https://www.aaai.org/ojs/index.php/aimagazine/article/view/1904/1802>.

logical rules. But there was another direction which is characteristic of philosophical logic: could we model thinking as a human mental process with logical rules? This is where the neural network history begins, with the seminal paper by Walter Pitts and Warren McCulloch titled *A Logical Calculus of Ideas Immanent in Nervous Activity* and published in the *Bulletin of Mathematical Biophysics*. A copy of the paper is available at <http://www.cs.cmu.edu/~epxing/Class/10715/reading/McCulloch.and.Pitts.pdf>, and we advise the student to try to read it to get a sense of how deep learning began.

Warren McCulloch was a philosopher, psychologist and psychiatrist by degree, but he would work in neurophysiology and cybernetics. He was a vivid character, embodying many academic stereotypes, and as such was a curious person whose interests could be described as interdisciplinary. He met the homeless Walter Pitts in 1942 when he got a job at the Department of Psychiatry at the University of Chicago, and invited Pitts to come to live with his family. They shared a lifelong interest in Leibniz, and they wanted to bring his ideas to fruition and create a machine which could implement logical reasoning.³ The two men worked every night on their idea of capturing reasoning with a logical calculus inspired by the biological neurons. This meant constructing a formal neuron with capabilities similar to that of a Turing machine. The paper had only three references, and all of them are classical works in logic: Carnap's *Logical Syntax of Language* [6], Russell's and Whitehead's *Principia Mathematica* [7] and the Hilbert and Ackermann *Grundzüge der Theoretischen Logik*. The paper itself approached the problem of neural networks as a logical one, proceeding from definitions, over lemmas to theorems.

Their paper introduced the idea of the artificial neural network, as well as some of the definitions we take for granted today. One of these is what would it mean for a logical predicate to be realizable on a neural network. They divided the neurons in two groups, the first called *peripheral afferents* (which are now called 'input neurons'), and the rest, which are actually output neurons, since at this time there was no hidden layer—the hidden layer came to play only in the 1970s and 1980s. Neurons can be in two states, firing and non-firing, and they define for every neuron i a predicate which is true when the neuron is firing at the moment t . This predicate is denoted as $N_i(t)$. The *solution* of a network is then an equivalence of the form $N_i(t) \equiv B$ where B is a conjunction of firings from the previous moment of the peripheral afferents, and i is not an input neuron. A sentence like this is realizable in a neural network if and only if the network can compute it, and all sentences for which there is a network which computes them are called a *temporal propositional expression (TPE)*. Notice that *TPEs* have a logical characterization. The main result of the paper (asides from defining artificial neural networks) is that any *TPE* can be computed by an artificial neural network. This paper would be cited later by John von Neumann as a major influence in his own work. This is just a short and incomplete glimpse into this exciting historical paper, but let us return to the story of the second protagonist.

³This was 15 years before artificial intelligence was defined as a scientific field.

Walter Pitts was an interesting person, and, one could argue, *the* father of artificial neural networks. At the age of 12, he ran away from home and hid in a library, where he read *Principia Mathematica* [7] by the famous logician Bertrand Russell. Pitts contacted Russell, who invited him to come to study at Cambridge under his tutorship, but Pitts was still a child. Several years later, Pitts, now a teenager, found out that Russell was holding a lecture at the University of Chicago. He met with Russell in person, and Russell told him to go and meet his old friend from Vienna, the logician Rudolph Carnap, who was a professor there. Carnap gave Pitts his seminal book *Logical Syntax of Language*⁴ [6], which would highly influence Pitts in the following years. After his initial contact with Carnap, Pitts disappeared for a year, and Carnap could not find him, but after he did, he used his academic influence to get Pitts a student job at the university, so that Pitts does not have to do menial jobs during days and ghostwrite student papers during nights just to survive.

Another person Pitts met during Russell was Jerome Lettvin, who at the time was a pre-med student there, and who would later become neurologist and psychiatrist by degree, but he will also write papers in philosophy and politics. Pitts and Lettvin became close friends, and would eventually write an influential paper together (along with McCulloch and Maturana) titled *What the Frog's Eye Tells the Frog's Brain* in 1959 [8]. Lettvin would also introduce Pitts to the mathematician Norbert Wiener from MIT who later became known as the father of cybernetics, a field colloquially known as 'the science of steering', dedicated to studying system control both in biological and artificial systems. Wiener invited Pitts to come to work at MIT (as a lecturer in formal logic) and the two men worked together for a decade. Neural networks were at this time considered to be a part of cybernetics, and Pitts and McCulloch were very active in the field, both attending the Macy conferences, with McCulloch becoming the president of the American Society for Cybernetics in 1967–1968. During his stay at Chicago, Pitts also met the theoretical physicist Nicolas Rashevsky, who was a pioneer in mathematical biophysics, a field which tried to explain biological processes with a combination of logic and physics. Physics might seem distant to neural networks, but in fact, we will soon discuss the role physicists played in the history of deep learning.

Pitts would remain connected with the University, but he had minor jobs there due to his lack of formal academic credentials, and in 1944 was hired by the Kellogg Corporation (with the help of Wiener), which participated in the Manhattan project. He detested the authoritarian General Groves (head of the Manhattan project), and would play pranks to mock the strict and sometimes meaningless rules that he enacted. He was granted an Associate of Arts degree (2-year degree) by the University of Chicago as a token of recognition of his 1943 paper, and this would remain the only academic degree he ever earned. He has never been fond of the usual academic procedures and this posed a major problem in his formal education. As an illustration, Pitts attended a

⁴The author has a fond memory of this book, but beware: here be dragons. The book is highly complex due to archaic notation and a system quite different from today's logic, but it is a worthwhile read if you manage to survive the first 20 pages.

course taught by professor Wilfrid Rall (the pioneer of computational neuroscience), and Rall remembered Pitts as ‘an oddball who felt compelled to criticize exam questions rather than answer them’.

In 1952, Norbert Weiner broke all relations with McCulloch, which devastated Pitts. Weiner wife accused McCulloch that his boys (Pitts and Lettvin) seduced their daughter, Barbara Weiner. Pitts turned to alcohol to the point that he could not take care of his dog anymore,⁵ and succumbed to cirrhosis complications in 1969, at the age of 46. McCulloch died the same year at the age of 70. Both of the Pitts’ papers we mentioned remain to this day two of the most cited papers in all of science. It is interesting to note that even though Pitts had direct or mediated contact with most of the pioneers of AI, Pitts himself never thought about his work as geared towards building a machine replica of the mind, but rather as a quest to formalize and better understand human thinking [9], and that puts him squarely in the realm of what is known today as philosophical logic.⁶

The story of Walter Pitts is a story of influences of ideas and of collaboration between scientists of different backgrounds, and in a way a neural network nicely symbolizes this interaction. One of the main aims of this book is to (re-)introduce neural networks and deep learning to all the disciplines⁷ which contributed to the birth and formation of the field, but currently shy away from it. The majority of the story about Walter Pitts we presented is taken from a great article named *The man who tried to redeem the world with logic* by Amanda Geffer published in Nautilus [10] and the paper *Walter Pitts* by Neil R. Smalheiser [9], both of which we highly recommend.⁸

1.2 The XOR Problem

In the 1950s, the Dartmouth conference took place and the interest of the newly born field of artificial intelligence in neural networks was evident from the very conference manifest. Marvin Minsky, one of the founding fathers of AI and participant to the Dartmouth conference was completing his dissertation at Princeton in 1954, and the title was *Neural Nets and the Brain Model Problem*. Minsky’s thesis addressed several technical issues, but it became the first publication which collected all up to date results and theorems on neural networks. In 1951, Minsky built a machine (funded

⁵A Newfoundland, name unknown.

