1

Introduction

The term *neural networks* historically refers to networks of neurons in the mammalian brain. Neurons are its fundamental units of computation, and they are connected together in networks to process data. This can be a very complex task. The dynamics of such neural networks in response to external stimuli is therefore often quite intricate. Inputs and outputs of each neuron vary as functions of time in the form of spike trains, but the network itself also changes over time: we learn and improve our data-processing capabilities by establishing new connections between neurons.

Neural-network algorithms for machine learning are inspired by the architecture and the dynamics of networks of neurons in the brain. The algorithms use highly idealised neuron models. Nevertheless, the fundamental principle is the same: artificial neural networks learn by changing the connections between their neurons. Such networks can perform a multitude of information-processing tasks.

Neural networks can for instance learn to recognise structures in a data set and, to some extent, generalise what they have learnt. A *training set* contains a list of input patterns together with a list of corresponding labels, or target values, that encode the properties of the input patterns the network is supposed to learn. Artificial neural networks can be trained to classify such data very accurately by adjusting the connection strengths between their neurons, and can learn to generalise the result to other data sets – provided that the new data is not too different from the training data. A prime example for a problem of this type is object recognition in images, for instance in the sequence of camera images taken by a self-driving car. Recent interest in machine learning with neural networks is driven in part by the success of neural networks in *visual object recognition*.

Another task at which neural networks excel is *machine translation* with dynamical or recurrent networks. Such networks take sentences as inputs. As one feeds word after word, the network outputs the words in the translated sentence. Recurrent networks can be efficiently trained on large training sets of input sentences

https://doi.org/10.1017/9781108860604.001 Published online by Cambridge University Press

and their translations. Google translate works in this way. Recurrent networks have also been used with considerable success to predict chaotic dynamics. These are all examples of *supervised* learning, where the networks are trained to associate a certain target, or label, with each input.

Artificial neural networks are also good at analysing large sets of unlabelled, often high-dimensional data – where it may be difficult to determine a priori which questions are most relevant and rewarding to ask. *Unsupervised*-learning algorithms organise the unlabelled input data in different ways. They can, for instance, detect familiarity and similarity (clusters) of input patterns, or other structures in the input data. Unsupervised-learning algorithms work well when there is redundancy in the input data, and they are particularly useful for large, high-dimensional data sets, where it may be a challenge to detect clusters or other data structures by inspection.

Many problems lie between these two extremes of supervised and unsupervised learning. Consider how an agent may learn to navigate a complex environment, in order to get from one location to another as quickly as possible, or expending as little energy as possible. The method of *reinforcement learning* allows the agent to do just that, by optimising its behaviour in response to environmental cues in the shape of penalties and rewards. In short, the agent learns to act in such a way that it receives positive feedback (reward) more often than a penalty.

The tools for machine learning with neural networks were developed long ago, most of them during the second half of the last century. In 1943, McCulloch and Pitts [6] analysed how networks of neurons can process information. Using an abstract model for a neuron, they demonstrated how such units can be coupled together to represent logical functions (Figure 1.1). Their analysis and conclusions are formulated using Carnap's logical syntax [7], not in terms of algebraic equations as we are used to today. Nevertheless, their neuron model is essentially the binary threshold unit, closely related to the fundamental building block of most neural-network algorithms for machine learning to this date. In this book, we therefore refer to this model as the *McCulloch-Pitts neuron*. The purpose of this early

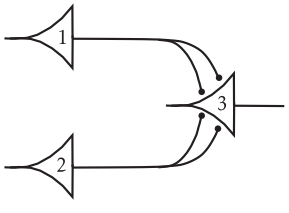

Figure 1.1 Logical OR function represented by three neurons. Neuron 3 fires actively if at least one of the neurons 1 and 2 is active. After Figure 1b by McCulloch and Pitts [6]

https://doi.org/10.1017/9781108860604.001 Published online by Cambridge University Press

research on neural networks was to explain neuro-physiological mechanisms [8]. Perhaps the most significant advance was Hebb's learning principle, describing how neural networks learn by strengthening connections between neurons that are active simultaneously. The principle is described in Hebb's book *The Organization of Behavior: A Neuropsychological Theory* [9], published in 1949.

