Raúl Rojas

Neural Networks

A Systematic Introduction

Springer

Berlin Heidelberg New York
Hong Kong London
Milan Paris Tokyo

Foreword

One of the well-springs of mathematical inspiration has been the continuing attempt to formalize human thought. From the syllogisms of the Greeks, through all of logic and probability theory, cognitive models have led to beautiful mathematics and wide ranging application. But mental processes have proven to be more complex than any of the formal theories and the various idealizations have broken off to become separate fields of study and application.

It now appears that the same thing is happening with the recent developments in connectionist and neural computation. Starting in the 1940s and with great acceleration since the 1980s, there has been an effort to model cognition using formalisms based on increasingly sophisticated models of the physiology of neurons. Some branches of this work continue to focus on biological and psychological theory, but as in the past, the formalisms are taking on a mathematical and application life of their own. Several varieties of adaptive networks have proven to be practical in large difficult applied problems and this has led to interest in their mathematical and computational properties.

We are now beginning to see good textbooks for introducing the subject to various student groups. This book by Raúl Rojas is aimed at advanced undergraduates in computer science and mathematics. This is a revised version of his German text which has been quite successful. It is also a valuable self-instruction source for professionals interested in the relation of neural network ideas to theoretical computer science and articulating disciplines.

The book is divided into eighteen chapters, each designed to be taught in about one week. The first eight chapters follow a progression and the later ones can be covered in a variety of orders. The emphasis throughout is on explicating the computational nature of the structures and processes and relating them to other computational formalisms. Proofs are rigorous, but not overly formal, and there is extensive use of geometric intuition and diagrams. Specific applications are discussed, with the emphasis on computational rather than engineering issues. There is a modest number of exercises at the end of most chapters.

The most widely applied mechanisms involve adapting weights in feed-forward networks of uniform differentiable units and these are covered thoroughly. In addition to chapters on the background, fundamentals, and variations on backpropagation techniques, there is treatment of related questions from statistics and computational complexity.

There are also several chapters covering recurrent networks including the general associative net and the models of Hopfield and Kohonen. Stochastic variants are presented and linked to statistical physics and Boltzmann learning. Other chapters (weeks) are dedicated to fuzzy logic, modular neural networks, genetic algorithms, and an overview of computer hardware developed for neural computation. Each of the later chapters is self-contained and should be readable by a student who has mastered the first half of the book.

The most remarkable aspect of neural computation at the present is the speed at which it is maturing and becoming integrated with traditional disciplines. This book is both an indication of this trend and a vehicle for bringing it to a generation of mathematically inclined students.

Berkeley, California Jerome Feldman

Preface

This book arose from my lectures on neural networks at the Free University of Berlin and later at the University of Halle. I started writing a new text out of dissatisfaction with the literature available at the time. Most books on neural networks seemed to be chaotic collections of models and there was no clear unifying theoretical thread connecting them. The results of my efforts were published in German by Springer-Verlag under the title *Theorie der neuronalen Netze*. I tried in that book to put the accent on a systematic development of neural network theory and to stimulate the intuition of the reader by making use of many figures. Intuitive understanding fosters a more immediate grasp of the objects one studies, which stresses the concrete meaning of their relations. Since then some new books have appeared, which are more systematic and comprehensive than those previously available, but I think that there is still much room for improvement. The German edition has been quite successful and at the time of this writing it has gone through five printings in the space of three years.

However, this book is not a translation. I rewrote the text, added new sections, and deleted some others. The chapter on fast learning algorithms is completely new and some others have been adapted to deal with interesting additional topics. The book has been written for undergraduates, and the only mathematical tools needed are those which are learned during the first two years at university. The book offers enough material for a semester, although I do not normally go through all chapters. It is possible to omit some of them so as to spend more time on others. Some chapters from this book have been used successfully for university courses in Germany, Austria, and the United States.

The various branches of neural networks theory are all interrelated closely and quite often unexpectedly. Even so, because of the great diversity of the material treated, it was necessary to make each chapter more or less self-contained. There are a few minor repetitions but this renders each chapter understandable and interesting. There is considerable flexibility in the order of presentation for a course. Chapter 1 discusses the biological motivation of the whole enterprise. Chapters 2, 3, and 4 deal with the basics of threshold logic and should be considered as a unit. Chapter 5 introduces vector quantization and unsupervised learning. Chapter 6 gives a nice geometrical interpretation of perceptron learning. Those interested in stressing current applications of neural networks can skip Chapters 5 and 6 and go directly to the backpropagation algorithm (Chapter 7). I am especially proud of this chapter because it introduces backpropagation with minimal effort, using a graphical approach, yet the result is more general than the usual derivations of the algorithm in other books. I was rather surprised to see that *Neural Computation* published in 1996 a paper about what is essentially the method contained in my German book of 1993.

Those interested in statistics and complexity theory should review Chapters 9 and 10. Chapter 11 is an *intermezzo* and clarifies the relation between fuzzy logic and neural networks. Recurrent networks are handled in the three chapters, dealing respectively with associative memories, the Hopfield model, and Boltzmann machines. They should be also considered a unit. The book closes with a review of self-organization and evolutionary methods, followed by a short survey of currently available hardware for neural networks.

We are still struggling with neural network theory, trying to find a more systematic and comprehensive approach. Every chapter should convey to the reader an understanding of one small additional piece of the larger picture. I sometimes compare the current state of the theory with a big puzzle which we are all trying to put together. This explains the small puzzle pieces that the reader will find at the end of each chapter. Enough discussion – Let us start our journey into the fascinating world of artificial neural networks without further delay.

Errata and electronic information

This book has an Internet home page. Any errors reported by readers, new ideas, and suggested exercises can be downloaded from Berlin, Germany. The WWW link is: http://www.inf.fu-berlin.de/~rojas/neural. The home page offers also some additional useful information about neural networks. You can send your comments by e-mail to rojas@inf.fu-berlin.de.

Acknowledgements

Many friends and colleagues have contributed to the quality of this book. The names of some of them are listed in the preface to the German edition of 1993. Phil Maher, Rosi Weinert-Knapp, and Gaye Rochow revised my original manuscript. Andrew J. Ross, English editor at Springer-Verlag in Heidelberg, took great care in degermanizing my linguistic constructions.

The book was written at three different institutions: The Free University of Berlin provided an ideal working environment during the first phase of writing. Vilim Vesligaj configured TeX so that it would accept Springer's style.

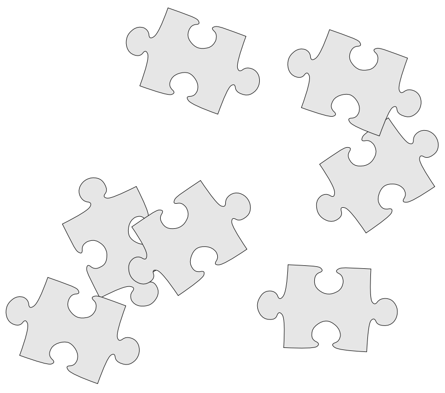
Günter Feuer, Marcus Pfister, Willi Wolf, and Birgit Müller were patient discussion partners. I had many discussions with Frank Darius on damned lies and statistics. The work was finished at Halle's Martin Luther University. My collaborator Bernhard Frötschl and some of my students found many of my early TeX-typos. I profited from two visits to the International Computer Science Institute in Berkeley during the summers of 1994 and 1995. I especially thank Jerry Feldman, Joachim Beer, and Nelson Morgan for their encouragement. Lokendra Shastri tested the backpropagation chapter "in the field", that is in his course on connectionist models at UC Berkeley. It was very rewarding to spend the evenings talking to Andres and Celina Albanese about other kinds of networks (namely real computer networks). Lotfi Zadeh was very kind in inviting me to present my visualization methods at his Seminar on Soft Computing. Due to the efforts of Dieter Ernst there is no good restaurant in the Bay Area where I have not been.

It has been a pleasure working with Springer-Verlag and the head of the planning section, Dr. Hans Wössner, in the development of this text. With him cheering from Heidelberg I could survive the whole ordeal of TeXing more than 500 pages.

Finally, I thank my daughter Tania and my wife Margarita Esponda for their love and support during the writing of this book. Since my German book was dedicated to Margarita, the new English edition is now dedicated to Tania. I really hope she will read this book in the future (and I hope she will like it).

Berlin and Halle
March 1996

Raúl Rojas González



"For Reason, in this sense, is nothing but
Reckoning (that is, Adding and Subtracting)."

Thomas Hobbes, *Leviathan.*

1

The Biological Paradigm

1.1 Neural computation

Research in the field of neural networks has been attracting increasing attention in recent years. Since 1943, when Warren McCulloch and Walter Pitts presented the first model of artificial neurons, new and more sophisticated proposals have been made from decade to decade. Mathematical analysis has solved some of the mysteries posed by the new models but has left many questions open for future investigations. Needless to say, the study of neurons, their interconnections, and their role as the brain's elementary building blocks is one of the most dynamic and important research fields in modern biology. We can illustrate the relevance of this endeavor by pointing out that between 1901 and 1991 approximately ten percent of the Nobel Prizes for Physiology and Medicine were awarded to scientists who contributed to the understanding of the brain. It is not an exaggeration to say that we have learned more about the nervous system in the last fifty years than ever before.

In this book we deal with *artificial neural networks*, and therefore the first question to be clarified is their relation to the biological paradigm. What do we abstract from real neurons for our models? What is the link between neurons and artificial computing units? This chapter gives a preliminary answer to these important questions.

1.1.1 Natural and artificial neural networks

Artificial neural networks are an attempt at modeling the information processing capabilities of nervous systems. Thus, first of all, we need to consider the essential properties of biological neural networks from the viewpoint of information processing. This will allow us to design abstract models of artificial neural networks, which can then be simulated and analyzed.

Although the models which have been proposed to explain the structure of the brain and the nervous systems of some animals are different in many respects, there is a general consensus that the essence of the operation of neural ensembles is “control through communication” [72]. Animal nervous systems are composed of thousands or millions of interconnected cells. Each one of them is a very complex arrangement which deals with incoming signals in many different ways. However, neurons are rather slow when compared to electronic logic gates. These can achieve switching times of a few nanoseconds, whereas neurons need several milliseconds to react to a stimulus. Nevertheless the brain is capable of solving problems which no digital computer can yet efficiently deal with.