⁶An additional point here is the great influence of Russell and Carnap on Pitts. It is a great shame that many logicians today do not know of Pitts, and we hope the present volume will help bring the story about this amazing man back to the community from which he arose, and that he will receive the place he deserves.

⁷And any other scientific discipline which might be interested in studying or using deep neural networks.

⁸Also, there is a webpage on Pitts <http://www.abstractnew.com/2015/01/walter-pitts-tribute-to-unknown-genius.html> worth visiting.

by the Air Force Office of Scientific Research) which implemented neural networks called SNARC (Stochastic Neural Analog Reinforcement Calculator), which was the first major computer implementation of a neural network. As a bit of trivia, Marvin Minsky was an advisor to Arthur C. Clarke's and Stanley Kubrick's *2001: A Space Odyssey* movie. Also, Isaac Asimov claimed that Marvin Minsky was one of two people he has ever met whose intelligence surpassed his own (the other one being Carl Sagan). Minsky will return to our story soon, but first let us present another hero of deep learning.

Frank Rosenblatt received his PhD in Psychology at Cornell University in 1956. Rosenblatt made a crucial contribution to neural networks, by discovering the *perceptron learning rule*, a rule which governs *how* to update the weights of neural networks, which we shall explore in detail in the forthcoming chapters. His perceptrons were initially developed as a program on an IBM 704 computer at Cornell Aeronautical Laboratory in 1957, but Rosenblatt would eventually develop the Mark I Perceptron, a computer built with the sole purpose of implementing neural networks with the perceptron rule. But Rosenblatt did more than just implement the perceptron. His 1962 book *Principles of Neurodynamics* [11] explored a number of architectures, and his paper [12] explored the idea of multilayered networks similar to modern convolutional networks, which he called *C-system*, which might be seen as the theoretical birth of deep learning. Rosenblatt died in 1971 on his 43rd birthday in a boating accident.

There were two major trends underlying the research in the 1960s. The first one was the results that were delivered by programs working on symbolic reasoning, using deductive logical systems. The two most notable were the Logic Theorist by Herbert Simon, Cliff Shaw and Allen Newell, and their later program, the General Problem Solver [13]. Both programs produced working results, something neural networks did not. Symbolic systems were also appealing since they seemed to provide control and easy extensibility. The problem was not that neural networks were not giving any result, just that the results they have been giving (like image classification) were not really considered that intelligent at the time, compared to symbolic systems that were proving theorems and playing chess—which were the hallmark of human intelligence. The idea of this intelligence hierarchy was explored by Hans Moravec in the 1980s [14], who concluded that symbolic thinking is considered a rare and desirable aspect of intelligence in humans, but it comes rather natural to computers, which have much more trouble with reproducing ‘low-level’ intelligent behaviour that many humans seem to exhibit with no trouble, such as recognizing that an animal in a photo is a dog, and picking up objects.⁹

The second trend was the Cold War. Starting with 1954, the US military wanted to have a program to automatically translate Russian documents and academic papers.

⁹Even today people consider playing chess or proving theorems as a higher form of intelligence than for example gossiping, since they point to the rarity of such forms of intelligence. The rarity of an aspect of intelligence does not directly correlate with its computational properties, since problems that are computationally easy to describe are easier to solve regardless of the cognitive rarity in humans (or machines for that matter).

Funding was abundant, but many technically inclined researchers underestimated the linguistic complexities involved in extracting meaning from words. A famous example was the back and forth translation from English to Russian and back to English of the phrase ‘the spirit was willing but the flesh was weak’ which produced the sentence ‘the vodka was good, but the meat was rotten’. In 1964, there were some concerns about wasting government money in a dead end, so the National Research Council formed the Automatic Language Processing Advisory Committee or ALPAC [13]. The ALPAC report from 1966 cut funding to all machine translation projects, and without the funding, the field lingered. This in turn created turmoil in the whole AI community.

But the final stroke which nearly killed off neural networks came in 1969, from Marvin Minsky and Seymour Papert [15], in their monumental book *Perceptrons: An Introduction to Computational Geometry*. Remember that McCulloch and Pitts proved that a number of logical functions can be computed with a neural network. It turns out, as Minsky and Papert showed in their book, they missed a simple one, the equivalence. The computer science and AI community tend to favour looking at this problem as the XOR function, which is the negation of an equivalence, but it really does not matter, since the only thing different is how you place the labels.

It turns out that perceptrons, despite the peculiar representations of the data they process, are only linear classifiers. The perceptron learning procedure is remarkable, since it is guaranteed to converge (terminate), but it did not add a capability of capturing nonlinear regularities to the neural network. The XOR is a nonlinear problem, but this is not clear at first.¹⁰ To see the problem, imagine¹¹ a simple 2D coordinate system, with only 0 and 1 on both axes. The XOR of 0 and 0 is 0, and write an O at coordinates (0, 0). The XOR of 0 and 1 is 1, and now write an X at the coordinates (0,1). Continue with $\text{XOR}(1, 0) = 1$ and $\text{XOR}(1, 1) = 0$. You should have two Xs and two Os. Now imagine you are the neural network, and you have to find out how to draw a curve to separate the Xs from Os. If you can draw anything, it is easy. But you are not a modern neural network, but a perceptron, and you must use a straight line—no curves. It soon becomes obvious that this is impossible.¹² The problem with the perceptron was the linearity. The idea of a multilayered perceptron was here, but it was impossible to build such a device with the perceptron learning rule. And so, seemingly, no neural network could handle (learn to compute) even the basic logical operations, something symbolic systems could do in an instant. A quiet darkness fell across the neural networks, lasting many years. One might wonder what was happening in the USSR at this time, and the short answer is that cybernetics, as neural networks were still called in the USSR in this period, was considered a bourgeois pseudoscience. For a more detailed account, we refer the reader to [16].

¹⁰The view is further dimmed by the fact that the perceptron could process an image (at least rudimentary), which intuitively seems to be quite harder than simple logical operations.