About 10 years later, research in artificial neural networks had intensified, sparked by the influential work of Rosenblatt. In 1958, he formulated a learning rule for the McCulloch-Pitts neuron, related to Hebb's rule, and demonstrated that the rule converges to the correct solution for all problems this model can solve [10]. He coined the term *perceptron* for layered networks of McCulloch-Pitts neurons and showed that such neural networks could in principle solve tasks that a single McCulloch-Pitts neuron could not. However, there was no general learning rule for perceptrons at the time. The work of Minsky and Papert [11] emphasised the geometric aspects of learning. This allowed them to prove which kinds of problems perceptrons could solve, and which not. In 1969, they summarised these results in their elegant book *Perceptrons: An Introduction to Computational Geometry*.

Perceptrons are classifiers that output a label for each input pattern. A perceptron represents a mathematical function, an input-output mapping. A breakthrough in perceptron learning was the paper by Rumelhart et al. [12]. The authors demonstrated in 1986 that perceptrons can be trained by gradient descent. This means that the connection strengths between the neurons are iteratively changed in small steps, to eventually minimise the output error. For a single McCulloch-Pitts neuron, this gives essentially Hebb's rule. The important point is that gradient descent allows one to efficiently train perceptrons with many layers (*backpropagation* for multilayer perceptrons). A second contribution of Rumelhart et al. is the idea to use local feature maps for object recognition with neural networks. The corresponding mathematical operation is a convolution. Therefore such architectures are now known as *convolutional networks*.

The work of Hopfield popularised an entirely different approach, also based on Hebb's rule. In 1982, Hopfield analysed the properties of a dynamical, or recurrent, network that functions as a memory [13]: the dynamics is designed to find stored patterns by converging to a corresponding steady state. Such *Hopfield networks* were especially popular amongst physicists because there are close connections to the statistical physics of spin glasses that made it possible to derive a precise mathematical understanding of such artificial neural networks. Hopfield networks are the basis for important developments in computer science. More general recurrent networks, for example, are trained like perceptrons for language processing. Hopfield networks with hidden neurons, so-called Boltzmann machines [14], are generative models that allow one to sample from a distribution the neural network learned. The training algorithm for Boltzmann machines with many hidden

layers [15], published in 1986, is one of the first algorithms for training networks with many layers, so-called *deep* networks.

An important problem in behavioural psychology is to understand how we learn from experience. One hypothesis is that desirable behaviours are reinforced by positive feedback. Around the same time as researchers began to analyse perceptrons, a different line of neural-network research emerged: to find learning rules that allow neural networks to learn by reinforcement. In many problems of this kind, the positive feedback or reward is not immediate but is received at some time in the future, as in a board game, for example. Therefore it is necessary to understand how to estimate the future reward for a certain behaviour, and how to find strategies that optimise the future reward. *Reinforcement* learning [16] is designed for this purpose. In 1995, an early application of this method demonstrated how a neural network could learn to play the board game backgammon [17].

A related research field originated from the neuro-physiological question: how do we learn to map visual or sensory stimuli to spatio-temporal patterns of neural activity in the brain? In 1992, Kohonen described a *self-organising map* [18] that suggests how neurons might create meaningful geometric representations of inputs. At the same time, Kohonen's algorithm is one of the first methods for non-linear dimensionality reduction for large data sets.

There are many connections between neural-network algorithms for machine learning and methods used in mathematical statistics, such as for instance Markov-chain Monte-Carlo algorithms and simulated-annealing methods. Certain unsupervised learning algorithms are related to principal-component analysis, others to clustering algorithms such as K-means clustering. Supervised learning with deep networks is essentially regression analysis, trying to fit an input-output function to the training data. In other words, this is just function fitting – and usually with a very large number of fitting parameters. Recent convolutional neural networks have millions of parameters. To determine so many parameters requires very large and accurate data sets. This makes it clear that neural networks are not a *solution for everything*. One of the difficult problems is to understand when machine learning with neural networks is called for and when it is not. Therefore we need a detailed understanding of how the algorithms work, and in particular when and how they fail.

There were some early successes of machine learning with neural networks, but these methods were not widely used in the last century. During the past decade, by contrast, machine learning with neural networks has become increasingly successful and popular. For many applications, neural-network-based algorithms are now regarded as the method of choice, for example for predicting how proteins fold [19]. What caused this paradigm shift? After all, the methods are essentially those developed forty or more years ago. A reason for the new success is perhaps

that industry, in acute need of machine-learning algorithms for large data sets, has invested time and effort into generating larger and more accurate training sets than previously available. Computer hardware has improved too, so that networks with many layers containing many neurons can now be efficiently trained, making the recent progress possible.