Massive and hierarchical networking of the brain seems to be the fundamental precondition for the emergence of consciousness and complex behavior [202]. So far, however, biologists and neurologists have concentrated their research on uncovering the properties of individual neurons. Today, the mechanisms for the production and transport of signals from one neuron to the other are well-understood physiological phenomena, but how these individual systems cooperate to form complex and massively parallel systems capable of incredible information processing feats has not yet been completely elucidated. Mathematics, physics, and computer science can provide invaluable help in the study of these complex systems. It is not surprising that the study of the brain has become one of the most interdisciplinary areas of scientific research in recent years.

However, we should be careful with the metaphors and paradigms commonly introduced when dealing with the nervous system. It seems to be a constant in the history of science that the brain has always been compared to the most complicated contemporary artifact produced by human industry [297]. In ancient times the brain was compared to a pneumatic machine, in the Renaissance to a clockwork, and at the end of the last century to the telephone network. There are some today who consider computers the paradigm par excellence of a nervous system. It is rather paradoxical that when John von Neumann wrote his classical description of future universal computers, he tried to choose terms that would describe computers in terms of brains, not brains in terms of computers.

The nervous system of an animal is an information processing totality. The sensory inputs, i.e., signals from the environment, are coded and processed to evoke the appropriate response. Biological neural networks are just one of many possible solutions to the problem of processing information. The main difference between neural networks and conventional computer systems is the massive parallelism and redundancy which they exploit in order to deal with the unreliability of the individual computing units. Moreover, biological neural networks are self-organizing systems and each individual neuron is also a delicate self-organizing structure capable of processing information in many different ways.

In this book we study the information processing capabilities of complex hierarchical networks of simple computing units. We deal with systems whose structure is only partially predetermined. Some parameters modify the capabilities of the network and it is our task to find the best combination for the solution of a given problem. The adjustment of the parameters will be done through a *learning algorithm*, i.e., not through explicit programming but through an automatic adaptive method.

A cursory review of the relevant literature on artificial neural networks leaves the impression of a chaotic mixture of very different network topologies and learning algorithms. Commercial neural network simulators sometimes offer several dozens of possible models. The large number of proposals has led to a situation in which each single model appears as part of a big puzzle whereas the bigger picture is absent. Consequently, in the following chapters we try to solve this puzzle by systematically introducing and discussing each of the neural network models in relation to the others.

Our approach consists of stating and answering the following questions: what information processing capabilities emerge in hierarchical systems of primitive computing units? What can be computed with these networks? How can these networks determine their structure in a self-organizing manner?

We start by considering biological systems. Artificial neural networks have aroused so much interest in recent years, not only because they exhibit interesting properties, but also because they try to mirror the kind of information processing capabilities of nervous systems. Since information processing consists of transforming signals, we deal with the biological mechanisms for their generation and transmission in this chapter. We discuss those biological processes by which neurons produce signals, and absorb and modify them in order to retransmit the result. In this way biological neural networks give us a clue regarding the properties which would be interesting to include in our artificial networks.

1.1.2 Models of computation

Artificial neural networks can be considered as just another approach to the problem of computation. The first formal definitions of computability were proposed in the 1930s and '40s and at least five different alternatives were studied at the time. The computer era was started, not with one single approach, but with a contest of alternative computing models. We all know that the von Neumann computer emerged as the undisputed winner in this confrontation, but its triumph did not lead to the dismissal of the other computing models. Figure 1.1 shows the five principal contenders:

The mathematical model

Mathematicians avoided dealing with the problem of a function's computability until the beginning of this century. This happened not just because existence theorems were considered sufficient to deal with functions, but mainly because nobody had come up with a satisfactory definition of *computability*, certainly a relative concept which depends on the specific tools that can be used. The general solution for algebraic equations of degree five, for example, cannot be formulated using only algebraic functions, yet this can be done if a more general class of functions is allowed as computational primitives. The squaring of the circle, to give another example, is impossible using ruler and compass, but it has a trivial real solution.

If we want to talk about computability we must therefore specify which tools are available. We can start with the idea that some primitive functions and composition rules are "obviously" computable. All other functions which can be expressed in terms of these primitives and composition rules are then also computable.

David Hilbert, the famous German mathematician, was the first to state the conjecture that a certain class of functions contains all intuitively computable functions. Hilbert was referring to the primitive recursive functions, the class of functions which can be constructed from the zero and successor function using composition, projection, and a deterministic number of iterations (primitive recursion). However, in 1928, Wilhelm Ackermann was able to find a computable function which is not primitive recursive. This led to the definition of the general recursive functions [154]. In this formalism, a new composition rule has to be introduced, the so-called μ operator, which is equivalent to an indeterminate recursion or a lookup in an infinite table. At the same time Alonzo Church and collaborators developed the lambda calculus, another alternative to the mathematical definition of the computability concept [380]. In 1936, Church and Kleene were able to show that the general recursive functions can be expressed in the formalism of the lambda calculus. This led to the *Church thesis* that computable functions are the general recursive functions. David Deutsch has recently added that this thesis should be considered to be a statement about the physical world and be given the same status as a physical principle. He thus speaks of a "Church principle" [109].

The logic-operational model (Turing machines)

In his classical paper "On Computable Numbers with an Application to the Entscheidungsproblem" Alan Turing introduced another kind of computing model. The advantage of his approach is that it consists in an operational, mechanical model of computability. A Turing machine is composed of an infinite tape, in which symbols can be stored and read again. A read-write head can move to the left or to the right according to its internal state, which is updated at each step. The *Turing thesis* states that computable functions are those which can be computed with this kind of device. It was formulated concurrently with the Church thesis and Turing was able to show almost immediately that they are equivalent [435]. The Turing approach made clear for the first time what "programming" means, curiously enough at a time when no computer had yet been built.

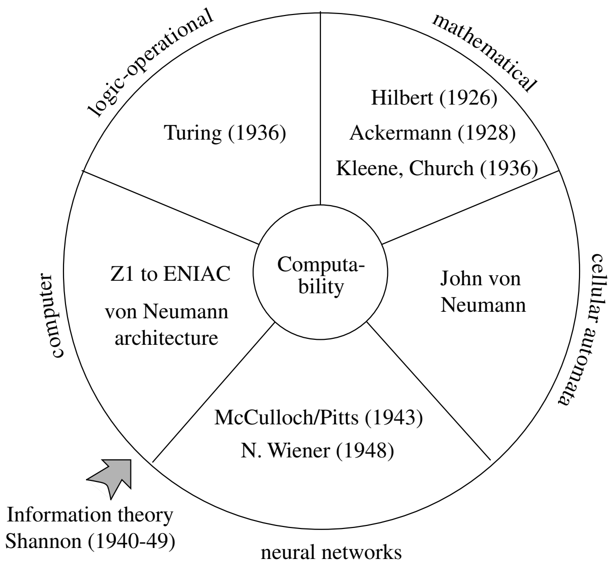


Fig. 1.1. Five models of computation

The computer model

The first electronic computing devices were developed in the 1930s and '40s. Since then, "computation-with-the-computer" has been regarded as computability itself. However the first engineers developing computers were for the most part unaware of Turing's or Church's research. Konrad Zuse, for example, developed in Berlin between 1938 and 1944 the computing machines Z1 and Z3 which were programmable but not universal, because they could not reach the whole space of the computable functions. Zuse's machines were able to process a sequence of instructions but could not iterate. Other computers of the time, like the Mark I built at Harvard, could iterate a constant number of times but were incapable of executing open-ended iterations (WHILE loops). Therefore the Mark I could compute the primitive but not the general recursive functions. Also the ENIAC, which is usually hailed as the world's first electronic computer, was incapable of dealing with open-ended loops, since iterations were determined by specific connections between modules of the machine. It seems that the first universal computer was the Mark I built in Manchester [96, 375]. This machine was able to cover all computable functions by making use of conditional branching and self-modifying programs, which is one possible way of implementing indexed addressing [268].

Cellular automata

The history of the development of the first mechanical and electronic computing devices shows how difficult it was to reach a consensus on the architecture of universal computers. Aspects such as the economy or the dependability of the building blocks played a role in the discussion, but the main problem was the definition of the minimal architecture needed for universality. In machines like the Mark I and the ENIAC there was no clear separation between memory and processor, and both functional elements were intertwined. Some machines still worked with base 10 and not 2, some were sequential and others parallel.

John von Neumann, who played a major role in defining the architecture of sequential machines, analyzed at that time a new computational model which he called *cellular automata*. Such automata operate in a "computing space" in which all data can be processed simultaneously. The main problem for cellular automata is communication and coordination between all the computing cells. This can be guaranteed through certain algorithms and conventions. It is not difficult to show that all computable functions, in the sense of Turing, can also be computed with cellular automata, even of the one-dimensional type, possessing only a few states. Turing himself considered this kind of computing model at one point in his career [192].

Cellular automata as computing model resemble massively parallel multiprocessor systems of the kind that has attracted considerable interest recently.

The biological model (neural networks)

The explanation of important aspects of the physiology of neurons set the stage for the formulation of artificial neural network models which do not operate sequentially, as Turing machines do. Neural networks have a hierarchical multilayered structure which sets them apart from cellular automata, so that information is transmitted not only to the immediate neighbors but also to more distant units. In artificial neural networks one can connect each unit to any other. In contrast to conventional computers, no program is handed over to the hardware – such a program has to be created, that is, the free parameters of the network have to be found adaptively.

Although neural networks and cellular automata are potentially more efficient than conventional computers in certain application areas, at the time of their conception they were not yet ready to take center stage. The necessary theory for harnessing the dynamics of complex parallel systems is still being developed right before our eyes. In the meantime, conventional computer technology has made great strides.

There is no better illustration for the simultaneous and related emergence of these various computability models than the life and work of John von Neumann himself. He participated in the definition and development of at least three of these models: in the architecture of sequential computers [417], the theory of cellular automata and the first neural network models. He also collaborated with Church and Turing in Princeton [192].