¹¹Pick up a pen and paper and draw along.

¹²If you wish to try the equivalence instead of XOR, you should do the same but with $\text{EQUIV}(0, 0) = 1$, $\text{EQUIV}(0, 1) = 0$, $\text{EQUIV}(1, 0) = 0$, $\text{EQUIV}(1, 1) = 1$, keeping the Os for 0 and Xs for 1. You will see it is literally the same thing as XOR in the context of our problem.

1.3 From Cognitive Science to Deep Learning

But the idea of neural networks lingered on in the minds of only a handful of believers. But there were processes set in motion which would enable their return in style. In the context of neural networks, the 1970s were largely uneventful. But there were two trends present which would help the revival of the 1980s. The first one was the advent of cognitivism in psychology and philosophy. Perhaps the most basic idea that cognitivism brought in the mainstream is the idea that the mind, as a complex system made from many interacting parts, should be explored on its own (independent of the brain), but with formal methods.¹³ While the neurological reality that determines cognition should not be ignored, it can be helpful to build and analyse systems that try to recreate portions of the neurological reality, and at the same time they should be able to recreate some of the behaviour. This is a response to both Skinner's behaviourism [18] in psychology of the 1950s, which aimed to focus a scientific study of the mind as a black box processor (everything else is purely speculation¹⁴) and to the dualism of the mind and brain which was strongly implied by a strict formal study of knowledge in philosophy (particularly as a response to Gettier [19]).

Perhaps one of the key ideas in the whole scientific community at that time was the idea of a paradigm shift in science, proposed by Thomas Kuhn in 1962 [20], and this was undoubtedly helpful to the birth of cognitive science. By understanding the idea of the paradigm shift, for the first time in history, it felt legitimate to abandon a state-of-the-art method for an older, underdeveloped idea and then dig deep into that idea and bring it to a whole new level. In many ways, the shift proposed by cognitivism as opposed to the older behavioural and causal explanations was a shift from studying an immutable structure towards the study of a mutable change. The first truly cognitive turn in the so-called cognitive sciences is probably the turn made in linguistics by Chomsky's universal grammar [21] and his earlier and ingenious attack on Skinner [22]. Among other early contributions to the cognitive revolution, we find the most interesting one the paper from our old friends [23]. This paradigm shift happened across six disciplines (the cognitive sciences), which would become the founding disciplines of cognitive science: anthropology, computer science, linguistics, neuroscience, philosophy and psychology.

The second was another setback in funding caused by a government report. It was the paper *Artificial Intelligence: A General Survey* by James Lighthill [24], which was presented to the British Science Research Council in 1973, and became widely known as the Lighthill report. Following the Lighthill report, the British government would close all but three AI departments in the UK, which forced many scientists to abandon their research projects. One of the three AI departments that survived was Edinburgh. The Lighthill report enticed one Edinburgh professor to issue a statement, and in this statement, cognitive science was referenced for the first time

¹³A great exposition of the cognitive revolution can be found in [17].

¹⁴It must be acknowledged that Skinner, by insisting on focusing only on the objective and measurable parts of the behaviour, brought scientific rigor into the study of behaviour, which was previously mainly a speculative area of research.

in history, and its scope was roughly defined. It was Christopher Longuet-Higgins, Fellow of the Royal Society, a chemist by formal education, who began work in AI in 1967 when he took a job at the University of Edinburgh, where he joined the Theoretical Psychology Unit. In his reply,¹⁵ Longuet-Higgins asked a number of important questions. He understood that Lighthill wanted the AI community to give a proper justification of AI research. The logic was simple, if AI does not work, why do we want to keep it? Longuet-Higgins provided an answer, which was completely in the spirit of McCulloch and Pitts: we need AI not to build machines (although that would be nice), but to understand humans. But Lighthill was aware of this line of thought, and he has acknowledged in his report that some aspects, in particular neural networks, are scientifically promising. He thought that the study of neural networks can be understood and reclassified as *Computer-based studies of the central nervous system*, but it had to abide by the latest findings of neuroscience, and model neurons as they are, and not weird variations of their simplifications. This is where Longuet-Higgins diverged from Lighthill. He used an interesting metaphor: just like hardware in computers is only a part of the whole system, so is actual neural brain activity, and to study what a computer does, one needs to look at the software, and so to see what a human does, one need to look at mental processes, and *how they interact*. Their interaction is the basis of cognition, all processes taking parts are cognitive processes, and AI needs to address the question of their interaction in a precise and formal way. This is the true knowledge gained from AI research: understanding, modelling and formalizing the interactions of cognitive processes. An this is why we need AI as a field and all of its simplified and sometimes inaccurate and weird models. This is the true scientific gain from AI, and not the technological, martial and economic gain that was initially promised to obtain funding.

Before the turn of the decade, another thing happened, but it went unnoticed. Up until now, the community knew how to train a single-layer neural network, and that having a hidden layer would greatly increase the power of neural networks. The problem was, nobody knew how to train a neural network with more than one layer. In 1975, Paul Werbos [25], an economist by degree, discovered backpropagation, a way to propagate the error back through the hidden (middle) layer. His discovery went unnoticed, and was rediscovered by David Parker [26], who published the result in 1985. Yann LeCun also discovered backpropagation in 1985 and published in [27]. Backpropagation was discovered for the last time in San Diego, by Rumelhart, Hinton and Williams [28], which takes us to the next part of our story, the 1980s, in sunny San Diego, to the cognitive era of deep learning.

The San Diego circle was composed of several researchers. Geoffrey Hinton, a psychologist, was a PhD student of Christopher Longuet-Higgins back in the Edinburgh AI department, and there he was looked down upon by the other faculty, because he wanted to research neural networks, so he called them *optimal networks*

¹⁵The full text of the reply is available from http://www.chilton-computing.org.uk/inf/literature/reports/lighthill_report/p004.htm.

to avoid problems.¹⁶ After graduating (1978), he came to San Diego as a visiting scholar to the Cognitive Science program at UCSD. There the academic climate was different, and the research in neural networks was welcome. David Rumelhart was one of the leading figures in UCSD. A mathematical psychologist, he is one of the founding fathers of cognitive science, and the person who introduced artificial neural networks as a major topic in cognitive science, under the name of *connectionism*, which had wide philosophical appeal, and is still one of the major theories in the philosophy of mind. Terry Sejnowski, a physicist by degree and later professor of computational biology, was another prominent figure in UCSD at the time, and he co-authored a number of seminal papers with Rumelhart and Hinton. His doctoral advisor, John Hopfield was another physicist who became interested in neural networks, and improved and popularized a recurrent neural network model called the *Hopfield Network* [29]. Jeffrey Elman, a linguist and cognitive science professor at UCSD, who would introduce Elman networks a couple of years later, and Michael I. Jordan, a psychologist, mathematician and cognitive scientist who would introduce Jordan networks (both of these networks are commonly called *simple recurrent networks* in today's literature), also belonged to the San Diego circle.