This book is based on notes for lectures on artificial neural networks I have given for more than 15 years at Gothenburg University and Chalmers University of Technology in Gothenburg, Sweden. When I prepared these lectures, my primary source was *Introduction to the Theory of Neural Computation* by Hertz, Krogh, and Palmer [1]. The material is organised into three parts: Hopfield networks, supervised learning of labelled data, and learning for unlabelled data sets (unsupervised and reinforcement learning). One reason for starting with Hopfield networks is that there is an elegant mathematical theory that describes how these neural networks learn, making it possible to understand their strengths and weaknesses from first principles. This is not the case for most of the other algorithms discussed in this book. The analysis of Hopfield networks sets the scene for the later parts of the book. Part II describes supervised learning with multilayer perceptrons and convolutional neural networks, starting from the simple geometrical picture emphasised by Minsky and Papert, and leading to the recent successes of convolutional networks in object recognition and recurrent networks in language processing. Part III explains what neural networks can learn about data that is not labelled, with particular emphasis on reinforcement learning. The overall goal is to explain the fundamental principles that allow neural networks to learn, emphasising ideas and concepts that are common to all three parts.

1.1 Neural Networks

Different regions in the mammalian brain perform different tasks. The *cerebral cortex* is the outer part of the mammalian brain, one of its largest and best developed segments. We can think of the cerebral cortex as a thin sheet (about 2 to 5 mm thick) that folds upon itself to form a layered structure with a large surface area that can accommodate large numbers of nerve cells, *neurons*. The human cerebral cortex contains about 10^{10} neurons. They are linked together by nerve strands (*axons*) that branch and end in *synapses*. These synapses are the connections to other neurons. The synapses connect to *dendrites*, branches extending from the neural cell body that are designed to collect input from other neurons in the form of electrical signals. A neuron in the human brain may have thousands of synaptic connections with other neurons. The resulting network of connected neurons in the cerebral cortex is responsible for processing visual, audio, and sensory data.

https://doi.org/10.1017/9781108860604.001 Published online by Cambridge University Press

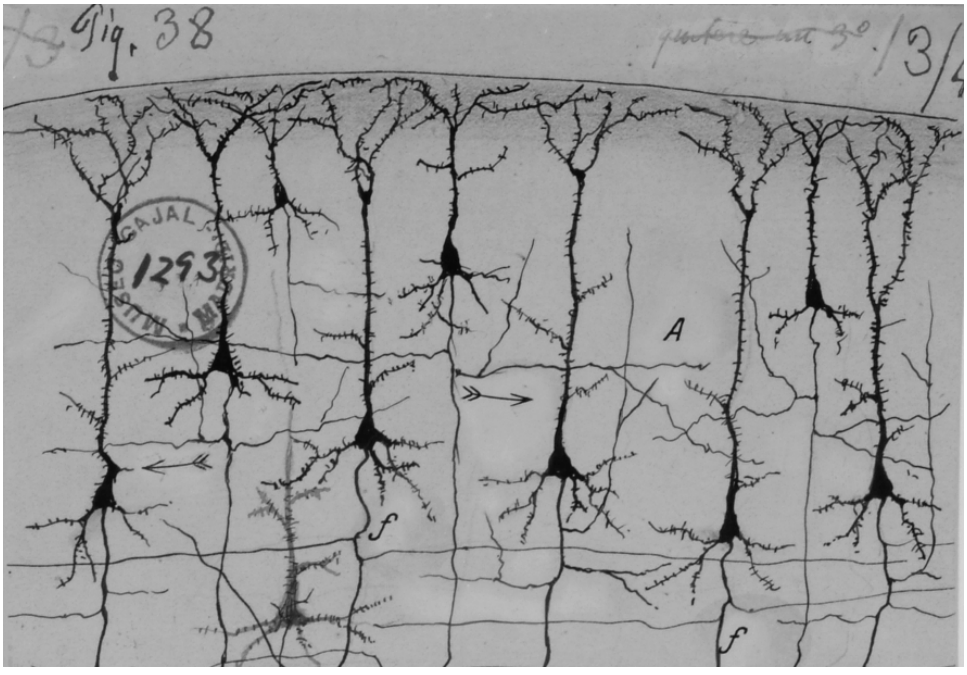

Figure 1.2 Neurons in the *cerebral cortex*, a part of the mammalian brain. Drawing by Santiago Ramón y Cajal, the Spanish neuroscientist who received the Nobel Prize in Physiology and Medicine in 1906 together with Camillo Golgi 'in recognition of their work on the structure of the nervous system' [20]. Courtesy of the Cajal Institute, 'Cajal Legacy', Spanish National Research Council (CSIC), Madrid, Spain. Original in colour