Artificial neural networks have, as initial motivation, the structure of biological systems, and constitute an alternative computability paradigm. For that reason we will review some aspects of the way in which biological systems perform information processing. The fascination which still pervades this research field has much to do with the points of contact with the surprisingly elegant methods used by neurons in order to process information at the cellular level. Several million years of evolution have led to very sophisticated solutions to the problem of dealing with an uncertain environment. In this chapter we will discuss some elements of these strategies in order to determine what features we want to adopt in our abstract models of neural networks.

1.1.3 Elements of a computing model

What are the *elementary components* of any conceivable computing model? In the theory of general recursive functions, for example, it is possible to reduce any computable function to some composition rules and a small set of primitive functions. For a universal computer, we ask about the existence of a minimal and sufficient instruction set. For an arbitrary computing model the following metaphoric expression has been proposed:

$$computation = storage + transmission + processing.$$

The mechanical computation of a function presupposes that these three elements are present, that is, that data can be stored, communicated to the functional units of the model and transformed. It is implicitly assumed that a certain coding of the data has been agreed upon. Coding plays an important role in information processing because, as Claude Shannon showed in 1948, when noise is present information can still be transmitted without loss, if the right code with the right amount of redundancy is chosen.

Modern computers transform storage of information into a form of information transmission. Static memory chips store a bit as a circulating current until the bit is read. Turing machines store information in an infinite tape, whereas transmission is performed by the read-write head. Cellular automata store information in each cell, which at the same time is a small processor.

1.2 Networks of neurons

In biological neural networks information is stored at the contact points between different neurons, the so-called *synapses*. Later we will discuss what role these elements play for the storage, transmission, and processing of information. Other forms of storage are also known, because neurons are themselves complex systems of self-organizing signaling. In the next few pages we cannot do justice to all this complexity, but we analyze the most salient features and, with the metaphoric expression given above in mind, we will ask: how do neurons compute?

1.2.1 Structure of the neurons

Nervous systems possess global architectures of variable complexity, but all are composed of similar building blocks, the neural cells or neurons. They can perform different functions, which in turn leads to a very variable morphology. If we analyze the human cortex under a microscope, we can find several different types of neurons. Figure 1.2 shows a diagram of a portion of the cortex. Although the neurons have very different forms, it is possible to recognize a hierarchical structure of six different layers. Each one has specific functional characteristics. Sensory signals, for example, are transmitted directly to the fourth layer and from there processing is taken over by other layers.

Fig. 1.2. A view of the human cortex [from Lassen et al. 1988]

Neurons receive signals and produce a response. The general structure of a generic neuron is shown in Figure 1.3[1]. The branches to the left are the transmission channels for incoming information and are called *dendrites*. Dendrites receive the signals at the contact regions with other cells, the synapses

[1] Some animals have neurons with a very different morphology. In insects, for example, the dendrites go directly into the axon and the cell body is located far from them. The way these neurons work is nevertheless very similar to the description in this chapter.

mentioned already. Organelles in the body of the cell produce all necessary chemicals for the continuous working of the neuron. The mitochondria, visible in Figure 1.3, can be thought of as part of the energy supply of the cell, since they produce chemicals which are consumed by other cell structures. The output signals are transmitted by the *axon*, of which each cell has at most one. Some cells do not have an axon, because their task is only to set some cells in contact with others (in the retina, for example).

Fig. 1.3. A typical motor neuron [from Stevens 1988]

These four elements, dendrites, synapses, cell body, and axon, are the minimal structure we will adopt from the biological model. Artificial neurons for computing will have input channels, a cell body and an output channel. Synapses will be simulated by contact points between the cell body and input or output connections; a *weight* will be associated with these points.

1.2.2 Transmission of information

The fundamental problem of any information processing system is the transmission of information, as data storage can be transformed into a recurrent transmission of information between two points [177].

Biologists have known for more than 100 years that neurons transmit information using electrical signals. Because we are dealing with biological structures, this cannot be done by simple electronic transport as in metallic cables. Evolution arrived at another solution involving ions and semipermeable membranes.

Our body consists mainly of water, 55% of which is contained within the cells and 45% forming its environment. The cells preserve their identity and biological components by enclosing the protoplasm in a membrane made of a double layer of molecules that form a diffusion barrier. Some salts, present in our body, dissolve in the intracellular and extracellular fluid and dissociate into negative and positive ions. Sodium chloride, for example, dissociates into positive sodium ions (Na^+) and negative chlorine ions (Cl^-). Other positive ions present in the interior or exterior of the cells are potassium (K^+) and calcium (Ca^{2+}). The membranes of the cells exhibit different degrees of permeability for each one of these ions. The permeability is determined by the number and size of pores in the membrane, the so-called *ionic channels*. These are macromolecules with forms and charges which allow only certain ions to go from one side of the cell membrane to the other. Channels are selectively permeable to sodium, potassium or calcium ions. The specific permeability of the membrane leads to different distributions of ions in the interior and the exterior of the cells and this, in turn, to the interior of neurons being negatively charged with respect to the extracellular fluid.

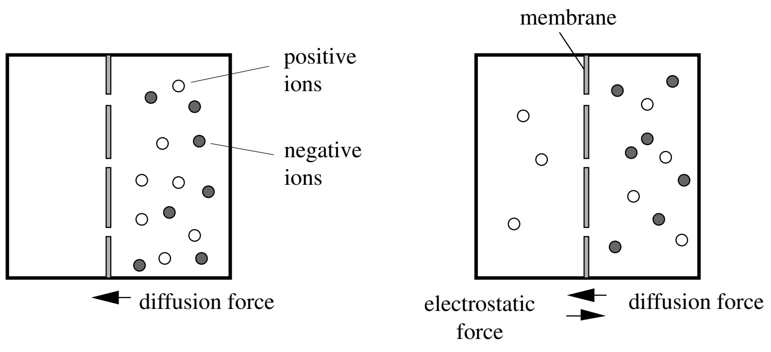


Fig. 1.4. Diffusion of ions through a membrane

Figure 1.4 illustrates this phenomenon. A box is divided into two parts separated by a membrane permeable only to positive ions. Initially the same number of positive and negative ions is located in the right side of the box. Later, some positive ions move from the right to the left through the pores in the membrane. This occurs because atoms and molecules have a thermodynamical tendency to distribute homogeneously in space by the process called diffusion. The process continues until the electrostatic repulsion from the positive ions on the left side balances the diffusion potential. A potential difference, called the *reversal potential*, is established and the system behaves like a small electric battery. In a cell, if the initial concentration of potassium ions in its interior is greater than in its exterior, positive potassium ions will diffuse through the open potassium-selective channels. If these are the only ionic channels, negative ions cannot disperse through the membrane. The interior of the cell becomes negatively charged with respect to the exterior, creating a potential difference between both sides of the membrane. This balances the diffusion potential, and, at some point, the net flow of potassium ions through the membrane falls to zero. The system reaches a steady state. The potential difference E for one kind of ion is given by the Nernst formula

$$E = k(\ln(c_o) - \ln(c_i))$$

where c_i is the concentration inside the cell, c_o the concentration in the extracellular fluid and k is a proportionality constant [295]. For potassium ions the equilibrium potential is -80 mV.

Because there are several different concentrations of ions inside and outside of the cell, the question is, what is the potential difference which is finally reached. The exact potential in the interior of the cell depends on the mixture of concentrations. A typical cell's potential is -70 mV, which is produced mainly by the ion concentrations shown in Figure 1.5 (A^- designates negatively charged biomolecules). The two main ions in the cell are sodium and potassium. Equilibrium potential for sodium lies around 58 mV. The cell reaches a potential between -80 mV and 58 mV. The cell's equilibrium potential is nearer to the value induced by potassium, because the permeability of the membrane to potassium is greater than to sodium. There is a net outflow of potassium ions at this potential and a net inflow of sodium ions. However, the sodium ions are less mobile because fewer open channels are available. In the steady state the cell membrane experiences two currents of ions trying to reach their individual equilibrium potential. An ion pump guarantees that the concentration of ions does not change with time.

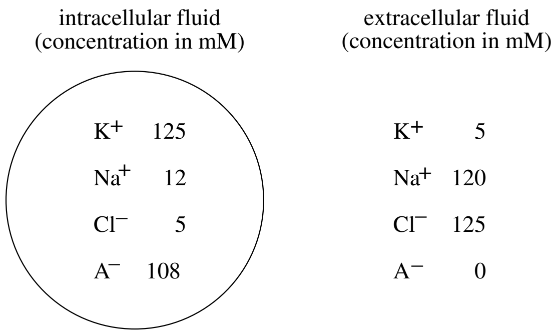


Fig. 1.5. Ion concentrations inside and outside a cell

The British scientists Alan Hodgkin and Andrew Huxley were able to show that it is possible to build an electric model of the cell membrane based on very simple assumptions. The membrane behaves as a capacitor made of two isolated layers of lipids. It can be charged with positive or negative ions. The different concentrations of several classes of ions in the interior and exterior of the cell provide an energy source capable of negatively polarizing the interior of the cell. Figure 1.6 shows a diagram of the model proposed by Hodgkin and

Huxley. The specific permeability of the membrane for each class of ion can be modeled like a conductance (the reciprocal of resistance).