This leads us to the 1990s and beyond. The early 1990s were largely uneventful, as the general support of the AI community shifted towards *support vector machines* (SVM). These machine learning algorithms are mathematically well founded, as opposed to neural networks which were interesting from a philosophical standpoint, and mainly developed by psychologists and cognitive scientists. To the larger AI community, which still had a lot of the GOFAI drive for mathematical precision, they were uninteresting, and SVMs seemed to produce better results as well. A good reference book for SVMs is [30]. In the late 1990s, two major events occurred, which produced neural networks which are even today the hallmark of deep learning. The *long short-term memory* was invented by Hochreiter and Schmidhuber [31] in 1997, which continue to be one of the most widely used recurrent neural network architectures and in 1998 LeCun, Bottou, Bengio and Haffner produced the first convolutional neural network called LeNet-5 which achieved significant results on the MNIST dataset [32]. Both convolutional neural networks and LSTMs went unnoticed by the larger AI community, but the events were set in motion for neural networks to come back one more time. The final event in the return of neural networks was the 2006 paper by Hinton, Osindero and Teh [33] which introduced *deep belief networks* (DBN) which produces significantly better results on the MNIST dataset. After this paper, the rebranding of deep neural networks to deep learning was complete, and a new period in AI history would begin. Many new architectures followed, and some of them we will be exploring in this book, while some we leave to the reader to explore by herself. We prefer not to write too much about recent history, since it is still actual and there is a lot of factors at stake which hinder objectivity.

¹⁶The full story about Hinton and his struggles can be found at <http://www.chronicle.com/article/The-Believers/190147>.

For an exhaustive treatment of the history of neural networks, we point the reader to the paper by Jürgen Schmidhuber [34].

1.4 Neural Networks in the General AI Landscape

We have explored the birth of neural networks from philosophical logic, the role psychology and cognitive science played in their development and their grand return to mainstream computer science and AI. One question that is particularly interesting is where do artificial neural networks live in the general AI landscape. There are two major societies that provide a formal classification of AI, which is used in their publications to classify a research paper, the American Mathematical Society (AMS) and the Association for Computing Machinery (ACM). The AMS maintains the so-called *Mathematics Subject Classification 2010* which divides AI into the following subfields¹⁷: General, Learning and adaptive systems, Pattern recognition and speech recognition, Theorem proving, Problem solving, Logic in artificial intelligence, Knowledge representation, Languages and software systems, Reasoning under uncertainty, Robotics, Agent technology, Machine vision and scene understanding and Natural language processing. The ACM classification¹⁸ for AI provides, in addition to subclasses of AI, their subclasses as well. The subclasses of AI are: Natural language processing, knowledge representation and reasoning, planning and scheduling, search methodologies, control methods, philosophical/theoretical foundations of AI, distributed artificial intelligence and computer vision. Machine learning is a parallel category to AI, not subordinated to it.

What can be concluded from these two classifications is that there are a few broad fields of AI, inside which all other fields can be subsumed:

- Knowledge representation and reasoning,
- Natural language processing,
- Machine Learning,
- Planning,
- Multi-agent systems,
- Computer vision,
- Robotics,
- Philosophical aspects.

In the simplest possible view, deep learning is a name for a specific class of artificial neural networks, which in turn are a special class of machine learning algorithms, applicable to natural language processing, computer vision and robotics. This is a very simplistic view, and we think it is wrong, not because it is not true (it is true), but

¹⁷See <http://www.ams.org/msc/>.

¹⁸See <http://www.acm.org/about/class/class/2012>.