Figure 1.2 shows neurons in the cerebral cortex. This drawing was made by Santiago Ramón y Cajal more than 100 years ago. By microscope he studied the structure of neural networks in the brain and documented his observations by ink-on-paper drawings like the one reproduced in Figure 1.2. One can distinguish the cell bodies of the neural cells, their axons (f), and their dendrites. The axons of some neurons connect to the dendrites of other neurons, forming a neural network (see Ref. [21] for a slightly more detailed description of this drawing).

A schematic image of a neuron is drawn in Figure 1.3. Information is processed from left to right. On the left are the dendrites that receive signals and connect to the cell body of the neuron where the signal is processed. The right part of the figure shows the axon, through which the output is sent to other neurons. The axon connects to their dendrites via synapses.

Information is transmitted as an electrical signal. Figure 1.4 shows an example of the time series of the electric potential for a *pyramidal* neuron in fish [22]. The time

https://doi.org/10.1017/9781108860604.001 Published online by Cambridge University Press

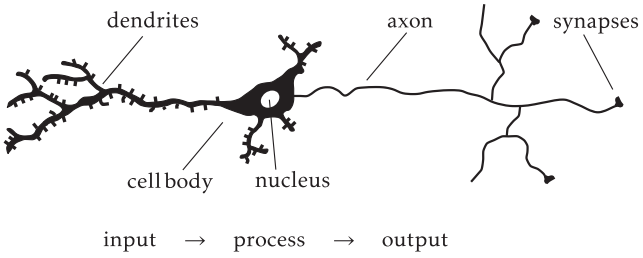

Figure 1.3 Schematic image of a neuron. Dendrites receive input in the form of electrical signals, via synapses. The signals are processed in the cell body of the neuron. The cell nucleus is shown as a white blob. The output travels from the neural cell body along the axon which connects via synapses to other neurons

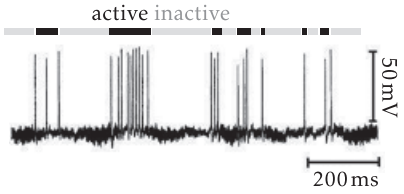

Figure 1.4 Spike train in electro-sensory pyramidal neuron of a fish. The time series is from Ref. [22]. It is reproduced by permission of the publisher. The labels were added

series consists of an intermittent series of electrical-potential spikes. Quiescent periods without spikes occur when the neuron is *inactive*, during spike-rich periods we say that the neuron is *active*.

1.2 McCulloch-Pitts Neurons

In artificial neural networks, the ways in which information is processed and signals are transferred are highly idealised. McCulloch and Pitts [6] modelled the neuron, the computational unit of the neural network, as a *binary threshold unit*. It has only two possible outputs, or *states*: active or inactive. To compute the output, the unit sums the weighted inputs. If the sum exceeds a given threshold, the state of the neuron is said to be active, otherwise inactive. A slightly more general model than the original one is illustrated in Figure 1.5. The model performs repeated computations in discrete time steps $t = 0, 1, 2, 3, \ldots$. The state of neuron number j at time step t is denoted by

$$s_j(t) = \begin{cases} -1 & \text{inactive,} \\ 1 & \text{active.} \end{cases} \tag{1.1}$$

https://doi.org/10.1017/9781108860604.001 Published online by Cambridge University Press

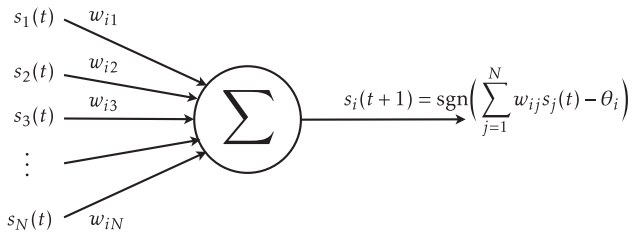

Figure 1.5 Schematic diagram of a McCulloch-Pitts neuron. The index of the neuron is i; it receives inputs from N neurons. The strength of the connection from neuron j to neuron i is denoted by w_{ij}. The *threshold* value for the neuron is denoted by θ_i. The index $t = 0, 1, 2, 3, \ldots$ labels the discrete time sequence of computation steps, and $\mathrm{sgn}(b)$ stands for the signum function [Figure 1.6 and Equation (1.3)]