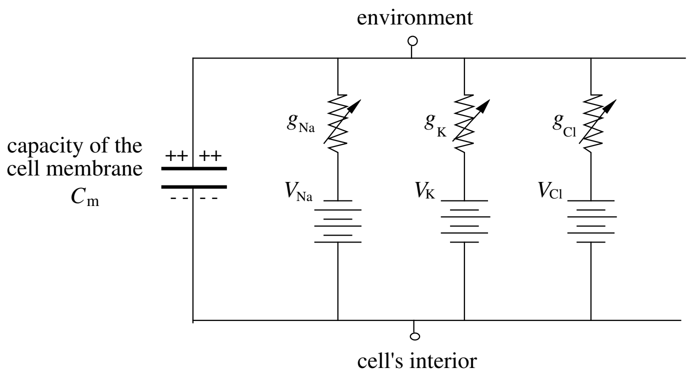


Fig. 1.6. The Hodgkin–Huxley model of a cell membrane

The electric model is a simplification, because there are other classes of ions and electrically charged proteins present in the cell. In the model, three ions compete to create a potential difference between the interior and exterior of the cell. The conductances g_{Na}, g_{K}, and g_{L} reflect the permeability of the membrane to sodium, potassium, and leakages, i.e., the number of open channels of each class. A signal can be produced by modifying the polarity of the cell through changes in the conductances g_{Na} and g_{K}. By making g_{Na} larger and the mobility of sodium ions greater than the mobility of potassium ions, the polarity of the cell changes from -70 mV to a positive value, nearer to the 58 mV at which sodium ions reach equilibrium. If the conductance g_{K} then becomes larger and g_{Na} falls back to its original value, the interior of the cell becomes negative again, overshooting in fact by going below -70 mV. To generate a signal, a mechanism for depolarizing and polarizing the cell in a controlled way is necessary.

The conductance and resistance of a cell membrane in relation to the different classes of ions depends on its permeability. This can be controlled by opening or closing excitable ionic channels. In addition to the static ionic channels already mentioned, there is another class which can be electrically controlled. These channels react to a depolarization of the cell membrane. When this happens, that is, when the potential of the interior of the cell in relation to the exterior reaches a threshold, the sodium-selective channels open automatically and positive sodium ions flow into the cell making its interior positive. This in turn leads to the opening of the potassium-selective channels and positive potassium ions flow to the exterior of the cell, restoring the original negative polarization.

Figure 1.7 shows a diagram of an electrically controlled sodium-selective channel which lets only sodium ions flow across. This effect is produced by the small aperture in the middle of the channel which is negatively charged (at time $t = 1$). If the interior of the cell becomes positive relative to the exterior, some negative charges are displaced in the channel and this produces the opening of a gate ($t = 2$). Sodium ions flow through the channel and into the cell. After a short time the second gate is closed and the ionic channel is sealed ($t = 3$). The opening of the channel corresponds to a change of membrane conductivity as explained above.

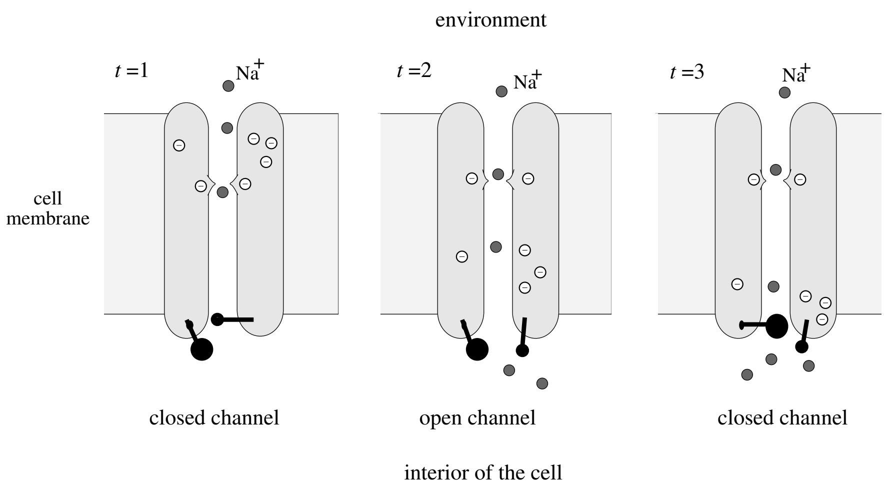


Fig. 1.7. Electrically controlled ionic channels

Static and electrically controlled ionic channels are not only found in neurons. As in any electrical system there are charge losses which have to be continuously balanced. A sodium ion pump (Figure 1.8) transports the excess of sodium ions out of the cell and, at the same time, potassium ions into its interior. The ion pump consumes adenosine triphosphate (ATP), a substance produced by the mitochondria, helping to stabilize the polarization potential of -70 mV. The ion pump is an example of a self-regulating system, because it is accelerated or decelerated by the differences in ion concentrations on both sides of the membrane. Ion pumps are constantly active and account for a considerable part of the energy requirements of the nervous system.

Neural signals are produced and transmitted at the cell membrane. The signals are represented by depolarization waves traveling through the axons in a self-regenerating manner. Figure 1.9 shows the form of such a depolarization wave, called an *action potential.* The x-dimension is shown horizontally and the diagram shows the instantaneous potential in each segment of the axon.

An action potential is produced by an initial depolarization of the cell membrane. The potential increases from -70 mV up to $+40$ mV. After some time the membrane potential becomes negative again but it overshoots, going

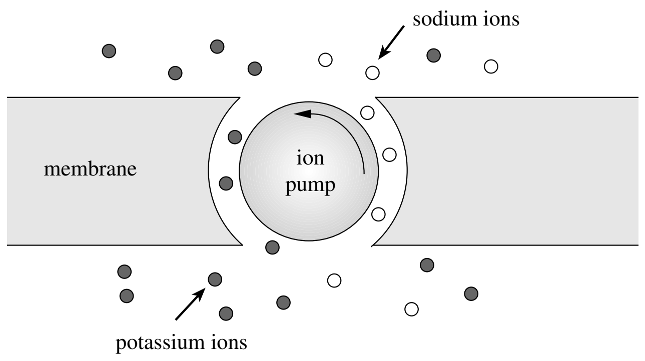


Fig. 1.8. Sodium and potassium ion pump

as low as −80 mV. The cell recovers gradually and the cell membrane returns to the initial potential. The switching time of the neurons is determined, as in any resistor-capacitor configuration, by the RC constant. In neurons, 2.4 milliseconds is a typical value for this constant.

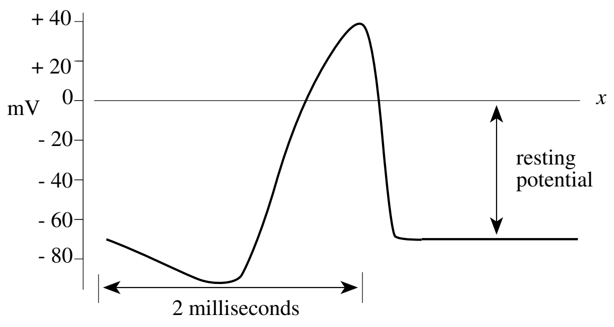


Fig. 1.9. Typical form of the action potential

Figure 1.10 shows an action potential traveling through an axon. A local perturbation, produced by the signals arriving at the dendrites, leads to the opening of the sodium-selective channels in a certain region of the cell membrane. The membrane is thus depolarized and positive sodium ions flow into the cell. After a short delay, the outward flow of potassium ions compensates the depolarization of the membrane. Both perturbations – the opening of the sodium and potassium-selective channels – are transmitted through the axon like falling dominos. In the entire process only local energy is consumed, that is, only the energy stored in the polarized membrane itself. The action potential is thus a wave of Na^+ permeability increase followed by a wave of K^+ permeability increase. It is easy to see that charged particles only move a short distance in the direction of the perturbation, only as much as is necessary to perturb the next channels and bring the next "domino" to fall.

Figure 1.10 also shows how impulse trains are produced in the cells. After a signal is produced a new one follows. Each neural signal is an all-or-nothing self-propagating regenerative event as each signal has the same form and amplitude. At this level we can safely speak about digital transmission of information.

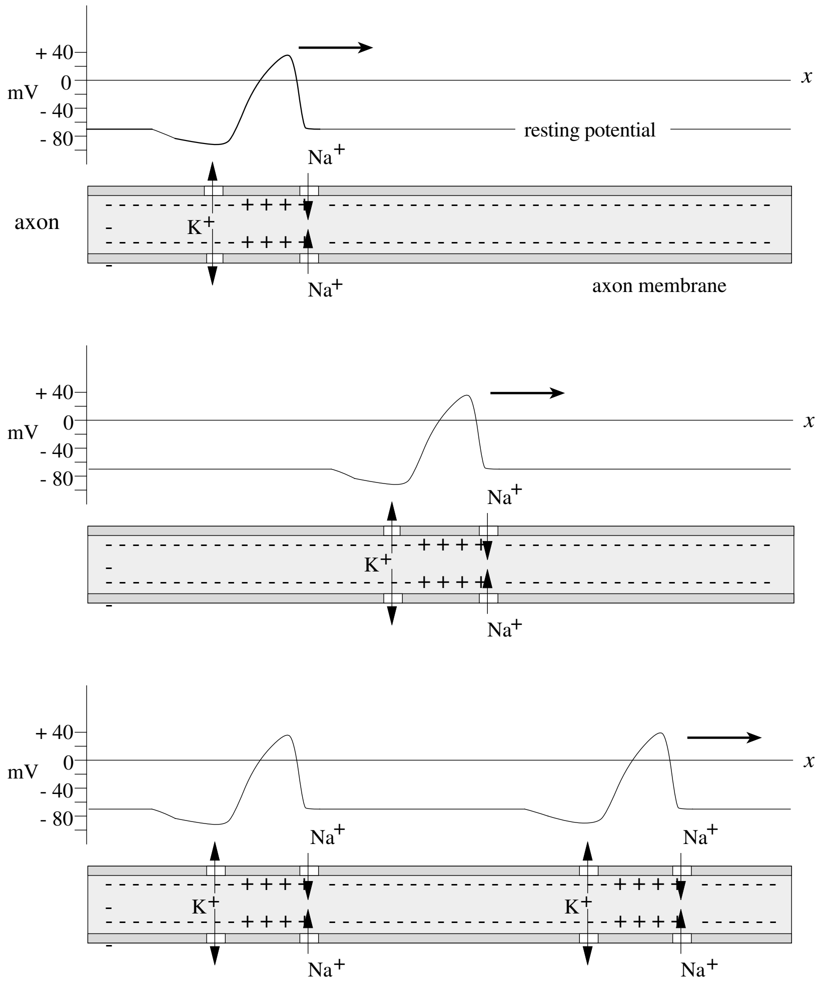


Fig. 1.10. Transmission of an action potential [Stevens 1988]

With this picture of the way an action potential is generated in mind, it is easy to understand the celebrated Hodgkin–Huxley differential equation which describes the instantaneous variation of the cell's potential V as a function of the conductances of sodium, potassium and leakages $(g_{\mathrm{Na}}, g_{\mathrm{K}}, g_{\mathrm{L}})$ and of the equilibrium potentials for all three groups of ions called $V_{\mathrm{Na}}, V_{\mathrm{K}}$ and V_{L} with respect to the current potential:

$$\frac{dV}{dt} = \frac{1}{C_{\mathrm{m}}}(I - g_{\mathrm{Na}}(V - V_{\mathrm{Na}}) - g_{\mathrm{K}}(V - V_{\mathrm{K}}) - g_{\mathrm{L}}(V - V_{\mathrm{L}})). \tag{1.1}$$

In this equation C_{m} is the capacitance of the cell membrane. The terms $V - V_{\mathrm{Na}}$, $V - V_{\mathrm{K}}$, $V - V_{\mathrm{L}}$ are the electromotive forces acting on the ions. Any variation of the conductances translates into a corresponding variation of the cell's potential V. The variations of g_{Na} and g_{K} are given by differential equations which describe their oscillations. The conductance of the leakages, g_{L}, can be taken as a constant.