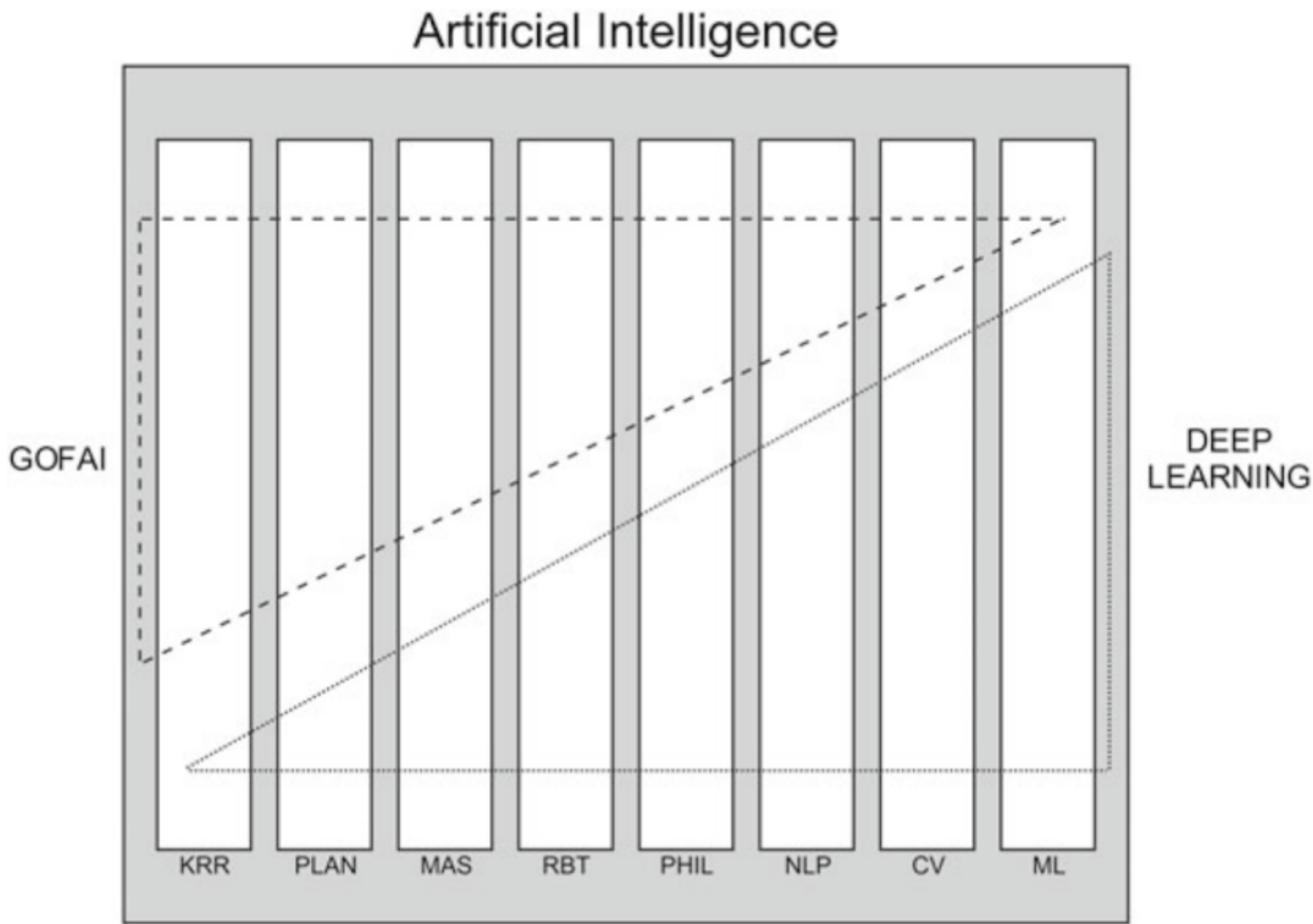


Fig. 1.1 Vertical and horizontal components of AI

because it misses an important aspect. Recall the Good Old-Fashioned AI (GOFAI), and consider what it is. Is it a subdiscipline of AI? The best answer is to think of subdivisions of AI as vertical components, and of GOFAI as a horizontal component that spans considerably more work in knowledge representation and reasoning than in computer vision (see Fig. 1.1). Deep learning, in our thinking, constitutes a second horizontal component, trying to unify across disciplines just as GOFAI did. Deep learning and GOFAI are in a way contenders to the whole AI, wanting to address all questions of AI with their respective methods: they both have their ‘strongholds’,¹⁹ but they both try to encompass as much of AI as they can. The idea of deep learning being a separate influence is explored in detail in [35], where the deep learning movement is called ‘connectionist tribe’.

1.5 Philosophical and Cognitive Aspects

So far, we have explored neural networks from a historical perspective, but there are two important things we have not explained. First, what the word ‘cognitive’ means. The term itself comes from neuroscience [36], where it has been used to characterize outward manifestations of mental behaviour which originates in the cortex. The what exactly comprises these abilities is non-debatable, since neuroscience grounds this division upon neural activity. A cognitive process in the context of AI is then an

¹⁹Knowledge representation and reasoning for GOFAI, machine learning for deep learning.

imitation of any mental process taking place in the human cortex. Philosophy also wants to abstract away from the brain, and define its terms in a more general setting. A working definition of ‘cognitive process’ might be: any process taking place in a similar way in the brain and the machine. This definition commits us to define ‘similar way’, and if we take artificial neural networks to be a simplified version of the real neuron, this might work for our needs here.

This leads us to the bigger issue. Some cognitive processes are simpler, and we could model them easily. Advances in deep learning sweep away one cognitive process at the time, but there is one major cognitive process eludes deep learning—reasoning. Capturing and describing reasoning is the very core of philosophical logic, and formal logic as the main method for a rigorous treatment of reasoning has been the cornerstone of GOFAI. Will deep learning ever conquer reasoning? Or is learning simply a process fundamentally different from reasoning? This would mean that reasoning is not learnable *in principle*. This discussion resonates the old philosophical dispute between rationalists and empiricists, where rationalists argued (in different ways) that there is a logical framework in our minds prior to any learning. A formal proof that no machine learning system could learn reasoning which is considered a distinctly human cognitive process would have a profound technological, philosophical and even theological significance.

The question about learning to reason can be rephrased. It is widely believed that dogs cannot learn relations.²⁰ A dog would then be an example of a trainable cognitive system incapable of learning relations. Suppose we want to teach a dog the relation ‘smaller’. We could devise a training setting where we hand the dog two different objects, and the dog should pick the smaller one when hearing the command ‘smaller’ (and he is rewarded for the right pick). But the task for the dog is very complex: he has to realize that ‘smaller’ is not a name of a single object which changes reference from one training sample to the next, but something immaterial that comes into existence when you have both objects, and then resolves to refer to a single object (the smaller one). If you think about it like that, the difficulties of learning relations become clearer.

Logic is inherently relational, and everything there is a relation. Relational reasoning is accomplished by formal rules and poses no problem. But logic has the very same problem (but seen from the other side): how to learn content for relations? The usual procedure was to hand define entities and relations and then perhaps add a dynamical factor which would modify them over time. But the divide between patterns and relations exists on both sides.

²⁰Whether this is true or not, is irrelevant for our discussion. The literature on animal cognitive abilities is notoriously hard to find as there are simply not enough academic studies connecting animal cognition and ethology. We have isolated a single paper dealing with limitations of dog learning [37], and therefore we would not dare to claim anything categorical—just hypothetical.

The paper that exposed this major philosophical issue in artificial neural networks and connectionism, is the seminal paper by Fodor and Pylyshyn [38]. They claimed that thinking and reasoning as a phenomena is inherently rule-based (symbolic, relational), and this was not so much a natural mental faculty but a complex ability that evolved as a tool for preserving truth and (to a lesser extent) predicting future events. They pose it as a challenge to connectionism: if connectionism will be able to reason, the only way it will be able to do so (since reasoning is inherently rule-based) is by making an artificial neural network which produces a system of rules. This would not be ‘connectionist reasoning’ but symbolic reasoning whose symbols are assigned meaningful things thanks to artificial neural networks. Artificial neural networks fill in the content, but the reasoning itself is still symbolic.

You might notice that the validity of this argument rests on the idea that thinking is inherently rule-based, so the most easy way to overcome their challenge it is to dispute this initial assumption. If thinking and reasoning would not be completely rule-based, it would mean that they have aspects that are processed ‘intuitively’, and not derived by rules. Connectionists have made an incremental but important step in bridging the divide. Consider the following reasoning: ‘it is too long for a walk, I better take my van’, ‘I forgot that my van is at the mechanic, I better take my wife’s car’. Notice that we have deliberately not framed this as a classic syllogism, but in a form similar to the way someone would actually think and reason.²¹ Notice that what makes this thinking valid,²² is the possibility of equating ‘car’ with ‘van’ as similar.²³ Word2vec [39] is a neural language model which learns numerical vectors for a given word and a context (several words around it), and this is learned from texts. The choice of texts is the ‘big picture’. A great feature of word2vec is that it clusters words by semantic similarity in the big picture. This is possible since semantically similar words share a similar immediate context: both Bob and Alice can be hungry, but neither can Plato nor the number 4. But substituting similar for similar is just proto-inference, the major incremental advance towards connectionist reasoning made possible by word2vec is the native calculations it enables. Suppose that $v(x)$ is the function which maps x (which is a string) to its learned vector. Once trained, the word vectors word2vec generates are special in the sense that one can calculate with them like $v(\text{king}) - v(\text{man}) + v(\text{woman}) \approx v(\text{queen})$. This is called *analogical reasoning* or *word analogies*, and it is the first major landmark in developing a purely connectionist approach to reasoning.

We will be exploring reasoning in the final chapter of the book in the context of question answering. We will be exploring also energy-based models and memory models, and the best current take on the issue of reasoning is with memory-based

²¹Plato defined *thinking* (in his *Sophist*) as the soul’s conversation with itself, and this is what we want to model, whereas the rule-based approach was championed by Aristotle in his *Organon*. More succinctly, we are trying to reframe reasoning in platonic terms instead of using the dominant Aristotelian paradigm.