Given the states $s_j(t)$, neuron number i computes

$$s_i(t+1) = \mathrm{sgn}\left(\sum_{j=1}^{N} w_{ij}s_j(t) - \theta_i\right) \equiv \mathrm{sgn}[b_i(t)]. \tag{1.2}$$

Here $\mathrm{sgn}(b)$ is the signum function (Figure 1.5):

$$\mathrm{sgn}(b) = \begin{cases} -1, & b < 0, \\ +1, & b \geq 0. \end{cases} \tag{1.3}$$

The argument of the signum function,

$$b_i(t) = \sum_{j=1}^{N} w_{ij}s_j(t) - \theta_i, \tag{1.4}$$

is called the *local field*. We see that the neuron performs a weighted average of the inputs $s_j(t)$. The parameters w_{ij} are called *weights*. Here the first index, i, refers to the neuron that does the computation, and j labels the neurons that connect to neuron i. In general weights between different pairs of neurons assume different numerical values, reflecting different strengths of the synaptic couplings. Weights can be positive or negative, and we say that there is no connection when $w_{ij} = 0$.

In this book, we refer to the model described in Figure 1.5 as the *McCulloch-Pitts neuron*, although their original model had additional constraints on the weights. The threshold[1] for neuron i is denoted by θ_i.

Finally, note that the computation (1.2) is performed for all neurons i in parallel, given the states $s_j(t)$ at time step t. The outputs $s_i(t+1)$ are the inputs to all neurons

[1] In the *deep-learning* literature [4], the thresholds are called *biases*, defined as the negative of θ_i, with a plus sign in Equation (1.4). In this book, we use the convention (1.4), with the minus sign.

https://doi.org/10.1017/9781108860604.001 Published online by Cambridge University Press

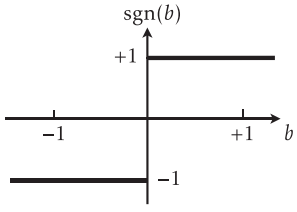

Figure 1.6 Signum function [Equation (1.3)]

at the next time step. Therefore the outputs have the time argument $t + 1$. These steps are repeated many times, resulting in time series of the activity levels of all neurons in the network.

1.3 Activation Functions

The McCulloch-Pitts model approximates the patterns of spiking activity in Figure 1.4 in terms of two states, -1 and $+1$, representing the inactive and active periods shown in the figure. For many computation tasks this is sufficient, and for our purposes it does not matter that the dynamics of electrical signals in the cortex is quite different in detail. The aim after all is not to model the neural dynamics in the brain but to construct computation models inspired by these dynamics.

It will become apparent later that the simplest model described above must be generalised somewhat for certain tasks and questions. For example, the jump in the signum function at $b = 0$ may cause large fluctuations in the activity levels of a network of neurons, caused by infinitesimal changes of the local fields across $b = 0$. To dampen this effect, one allows the neuron to respond continuously to its inputs, replacing Equation (1.2) by

$$s_i(t+1) = g\left(\sum_j w_{ij} s_j(t) - \theta_i\right). \tag{1.5}$$

Here $g(b)$ is a continuous *activation function*. It could just be a linear function, $g(b) \propto b$. But we shall see that many tasks require non-linear activation functions, such as $\tanh(b)$ (Figure 1.7). When the activation function is continuous, the neuron states assume continuous values too, not just the discrete values -1 and $+1$ given in Equation (1.1).