A neuron codes its level of activity by adjusting the frequency of the generated impulses. This frequency is greater for a greater stimulus. In some cells the mapping from stimulus to frequency is linear in a certain interval [72]. This means that information is transmitted from cell to cell using what engineers call frequency modulation. This form of transmission helps to increase the accuracy of the signal and to minimize the energy consumption of the cells.

1.2.3 Information processing at the neurons and synapses

Neurons transmit information using action potentials. The processing of this information involves a combination of electrical and chemical processes, regulated for the most part at the interface between neurons, the synapses.

Neurons transmit information not only by electrical perturbations. Although electrical synapses are also known, most synapses make use of chemical signaling. Figure 1.11 is a classical diagram of a typical synapse. The synapse appears as a thickening of the axon. The small vacuoles in the interior, the synaptic vesicles, contain chemical transmitters. The small gap between a synapse and the cell to which it is attached is known as the synaptic gap.

When an electric impulse arrives at a synapse, the synaptic vesicles fuse with the cell membrane (Figure 1.12). The transmitters flow into the synaptic gap and some attach themselves to the ionic channels, as in our example. If the transmitter is of the right kind, the ionic channels are opened and more ions can now flow from the exterior to the interior of the cell. The cell's potential is altered in this way. If the potential in the interior of the cell is increased, this helps prepare an action potential and the synapse causes an excitation of the cell. If negative ions are transported into the cell, the probability of starting an action potential is decreased for some time and we are dealing with an inhibitory synapse.

Synapses determine a direction for the transmission of information. Signals flow from one cell to the other in a well-defined manner. This will be expressed in artificial neural networks models by embedding the computing elements in a

Fig. 1.11. Transversal view of a synapse [from Stevens 1988]

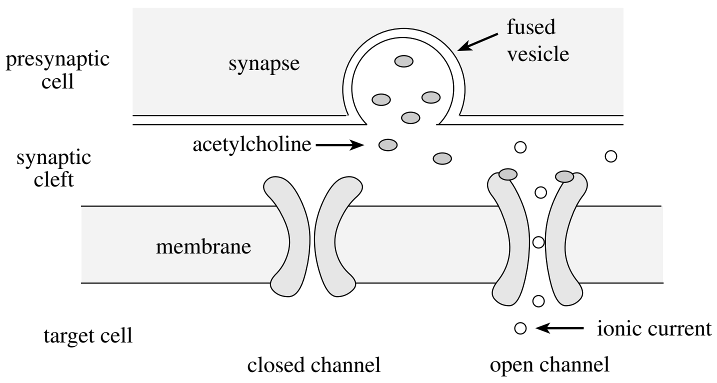


Fig. 1.12. Chemical signaling at the synapse

directed graph. A well-defined direction of information flow is a basic element in every computing model, and is implemented in digital systems by using diodes and directional amplifiers.

The interplay between electrical transmission of information in the cell and chemical transmission between cells is the basis for neural information processing. Cells process information by integrating incoming signals and by reacting to inhibition. The flow of transmitters from an excitatory synapse leads to a depolarization of the attached cell. The depolarization must exceed a threshold, that is, enough ionic channels have to be opened in order to produce an action potential. This can be achieved by several pulses arriving simultaneously or within a short time interval at the cell. If the quantity of transmitters reaches a certain level and enough ionic channels are triggered, the cell reaches its activation threshold. At this moment an action potential is generated at the axon of this cell.

In most neurons, action potentials are produced at the so-called axon hillock, the part of the axon nearest to the cell body. In this region of the cell, the number of ionic channels is larger and the cell's threshold lower [427]. The dendrites collect the electrical signals which are then transmitted electrotonically, that is through the cytoplasm [420]. The transmission of information at the dendrites makes use of additional electrical effects. Streams of ions are collected at the dendrites and brought to the axon hillock. There is spatial summation of information when signals coming from different dendrites are collected, and temporal summation when signals arriving consecutively are combined to produce a single reaction. In some neurons not only the axon hillock but also the dendrites can produce action potentials. In this case information processing at the cell is more complex than in the standard case.

It can be shown that digital signals combined in an excitatory or inhibitory way can be used to implement any desired logical function (Chap. 2). The number of computing units required can be reduced if the information is not only transmitted but also weighted. This can be achieved by multiplying the signal by a constant. Such is the kind of processing we find at the synapses. Each signal is an all-or-none event but the number of ionic channels triggered by the signal is different from synapse to synapse. It can happen that a single synapse can push a cell to fire an action potential, but other synapses can achieve this only by simultaneously exciting the cell. With each synapse i $(1 \leq i \leq n)$ we can therefore associate a numerical weight w_i. If all synapses are activated at the same time, the information which will be transmitted is $w_1 + w_2 + \cdots + w_n$. If this value is greater than the cell's threshold, the cell will fire a pulse.

It follows from this description that neurons process information at the membrane. The membrane regulates both transmission and processing of information. Summation of signals and comparison with a threshold is a combined effect of the membrane and the cytoplasm. If a pulse is generated, it is transmitted and the synapses set some transmitter molecules free. From this description an *abstract neuron* [72] can be modeled which contains dendrites, a cell body and an axon. The same three elements will be present in our artificial computing units.

1.2.4 Storage of information – learning

In neural networks information is stored at the synapses. Some other forms of information storage may be present, but they are either still unknown or not very well understood.

A synapse's efficiency in eliciting the depolarization of the contacted cell can be increased if more ionic channels are opened. In recent years NMDA receptors have been studied because they exhibit some properties which could help explain some forms of learning in neurons [72].

NMDA receptors are ionic channels permeable for different kinds of molecules, like sodium, calcium, or potassium ions. These channels are blocked by a magnesium ion in such a way that the permeability for sodium and calcium is low. If the cell is brought up to a certain excitation level, the ionic channels lose the magnesium ion and become unblocked. The permeability for Ca^{2+} ions increases immediately. Through the flow of calcium ions a chain of reactions is started which produces a durable change of the threshold level of the cell [420, 360]. Figure 1.13 shows a diagram of this process.

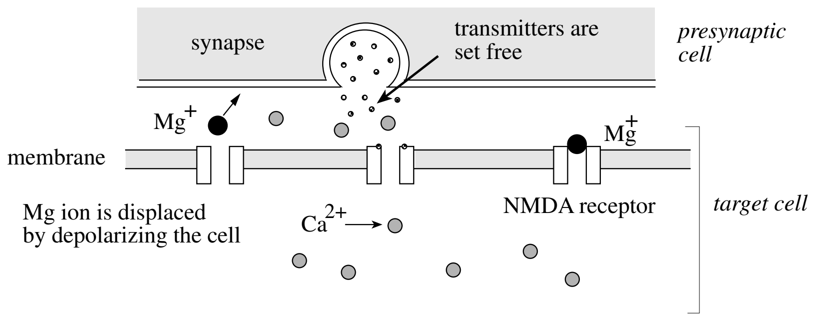

synapse
transmitters are set free
presynaptic cell
Mg+
Mg+
membrane
Mg ion is displaced by depolarizing the cell
Ca2+
NMDA receptor
target cell

Fig. 1.13. Unblocking of an NMDA receptor

NMDA receptors are just one of the mechanisms used by neurons to increase their plasticity, i.e., their adaptability to changing circumstances. Through the modification of the membrane's permeability a cell can be trained to fire more often by setting a lower firing threshold. NMDA receptors also offer an explanation for the observed phenomenon that cells which are not stimulated to fire tend to set a higher firing threshold. The stored information must be refreshed periodically in order to maintain the optimal permeability of the cell membrane.

This kind of information storage is also used in artificial neural networks. Synaptic efficiency can be modeled as a property of the edges of the network. The networks of neurons are thus connected through edges with different transmission efficiencies. Information flowing through the edges is multiplied by a constant which reflects their efficiency. One of the most popular learning algorithms for artificial neural networks is *Hebbian learning.* The efficiency of synapses is increased any time the two cells which are connected through this synapse fire simultaneously and is decreased when the firing states of the two cells are uncorrelated. The NMDA receptors act as coincidence detectors of presynaptic and postsynaptic activity, which in turn leads to greater synaptic efficiency.

1.2.5 The neuron – a self-organizing system

The short review of the properties of biological neurons in the previous sections is necessarily incomplete and can offer only a rough description of the mechanisms and processes by which neurons deal with information. Nerve cells are very complex self-organizing systems which have evolved in the course of millions of years. How were these exquisitely fine-tuned information processing organs developed? Where do we find the evolutionary origin of consciousness?