²²At this point, we deliberately avoid talking of ‘valid inference’ and use the term ‘valid thinking’.

²³Note that this interchangeability dependent on the big picture. If I need to move a piano, I could not do it with a car, but if I need to fetch groceries, I can do it with either the car or the van.

models. This is perhaps surprising since in the normal cognitive setting (undoubtedly under the influence of GOFAI), we consider memory (knowledge) and reasoning as two rather distinct aspects, but it seems that neural networks and connectionism do not share this dichotomy.

References

1. A.M. Turing, On computable numbers, with an application to the entscheidungsproblem. *Proc. Lond. Math. Soc.* **42**(2), 230–265 (1936)
2. V. Peckhaus, Leibniz's influence on 19th century logic. in *The Stanford Encyclopedia of Philosophy*, ed. by E.N. Zalta (2014)
3. J.S. Mill, *A System of Logic Ratiocinative and Inductive: Being a connected view of the Principles of Evidence and the Methods of Scientific Investigation* (1843)
4. G. Boole, *An Investigation of the Laws of Thought* (1854)
5. A.M. Turing, Computing machinery and intelligence. *Mind* **59**(236), 433–460 (1950)
6. R. Carnap, *Logical Syntax of Language* (Open Court Publishing, 1937)
7. A.N. Whitehead, B. Russell, *Principia Mathematica* (Cambridge University Press, Cambridge, 1913)
8. J.Y. Lettvin, H.R. Maturana, W.S. McCulloch, W.H. Pitts, What the frog's eye tells the frog's brain. *Proc. IRE* **47**(11), 1940–1959 (1959)
9. N.R. Smalheiser, Walter pitts. *Perspect. Biol. Med.* **43**(1), 217–226 (2000)
10. A. Gefter, The man who tried to redeem the world with logic. *Nautilus* **21** (2015)
11. F. Rosenblatt, *Principles of Neurodynamics: perceptrons and the theory of brain mechanisms* (Spartan Books, Washington, 1962)
12. F. Rosenblatt, Recent work on theoretical models of biological memory, in *Computer and Information Sciences II*, ed. by J.T. Tou (Academic Press, 1967)
13. S. Russell, P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd edn. (Pearsons, London, 2010)
14. H. Moravec, *Mind Children: The Future of Robot and Human Intelligence* (Harvard University Press, Cambridge, 1988)
15. M. Minsky, S. Papert, *Perceptrons: An Introduction to Computational Geometry* (MIT Press, Cambridge, 1969)
16. L.R. Graham, *Science in Russia and the Soviet Union. A Short History* (Cambridge University Press, Cambridge, 2004)
17. S. Pinker, *The Blank Slate* (Penguin, London, 2003)
18. B.F. Skinner, *The Possibility of a Science of Human Behavior* (The Free House, New York, 1953)
19. E.L. Gettier, Is justified true belief knowledge? *Analysis* **23**, 121–123 (1963)
20. T.S. Kuhn, *The Structure of Scientific Revolutions* (University of Chicago Press, Chicago, 1962)
21. N. Chomsky, *Aspects of the Theory of Syntax* (MIT Press, Cambridge, 1965)
22. N. Chomsky, A review of B. F. Skinner's verbal behavior. *Language* **35**(1), 26–58 (1959)
23. A. Newell, J.C. Shaw, H.A. Simon, Elements of a theory of human problem solving. *Psychol. Rev.* **65**(3), 151–166 (1958)
24. J. Lighthill, Artificial intelligence: a general survey, in *Artificial Intelligence: A Paper Symposium*, Science Research Council (1973)
25. Paul J. Werbos, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences* (Harvard University, Cambridge, 1975)

A set does not remember the order of elements or repetitions of one element. If we have a set that remembers repetitions but not order we have multisets or *bags*, so we have $\{1, 0, 1\} = \{1, 1, 0\}$ but neither is equal to $\{1, 0\}$, we are talking about multisets. The usual way to denote bags to distinguish them from sets is to number the elements, so instead of writing $\{1, 1, 1, 1, 0, 1, 0, 0\}$ we would write $\{"1" : 5, "0" : 3\}$. Bags will be very useful to model language via the so-called *bag of words* model as we will see in Chap. 3.

If we care both about the position and repetitions, we write $(1, 0, 0, 1, 1)$. This object is called a *vector*. If we have a vector of variables like (x_1, x_2, \dots, x_n) we write it as \mathbf{x} or \mathbf{x} . The individual x_i , $1 \leq i \leq n$, is called a *component* (in sets they used to be called members), and the number of components is called *the dimensionality of the vector \mathbf{x}* .

The terms *tuple* and *list* are very similar to vectors. Vectors are mainly used in theoretical discussions, whereas tuples and lists are used in realizing vectors in programming code. As such, tuples and lists are always named with programming variables such as `myList` or `vectorAsTuple`. So an example of either tuple or list would be `newThing := (11, 22, 33)`. The difference between tuple and a list is that lists are *mutable* and tuples are not. Mutability of a structure means that we can assign a new value to a member of that structure. For example, if we have `newThing := (11, 22, 33)` and then we do `newThing[1] ← 99` (to be read ‘assign to the *second*² item the value of 99’), we get `newThing := (11, 99, 33)`. This means that we have mutated the list. If we do not want to be able to do that, we use a tuple, in which case we cannot modify the elements. We can create a new tuple `newerThing` such that `newerThing[0] ← newThing[0]`, `newerThing[1] ← 99` and `newerThing[2] ← newThing[2]` but this is not *changing* the values, just copying it and composing a new tuple. Of course, if we have an unknown data structure, we can check whether it is a list or tuple by trying to modify some component. Sometimes, we might wish to model vectors as tuples, but we will usually want to model them as lists in our programming codes.

Now we have to turn our attention to functions. We will take a computational approach in their definition.³ A function is a magical artifact that takes arguments (inputs) and turns them into values (outputs). Of course, the trick with functions is that instead of using magic we must define in them how to get from inputs to outputs, or in other words how to transform the inputs into outputs. Recall a function, e.g. $y = 4x^3 + 18$ or equivalently $f(x) = 4x^3 + 18$, where x is the input, y is the output and f is the function’s ‘name’. The output y is defined to be the application of f to x , i.e. $y := f(x)$. We are omitting a few things here, but they are not important for this book, but we point the interested reader to [1].

When we think of a function like this, we actually have an instruction (algorithm) of how to transform the x to get the y , by using simpler functions such as addition,

²The counting starts with 0, and we will use this convention in the whole book.