Alternatively, one may use a piecewise linear activation function (Figure 1.8). This is motivated in part by the response curve of the *leaky integrate-and-fire* neuron, a model for the relation between the electrical current I through the cell membrane into the neuron cell, and the membrane potential U. The simplest model for the dynamics of the membrane potential represents the neuron as a capacitor. In the

https://doi.org/10.1017/9781108860604.001 Published online by Cambridge University Press

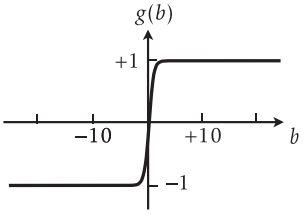

Figure 1.7 Continuous activation function $g(b) = \tanh(b)$

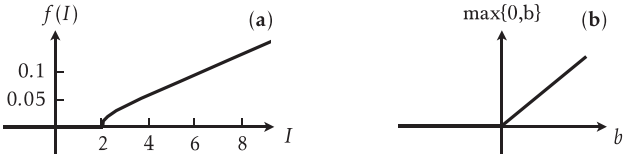

Figure 1.8 (**a**) Firing rate $f(I)$ of a *leaky integrate-and-fire* neuron as a function of the electrical current I through the cell membrane, Equation (1.7) for $\tau = 25$ and $U_c/R = 2$. (**b**) Piecewise linear activation function, $g(b) = \max\{0, b\}$

leaky integrate-and-fire neuron, leakage is modelled by adding a resistor R in parallel with the capacitor C, so that

$$I = \frac{U}{R} + C\frac{\mathrm{d}U}{\mathrm{d}t}\,. \tag{1.6}$$

For a constant current, the membrane potential grows from zero as a function of time, $U(t) = RI[1-\exp(-t/\tau)]$, where $\tau = RC$ is the time constant of the model. One says that the neuron produces a *spike* when the membrane potential exceeds a critical value, U_c. Immediately after, the membrane potential is set to zero (and begins to grow again). In this model, the *firing rate* $f(I)$ is thus given by t_c^{-1}, where t_c is the solution of $U(t) = U_c$. It follows that the firing rate exhibits a threshold behaviour. In other words, the system works like a rectifier:

$$f(I) = \begin{cases} 0 & \text{for} \quad I \leq U_c/R, \\ \left[\tau \log\left(\frac{RI}{RI-U_c}\right)\right]^{-1} & \text{for} \quad I > U_c/R. \end{cases} \tag{1.7}$$

This response curve is illustrated in Figure 1.8 (**a**). The main point is that there is a threshold below which the response is strictly zero. The response function looks qualitatively like the piecewise linear function

$$g(b) = \max\{0, b\}, \tag{1.8}$$

shown in panel (**b**). Neurons with this activation function are called *rectified linear units*, and the activation function (1.8) is called the *ReLU* function.

https://doi.org/10.1017/9781108860604.001 Published online by Cambridge University Press

1.4 Asynchronous Updates

Equations (1.2) and (1.5) are called *synchronous* update rules, because all neurons are updated in parallel: at time step t all inputs $s_j(t)$ are stored. Then all neurons i are simultaneously updated using the stored inputs. An alternative is to update only a single neuron per iteration, for example the one with index m:

$$s_i(t+1) = \begin{cases} g\big(\sum_j w_{mj} s_j(t) - \theta_m\big) & \text{for } i = m, \\ s_i(t) & \text{otherwise.} \end{cases} \tag{1.9}$$

This is called an *asynchronous* update rule. Different schemes for choosing neurons are used in asynchronous updating. One possibility is to arrange the neurons into a two-dimensional array and to update them one by one, in a certain order. In the *typewriter scheme*, for example, one updates the neurons in the top row of the array first, from left to right, then the second row from left to right, and so forth. A second possibility is to choose randomly which neuron to update.

If there are N neurons, then one synchronous step corresponds to N asynchronous steps, on average. This difference in time scales is not the only difference between synchronous and asynchronous updating. The asynchronous dynamics can be shown to converge to a definite state in certain cases, while the synchronous dynamics may fail to do so, resulting in periodic cycles that persist forever.

1.5 Summary

Artificial neural networks use a highly idealised model for the fundamental computation unit: the McCulloch-Pitts neuron (Figure 1.5) is a binary threshold unit, very similar to the model introduced originally by McCulloch and Pitts [6]. The units are linked together by weights w_{ij}, and each unit computes a weighted average of its inputs. The network performs these computations in sequence. Most neural-network algorithms are built using the model described in this chapter.

1.6 Further Reading

Two accounts of the history of artificial neural networks are especially recommended. First, the early history of the field is summarised in the introduction to the second edition of *Perceptrons: An Introduction to Computational Geometry* by Minsky and Papert [11], which came out in 1988. This book also contains a concise bibliography, with comments by Minsky and Papert. Second, in a short note, Kanal [23] reviews the work of Rosenblatt and puts it into context.

https://doi.org/10.1017/9781108860604.001 Published online by Cambridge University Press