The information processing capabilities of neurons depend essentially on the characteristics of the cell membrane. Ionic channels appeared very early in evolution to allow unicellular organisms to get some kind of feedback from the environment. Consider the case of a paramecium, a protozoan with cilia, which are hairlike processes which provide it with locomotion. A paramecium has a membrane cell with ionic channels and its normal state is one in which the interior of the cell is negative with respect to the exterior. In this state the cilia around the membrane beat rhythmically and propel the paramecium forward. If an obstacle is encountered, some ionic channels sensitive to contact open, let ions into the cell, and depolarize it. The depolarization of the cell leads in turn to a reversing of the beating direction of the cilia and the paramecium swims backward for a short time. After the cytoplasm returns to its normal state, the paramecium swims forward, changing its direction of movement. If the paramecium is touched from behind, the opening of ionic channels leads to a forward acceleration of the protozoan. In each case, the paramecium escapes its enemies [190].

From these humble origins, ionic channels in neurons have been perfected over millions of years of evolution. In the protoplasm of the cell, ionic channels are produced and replaced continually. They attach themselves to those regions of the neurons where they are needed and can move laterally in the membrane, like icebergs in the sea. The regions of increased neural sensitivity to the production of action potentials are thus changing continuously according to experience. The electrical properties of the cell membrane are not totally predetermined. They are also a result of the process by which action potentials are generated.

Consider also the interior of the neurons. The number of biochemical reaction chains and the complexity of the mechanical processes occurring in the neuron at any given time have led some authors to look for its *control system.* Stuart Hameroff, for example, has proposed that the cytoskeleton of neurons does not just perform a static mechanical function, but in some way provides the cell with feedback control. It is well known that the proteins that form the microtubules in axons coordinate to move synaptic vesicles and other materials from the cell body to the synapses. This is accomplished through a coordinated movement of the proteins, configured like a cellular automaton [173, 174].

Consequently, transmission, storage, and processing of information are performed by neurons exploiting many effects and mechanisms which we still do not understand fully. Each individual neuron is as complex or more complex than any of our computers. For this reason, we will call the elementary components of artificial neural networks simply "computing units" and not neurons. In the mid-1980s, the PDP (*Parallel Distributed Processing*) group already agreed to this convention at the insistence of Francis Crick [95].

1.3 Artificial neural networks

The discussion in the last section is only an example of how important it is to define the primitive functions and composition rules of the computational model. If we are computing with a conventional von Neumann processor, a minimal set of machine instructions is needed in order to implement all computable functions. In the case of artificial neural networks, the primitive functions are located in the nodes of the network and the composition rules are contained implicitly in the interconnection pattern of the nodes, in the synchrony or asynchrony of the transmission of information, and in the presence or absence of cycles.

1.3.1 Networks of primitive functions

Figure 1.14 shows the structure of an abstract neuron with n inputs. Each input channel i can transmit a real value x_i. The *primitive function* f computed in the body of the abstract neuron can be selected arbitrarily. Usually the input channels have an associated weight, which means that the incoming information x_i is multiplied by the corresponding weight w_i. The transmitted information is integrated at the neuron (usually just by adding the different signals) and the primitive function is then evaluated.

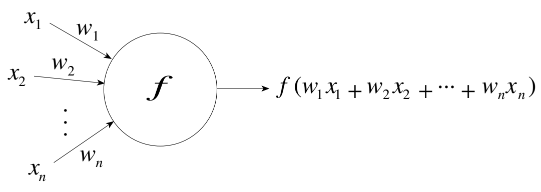


Fig. 1.14. An abstract neuron

If we conceive of each node in an artificial neural network as a primitive function capable of transforming its input in a precisely defined output, then artificial neural networks are nothing but *networks of primitive functions.* Different models of artificial neural networks differ mainly in the assumptions about the primitive functions used, the interconnection pattern, and the timing of the transmission of information.

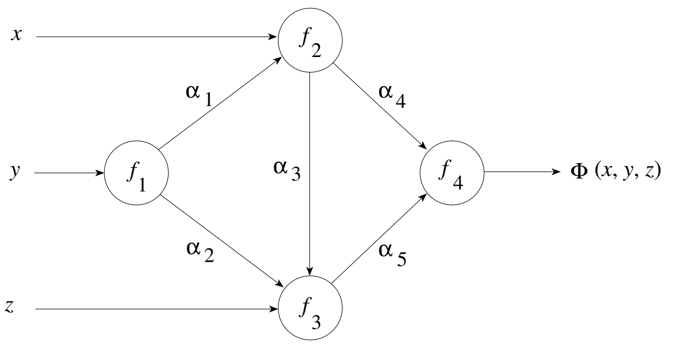


Fig. 1.15. Functional model of an artificial neural network

Typical artificial neural networks have the structure shown in Figure 1.15. The network can be thought of as a function Φ which is evaluated at the point (x, y, z). The nodes implement the primitive functions f_1, f_2, f_3, f_4 which are combined to produce Φ. The function Φ implemented by a neural network will be called the *network function*. Different selections of the weights $\alpha_1, \alpha_2, \ldots, \alpha_5$ produce different network functions. Therefore, tree elements are particularly important in any model of artificial neural networks:

- the structure of the nodes,
- the topology of the network,
- the learning algorithm used to find the weights of the network.

To emphasize our view of neural networks as networks of functions, the next section gives a short preview of some of the topics covered later in the book.

1.3.2 Approximation of functions

An old problem in approximation theory is to reproduce a given function $F : \mathbb{R} \to \mathbb{R}$ either exactly or approximately by evaluating a given set of primitive functions. A classical example is the approximation of one-dimensional functions using polynomials or Fourier series. The Taylor series for a function F which is being approximated around the point x_0 is

$$F(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)^2 + \cdots + a_n(x - x_0)^n + \cdots,$$

whereby the constants $a_0, ..., a_n$ depend on the function F and its derivatives at x_0. Figure 1.16 shows how the polynomial approximation can be represented as a network of functions. The primitive functions $z \mapsto 1, z \mapsto z^1, \ldots, z \mapsto z^n$ are computed at the nodes. The only free parameters are the constants $a_0, ..., a_n$. The output node additively collects all incoming information and produces the value of the evaluated polynomial. The weights of the network can be calculated in this case analytically, just by computing the first $n + 1$

terms of the Taylor series of F. They can also be computed using a learning algorithm, which is the usual case in the field of artificial neural networks.

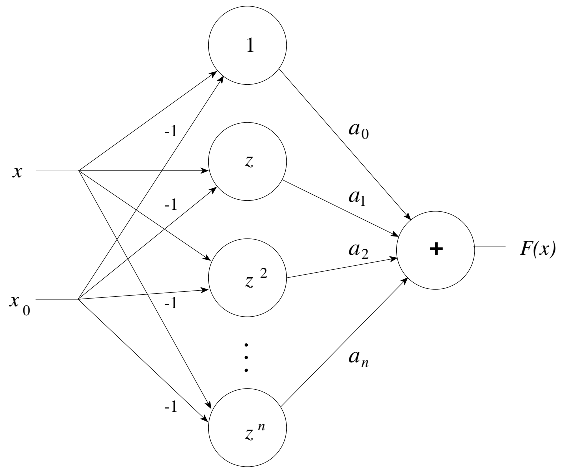


Fig. 1.16. A Taylor network

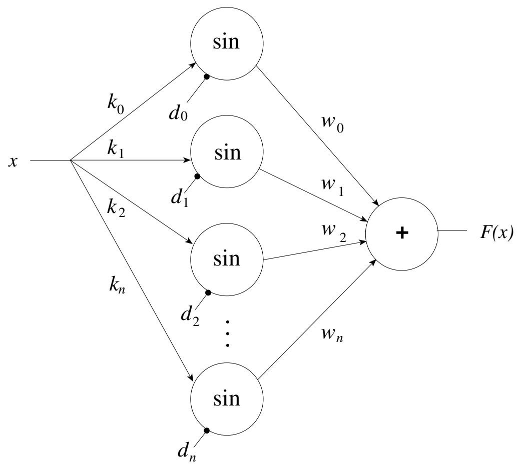


Fig. 1.17. A Fourier network

Figure 1.17 shows how a Fourier series can be implemented as a neural network. If the function F is to be developed as a Fourier series it has the form

$$F(x) = \sum_{i=0}^{\infty}(a_i \cos(ix) + b_i \sin(ix)). \tag{1.2}$$

An artificial neural network with the sine as primitive function can implement a finite number of terms in the above expression. In Figure 1.17 the constants $k_0, \ldots, k_n$ determine the wave numbers for the arguments of the sine functions. The constants $d_0, \ldots, d_n$ play the role of phase factors (with $d_0 = \pi/2$, for example, we have $\sin(x + d_0) = \cos(x)$) and we do not need to implement the cosine explicitly in the network. The constants $w_0, \ldots, w_n$ are the amplitudes of the Fourier terms. The network is indeed more general than the conventional formula because non-integer wave numbers are allowed as are phase factors which are not simple integer multiples of $\pi/2$.

The main difference between Taylor or Fourier series and artificial neural networks is, however, that the function F to be approximated is given not explicitly but implicitly through a set of input-output examples. We know F only at some points but we want to generalize as well as possible. This means that we try to adjust the parameters of the network in an optimal manner to reflect the information known and to extrapolate to new input patterns which will be shown to the network afterwards. This is the task of the *learning algorithm* used to adjust the network's parameters.

These two simple examples show that neural networks can be used as universal function approximators, that is, as computing models capable of approximating a given set of functions (usually the integrable functions). We will come back to this problem in Chap. 10.

1.3.3 Caveat

At this point we must issue a warning to the reader: in the theory of artificial neural networks we do not consider the whole complexity of real biological neurons. We only abstract some general principles and content ourselves with different levels of detail when simulating neural ensembles. The general approach is to conceive each neuron as a primitive function producing numerical results at some points in time. These will be the kinds of model that we will discuss in the first chapters of this book. However we can also think of artificial neurons as computing units which produce pulse trains in the way that biological neurons do. We can then simulate this behavior and look at the output of simple networks. This kind of approach, although more closely related to the biological paradigm, is still a very rough approximation of the biological processes. We will deal with asynchronous and spiking neurons in later chapters.