³The traditional definition uses sets to define tuples, tuples to define relations and relations to define functions, but that is an overly logical approach for our needs in the present volume. This definition provides a much wider class of entities to be considered functions.

multiplication and exponentiation. They in turn can be expressed from simpler functions, but we will not need the proofs for this book. The reader can find in [2] the details on how this can be done.

Note that if we have a function with 2 arguments⁴ $f(x, y) = x^y$ and pass in values (2, 3) we get 8. If we pass in (3, 2) we will get 9, which means that functions are order sensitive, i.e. they operate on vector inputs. This means that we can generalize and say that a function always takes a vector as an input, and a function taking an n -dimensional vector is called an n -ary function. This means that we are free to use the notation $f(\mathbf{x})$. A 0-ary function is a function that produces an output but takes in no input. Such a function is called a *constant*, e.g. $p() = 3.14159\dots$ (notice the notation with the open and closed parenthesis).

Note that we can take a function's argument input vector and add to it the output, so that we have $(x_1, x_2, \dots, x_n, y)$. This structure is called a *graph* of the function f for inputs \mathbf{x} . We will see how we can extend this to all inputs. A function can have parameters and the function $f(x) = ax + b$ has a and b as parameters. They are considered fixed, but we might want to tweak them to get a better version of the function. Note that a function always gives the same result if it is given the same input and you do not change the parameters. By changing the parameters, you can drastically change the output. This is very important for deep learning, since deep learning is a method for automatically tuning parameters which in turn modifies the output.

We can have a set A and we may wish to create a function of x which gives a value 1 to all values which are members of A and 0 to all other values for x . Since this function is different for all sets A , other than this, it always does the same thing, we can give it a name which includes A . We choose the name $\mathbb{1}_A$. This function is called *indicator function* or *characteristic function*, and it is sometimes denoted as χ_A in the literature. This is used for something which we will call *one-hot encoding* in the next chapter.

If we have a function $y = ax$, then the set from which we take the inputs is called the domain of the function, and the set to which the outputs belong is called the codomain of the function. In general, a function does not need to be defined for all members of the domain, and, if it is, it is called a *total* function. All functions that are not total are called *partial*. Remember that a function assigns to every vector of inputs always the same output (provided the parameters do not change). If by doing so the function 'exhausts' the whole codomain, i.e. after assignment there are no members of the codomain which are not outputs of some inputs, the function is called a *surjection*. If on the other hand the function never assigns to different input vectors the same output, it is called an *injection*. If it is both an injection and surjection, it is called a bijection. The set of outputs B given a set of inputs A is called an *image* and denoted by $f[A] = B$. If we look for a set of inputs A given the set of outputs B , we are looking at its inverse image denoted by $f^{-1}[B] = A$ (we can use the same notation for individual elements $f^{-1}(b) = a$).

⁴A function with n -arguments is called an *n-ary function*.

A function f is called *monotone* if for every x and y from the domain (for which the function is defined) the following holds: if $x < y$ then $f(x) \leq f(y)$ or if $x > y$ then $f(x) \geq f(y)$. Depending on the direction, this is called an *increasing* or *decreasing* function, and if we have $<$ instead of \leq , it is called *strictly increasing* (or strictly decreasing). A *continuous function* is a function that does not have gaps. For what we will be needing now, this definition is good enough—we are imprecise, but we are sacrificing precision for clearness. We will be returning to this later.

One interesting function is the characteristic function for rational numbers over all real numbers. This function returns 1 if and only if the real number it picked is also a rational number. This function is continuous nowhere. A different function which is continuous in parts but not everywhere is the so-called *step function* (we will mention it again briefly in Chap. 4):

$$\text{step}_0(x) = \begin{cases} 1, & x > 0 \\ -1, & x \leq 0 \end{cases}$$

Note that step_0 can be easily generalized to step_n by simply placing n instead of 0. Also, note that the 1 and -1 are entirely arbitrary, so we can put any values instead. A step function that takes in an n -dimensional vector is also sometimes called a *voting function*, but we will keep calling it a step function. In this version, all components of the input vector of the function are added before being compared with the threshold n (the threshold n is called a *bias* in neural network literature). Pay close attention to how we defined the step function with two cases: if a function is defined by cases, it is an important *hint* that the function might not be continuous. It is not always the case (in either way we look at it), but it is a good hint to follow and it is often true.⁵

Before continuing to derivations, we will be needing a few more concepts. If the outputs of the function f approach a value c (and settle in it), we say that the function *converges in c* . If there is no such value, the function is called *divergent*. In most mathematics textbooks, the definitions of convergence are more meticulous, but we will not be needing the additional mathematical finesse in this book, just the general intuition.

An important constant we will use is the *Euler number*, $e = 2.718281828459 \dots$. This is a constant and we will reserve for it the letter e . We will be using the basic numerical operations extensively, and we give a brief overview of their behaviour and notations used here:

- The reciprocal number of x is $\frac{1}{x}$ or equivalently x^{-1}
- The square root of x is $x^{\frac{1}{2}}$ or equivalently \sqrt{x}
- The exponential function has the properties: $x^0 = 1$, $x^1 = x$, $x^n \cdot x^m = x^{n+m}$, $(x^n)^m = x^{n \cdot m}$

⁵The ReLU or *rectified linear unit* defined by $\rho(x) = \max(x, 0)$ is an example of a function that is continuous even though it is (usually) defined by cases. We will be using ReLU extensively from Chap. 6 onwards.

- The logarithmic function has the properties: $\log_c 1 = 0$, $\log_c c = 1$, $\log_c(xy) = \log_c x + \log_c y$, $\log_c(\frac{x}{y}) = \log_c x - \log_c y$, $\log_c x^y = y \log_c x$, $\log_x y = \frac{\log_c y}{\log_c x}$, $\log_x x^y = y$, $x^{\log_x y} = y$, $\ln x := \log_e x$.

The last concept we will need before continuing to derivations is the concept of a *limit*. An intuitive definition would be that the limit of a function is a value which the outputs of the function approach but never reach.⁶ The trick is that the limit of the function is considered in relation to a change in inputs and it must be a concrete value, i.e. if the limit is ∞ or $-\infty$, we do not call it a limit. Note that this means that for the limit to exist it must be a *finite* value. For example, $\lim_{x \rightarrow 5} f(x) = 10$, if we take f to be $f(x) = 2x$. It is of vital importance not to confuse the number 5 which the inputs approach and the limit, 10, which the outputs of the function approach as the inputs approach 5.

The concept of limit is trivial (and mathematically weird) if we think of integer inputs. We shall assume when we think of limits that we are considering real numbers as inputs (where the idea of continuity makes sense). Therefore, when talking about limits (and derivations), the input vectors are real numbers and we want the function to be continuous (but sometimes it might not be). If we want to know a limit of a function, and it is continuous everywhere, we can try to plug in the value to which the inputs approach and see what we get for the output. If there are problems with this, we can either try to simplify the function expression or see what is happening to the pieces. In practice,⁷ the problems occur in two way: (i) the function is defined by cases or (ii) there are segments where the outputs are undefined due to a hidden division by 0 for some inputs.