1.4 Historical and bibliographical remarks

Philosophical reflection on consciousness and the organ in which it could possibly be localized spans a period of more than two thousand years. Greek philosophers were among the first to speculate about the location of the soul. Several theories were held by the various philosophical schools of ancient times. Galenus, for example, identified nerve impulses with pneumatic pressure signals and conceived the nervous system as a pneumatic machine. Several centuries later Newton speculated that nerves transmitted oscillations of the ether.

Our present knowledge of the structure and physiology of neurons is the result of 100 years of special research in this field. The facts presented in this chapter were discovered between 1850 and 1950, with the exception of the NMDA receptors which were studied mainly in the last decade. The electrical nature of nerve impulses was postulated around 1850 by Emil du Bois-Reymond and Hermann von Helmholtz. The latter was able to measure the velocity of nerve impulses and showed that it was not as fast as was previously thought. Signals can be transmitted in both directions of an axon, but around 1901 Santiago Ramón y Cajal postulated that the specific networking of the nervous cells determines a direction for the transmission of information. This discovery made it clear that the coupling of the neurons constitutes a hierarchical system.

Ramón y Cajal was also the most celebrated advocate of the *neuron theory.* His supporters conceived the brain as a highly differentiated hierarchical organ, while the supporters of the reticular theory thought of the brain as a grid of undifferentiated axons and of dendrites as organs for the nutrition of the cell [357]. Ramón y Cajal perfected Golgi's staining method and published the best diagrams of neurons of his time, so good indeed that they are still in use. The word *neuron* (Greek for *nerve*) was proposed by the Berlin Professor Wilhelm Waldeger after he saw the preparations of Ramón y Cajal [418].

The chemical transmission of information at the synapses was studied from 1920 to 1940. From 1949 to 1956, Hodgkin and Huxley explained the mechanism by which depolarization waves are produced in the cell membrane. By experimenting with the giant axon of the squid they measured and explained the exchange of ions through the cell membrane, which in time led to the now famous Hodgkin–Huxley differential equations. For a mathematical treatment of this system of equations see [97].

The Hodgkin–Huxley model was in some ways one of the first artificial neural models, because the postulated dynamics of the nerve impulses could be simulated with simple electric networks [303]. At the same time the mathematical properties of artificial neural networks were being studied by researchers like Warren McCulloch, Walter Pitts, and John von Neumann. Ever since that time, research in the neurobiological field has progressed in close collaboration with the mathematics and computer science community.

Exercises

1. Express the network function function Φ in Figure 1.15 in terms of the primitive functions $f_1, \ldots, f_4$ and of the weights $\alpha_1, \ldots, \alpha_5$.

2. Modify the network of Figure 1.17 so that it corresponds to a finite number of addition terms of equation (1.2).
3. Look in a neurobiology book for the full set of differential equations of the Hodgkin–Huxley model. Write a computer program that simulates an action potential.

2 Threshold Logic

2.1 Networks of functions

We deal in this chapter with the simplest kind of computing units used to build artificial neural networks. These computing elements are a generalization of the common logic gates used in conventional computing and, since they operate by comparing their total input with a threshold, this field of research is known as *threshold logic*.

2.1.1 Feed-forward and recurrent networks

Our review in the previous chapter of the characteristics and structure of biological neural networks provides us with the initial motivation for a deeper inquiry into the properties of networks of abstract neurons. From the viewpoint of the engineer, it is important to define how a network should behave, without having to specify completely all of its parameters, which are to be found in a learning process. Artificial neural networks are used in many cases as a *black box*: a certain input should produce a desired output, but how the network achieves this result is left to a self-organizing process.

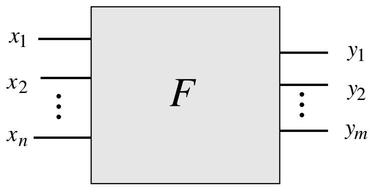


Fig. 2.1. A neural network as a black box

In general we are interested in mapping an n-dimensional real input $(x_1, x_2, \ldots, x_n)$ to an m-dimensional real output $(y_1, y_2, \ldots, y_m)$. A neural network thus behaves as a "mapping machine", capable of modeling a function $F : \mathbb{R}^n \to \mathbb{R}^m$. If we look at the structure of the network being used, some aspects of its dynamics must be defined more precisely. When the function is evaluated with a network of primitive functions, information flows through the directed edges of the network. Some nodes compute values which are then transmitted as arguments for new computations. If there are no cycles in the network, the result of the whole computation is well-defined and we do not have to deal with the task of synchronizing the computing units. We just assume that the computations take place without delay.

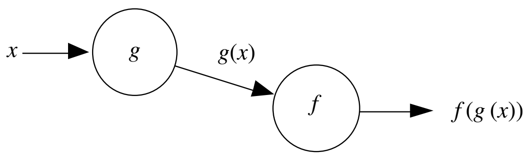


Fig. 2.2. Function composition

If the network contains cycles, however, the computation is not uniquely defined by the interconnection pattern and the temporal dimension must be considered. When the output of a unit is fed back to the same unit, we are dealing with a recursive computation without an explicit halting condition. We must define what we expect from the network: is the fixed point of the recursive evaluation the desired result or one of the intermediate computations? To solve this problem we assume that every computation takes a certain amount of time at each node (for example a time unit). If the arguments for a unit have been transmitted at time t, its output will be produced at time $t + 1$. A recursive computation can be stopped after a certain number of steps and the last computed output taken as the result of the recursive computation.

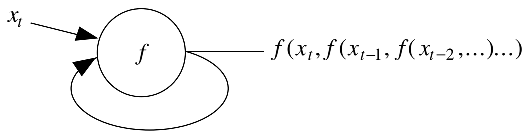


Fig. 2.3. Recursive evaluation

In this chapter we deal first with networks without cycles, in which the time dimension can be disregarded. Then we deal with recurrent networks and their temporal coordination. The first model we consider was proposed in 1943 by Warren McCulloch and Walter Pitts. Inspired by neurobiology they put forward a model of computation oriented towards the computational capabilities of real neurons and studied the question of abstracting universal concepts from specific perceptions [299].

We will avoid giving a general definition of a *neural network* at this point. So many models have been proposed which differ in so many respects that any definition trying to encompass this variety would be unnecessarily clumsy. As we show in this chapter, it is not necessary to start building neural networks with "high powered" computing units, as some authors do [384]. We will start our investigations with the general notion that a neural network is a *network of functions* in which synchronization can be considered explicitly or not.

2.1.2 The computing units

The nodes of the networks we consider will be called *computing elements* or simply *units*. We assume that the edges of the network transmit information in a predetermined direction and the number of incoming edges into a node is not restricted by some upper bound. This is called the *unlimited fan-in* property of our computing units.

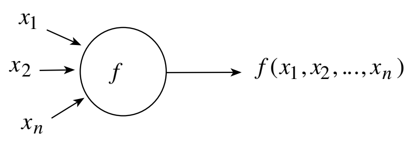


Fig. 2.4. Evaluation of a function of n arguments

The primitive function computed at each node is in general a function of n arguments. Normally, however, we try to use very simple primitive functions of one argument at the nodes. This means that the incoming n arguments have to be reduced to a single numerical value. Therefore computing units are split into two functional parts: an integration function g reduces the n arguments to a single value and the output or activation function f produces the output of this node taking that single value as its argument. Figure 2.5 shows this general structure of the computing units. Usually the integration function g is the addition function.

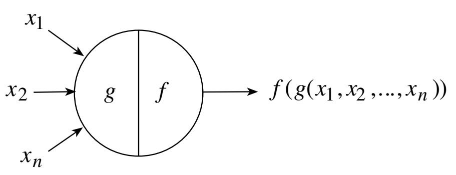


Fig. 2.5. Generic computing unit

McCulloch–Pitts networks are even simpler than this, because they use solely binary signals, i.e., ones or zeros. The nodes produce only binary results and the edges transmit exclusively ones or zeros. The networks are composed of directed unweighted edges of *excitatory* or of *inhibitory* type. The latter are marked in diagrams using a small circle attached to the end of the edge. Each McCulloch–Pitts unit is also provided with a certain threshold value θ.

At first sight the McCulloch–Pitts model seems very limited, since only binary information can be produced and transmitted, but it already contains all necessary features to implement the more complex models. Figure 2.6 shows an abstract McCulloch–Pitts computing unit. Following Minsky [311] it will be represented as a circle with a black half. Incoming edges arrive at the white half, outgoing edges leave from the black half. Outgoing edges can fan out any number of times.

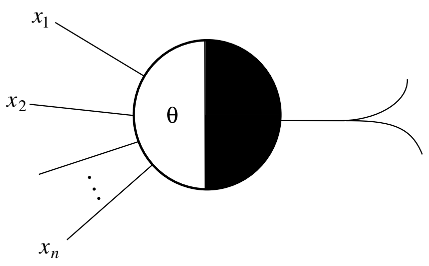


Fig. 2.6. Diagram of a McCulloch–Pitts unit

The rule for evaluating the input to a McCulloch–Pitts unit is the following:

- Assume that a McCulloch–Pitts unit gets an input $x_1, x_2, \ldots, x_n$ through n excitatory edges and an input $y_1, y_2, \ldots, y_m$ through m inhibitory edges.
- If $m \geq 1$ and at least one of the signals $y_1, y_2, \ldots, y_m$ is 1, the unit is inhibited and the result of the computation is 0.
- Otherwise the total excitation $x = x_1 + x_2 + \cdots + x_n$ is computed and compared with the threshold θ of the unit (if $n = 0$ then $x = 0$). If $x \geq \theta$ the unit *fires* a 1, if $x < \theta$ the result of the computation is 0.

This rule implies that a McCulloch–Pitts unit can be inactivated by a single inhibitory signal, as is the case with some real neurons. When no inhibitory signals are present, the units act as a *threshold gate* capable of implementing many other logical functions of n arguments.

Figure 2.7 shows the activation function of a unit, the so-called step function. This function changes discontinuously from zero to one at θ. When θ is zero and no inhibitory signals are present, we have the case of a unit producing the constant output one. If θ is greater than the number of incoming excitatory edges, the unit will never fire.