We can now replace our intuitive idea of continuity with a more rigorous definition. We call a function f continuous in a point $x = a$ if and only if the following conditions hold:

1. $f(a)$ is defined
2. $\lim_{x \rightarrow a} f(x)$ exists
3. $f(a) = \lim_{x \rightarrow a} f(x)$.

A function is called continuous everywhere if and only if it is continuous in all points. Note that all elementary functions are continuous everywhere⁸ and so are all

⁶This is why $0.999 \dots \neq 1$.

⁷This is especially true in programming, since when we program we need to approximate functions with real numbers by using functions with rational numbers. This approximation also goes a long way in terms of intuition, so it is good to think about this when trying to figure out how a function will behave.

⁸With the exception of division where the divisor is 0. In this case, the division function is undefined, and therefore the notion of continuity does not have any meaning in this point.

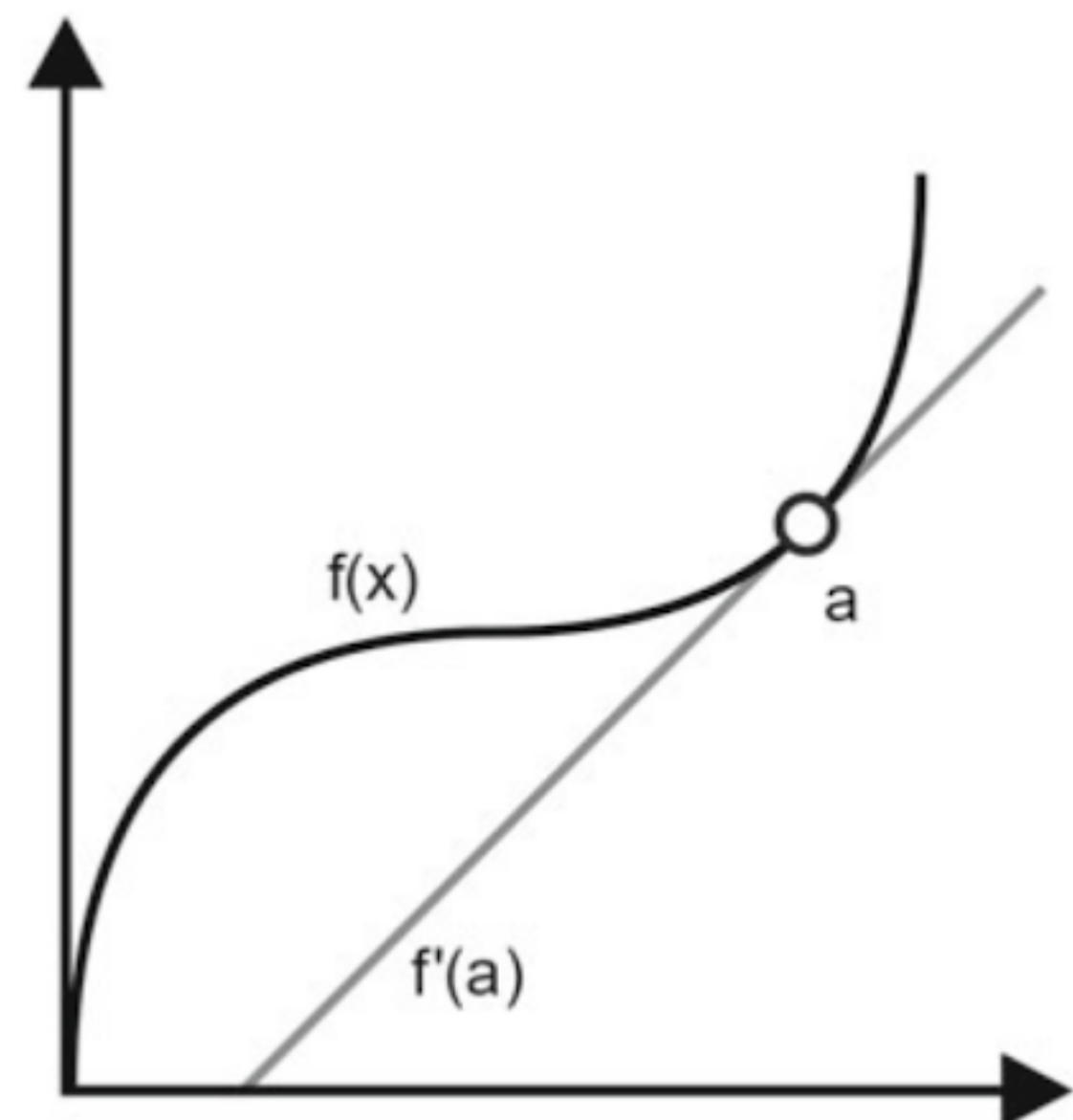
polynomial functions. Rational functions⁹ are continuous everywhere except where the value of the denominator is 0. Some equalities that hold for limits are

1. $\lim_{x \rightarrow a} c = c$
2. $\lim_{x \rightarrow 0^+} \frac{1}{x} = \infty$
3. $\lim_{x \rightarrow 0^-} \frac{1}{x} = -\infty$
4. $\lim_{x \rightarrow \infty} \frac{1}{x} = 0$
5. $\lim_{x \rightarrow \infty} \left(1 + \frac{1}{x}\right)^x = e.$

Now, we are all set to continue our journey to differentiation.¹⁰ We can develop a bit of intuition behind derivatives by noting that the derivative of a function can be imagined as the slope of the plot of that function in a given point. You can see an illustration in Fig. 2.1. If a function $f(x)$ (the domain is X) has a derivative in every point $a \in X$, then there exists a new function $g(x)$ which maps all values from X to its derivative. This function is called the *derivative* of f . As $g(x)$ depends on f and x , we introduce the notation $f'(x)$ (Lagrange notation) or, remembering that $f(x) = y$, we can use the notation $\frac{dy}{dx}$ or $\frac{df}{dx}$ (Leibniz notation). We will deliberately use these two notations inconsistently in this book, since some ideas are more intuitive when expressed in one notation, while some are more intuitive in the other. And we want to focus on the underlying mathematical phenomena, not the notational tidiness.

Let us address this in more detail. Suppose we have a function $f(x) = \frac{x}{2}$. The slope of this function can be obtained by selecting two points from it, e.g. $t_1 = (x_1, y_1)$ and $t_2 = (x_2, y_2)$. Without loss of generality, we can assume that t_1 comes before t_2 ,

Fig. 2.1 The derivative of $f(x)$ in the point a



⁹Rational functions are of the form $\frac{f(x)}{g(x)}$ where f and g are polynomial functions.

¹⁰The process of finding derivatives is called ‘differentiation’.