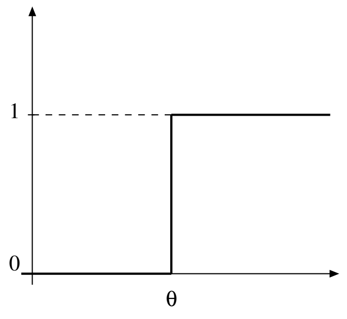


Fig. 2.7. The step function with threshold θ

In the following subsection we assume provisionally that there is no delay in the computation of the output.

2.2 Synthesis of Boolean functions

The power of threshold gates of the McCulloch–Pitts type can be illustrated by showing how to synthesize any given logical function of n arguments. We deal firstly with the more simple kind of logic gates.

2.2.1 Conjunction, disjunction, negation

Mappings from $\{0,1\}^n$ onto $\{0,1\}$ are called logical or Boolean functions. Simple logical functions can be implemented directly with a single McCulloch–Pitts unit. The output value 1 can be associated with the logical value *true* and 0 with the logical value *false*. It is straightforward to verify that the two units of Figure 2.8 compute the functions AND and OR respectively.

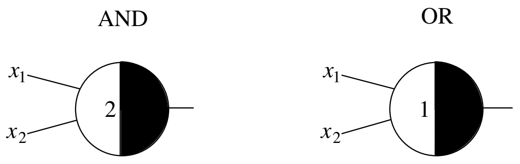


Fig. 2.8. Implementation of AND and OR gates

A single unit can compute the disjunction or the conjunction of n arguments as is shown in Figure 2.9, where the conjunction of three and four arguments is computed by two units. The same kind of computation requires several conventional logic gates with two inputs. It should be clear from this simple example that threshold logic elements can reduce the complexity of the circuit used to implement a given logical function.

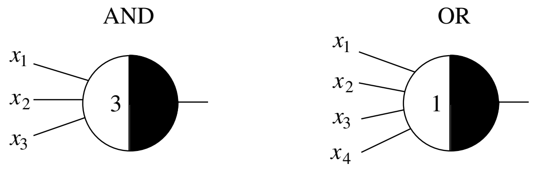


Fig. 2.9. Generalized AND and OR gates

As is well known, AND and OR gates alone cannot be combined to produce all logical functions of n variables. Since uninhibited threshold logic elements are capable of implementing more general functions than conventional AND or OR gates, the question of whether they can be combined to produce all logical functions arises. Stated another way: is inhibition of McCulloch–Pitts units necessary or can it be dispensed with? The following proposition shows that it is necessary. A monotonic logical function f of n arguments is one whose value at two given n-dimensional points $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$ is such that $f(x) \geq f(y)$ whenever the number of ones in the input y is a subset of the ones in the input x. An example of a non-monotonic logical function of one argument is logical negation.

Proposition 1. *Uninhibited threshold logic elements of the McCulloch–Pitts type can only implement monotonic logical functions.*

Proof. An example shows the kind of argumentation needed. Assume that the input vector $(1, 1, \ldots, 1)$ is assigned the function value 0. Since no other vector can set more edges in the network to 1 than this vector does, any other input vector can also only be evaluated to 0. In general, if the ones in the input vector y are a subset of the ones in the input vector x, then the first cannot set more edges to 1 than x does. This implies that $f(x) \geq f(y)$, as had to be shown. □

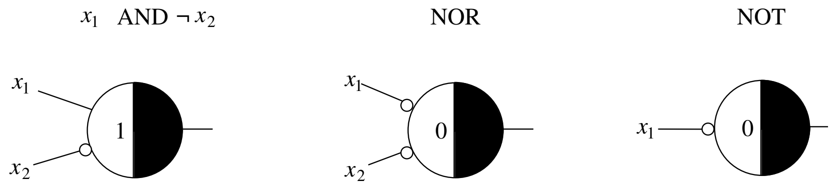


Fig. 2.10. Logical functions and their realization

The units of Figure 2.10 show the implementation of some non-monotonic logical functions requiring inhibitory connections. Logical negation, for example, can be computed using a McCulloch–Pitts unit with threshold 0 and an inhibitory edge. The other two functions can be verified by the reader.

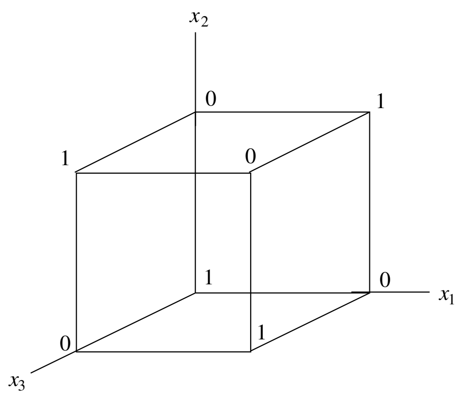


Fig. 2.11. Function values of a logical function of three variables

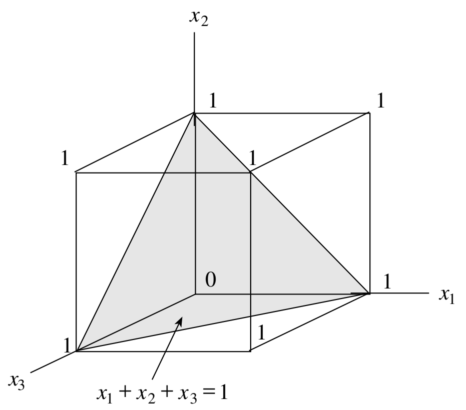


Fig. 2.12. Separation of the input space for the OR function

2.2.2 Geometric interpretation

It is very instructive to visualize the kind of functions that can be computed with McCulloch–Pitts cells by using a diagram. Figure 2.11 shows the eight vertices of a three-dimensional unit cube. Each of the three logical variables x_1, x_2 and x_3 can assume one of two possible binary values. There are eight possible combinations, represented by the vertices of the cube. A logical function is just an assignment of a 0 or a 1 to each of the vertices. The figure shows one of these assignments. In the case of n variables, the cube consists of 2^n vertices and admits 2^{2^n} different binary assignments.

McCulloch–Pitts units divide the input space into two half-spaces. For a given input (x_1, x_2, x_3) and a threshold θ the condition $x_1 + x_2 + x_3 \geq \theta$ is tested, which is true for all points to one side of the plane with the equation $x_1 + x_2 + x_3 = \theta$ and false for all points to the other side (without including the plane itself in this case). Figure 2.12 shows this separation for the case in

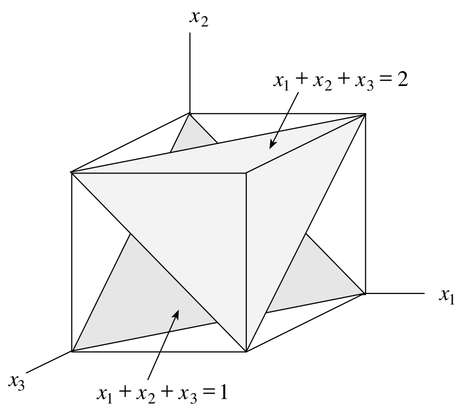


Fig. 2.13. Separating planes of the OR and majority functions

which $\theta = 1$, i.e., for the OR function. Only those vertices above the separating plane are labeled 1.

The majority function of three variables divides input space in a similar manner, but the separating plane is given by the equation $x_1 + x_2 + x_3 = 2$. Figure 2.13 shows the additional plane. The planes are always parallel in the case of McCulloch–Pitts units. Non-parallel separating planes can only be produced using weighted edges.

2.2.3 Constructive synthesis

Every logical function of n variables can be written in tabular form. The value of the function is written down for every one of the possible binary combinations of the n inputs. If we want to build a network to compute this function, it should have n inputs and one output. The network must associate each input vector with the correct output value. If the number of computing units is not limited in some way, it is always possible to build or synthesize a network which computes this function. The constructive proof of this proposition profits from the fact that McCulloch–Pitts units can be used as binary decoders.

Consider for example the vector $(1, 0, 1)$. It is the only one which fulfills the condition $x_1 \wedge \neg x_2 \wedge x_3$. This condition can be tested by a single computing unit (Figure 2.14). Since only the vector $(1, 0, 1)$ makes this unit fire, the unit is a decoder for this input.

Assume that a function F of three arguments has been defined according to the following table:

To compute this function it is only necessary to decode all those vectors for which the function's value is 1. Figure 2.15 shows a network capable of computing the function F.

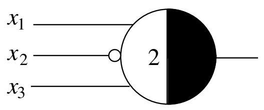


Fig. 2.14. Decoder for the vector $(1, 0, 1)$

input vectors	F
(0,0,1)	1
(0,1,0)	1
all others	0

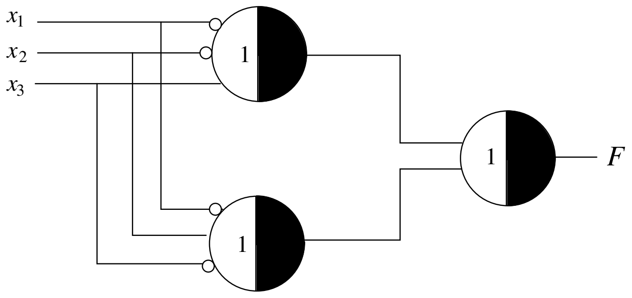


Fig. 2.15. Synthesis of the function F

The individual units in the first layer of the composite network are decoders. For each vector for which F is 1 a decoder is used. In our case we need just two decoders. Components of each vector which must be 0 are transmitted with inhibitory edges, components which must be 1 with excitatory ones. The threshold of each unit is equal to the number of bits equal to 1 that must be present in the desired input vector. The last unit to the right is a disjunction: if any one of the specified vectors can be decoded this unit fires a 1.

It is straightforward to extend this constructive method to other Boolean functions of any other dimension. This leads to the following proposition:

Proposition 2. *Any logical function* $F : \{0,1\}^n \rightarrow \{0,1\}$ *can be computed with a McCulloch–Pitts network of two layers.*

No attempt has been made here to minimize the number of computing units. In fact, we need as many decoders as there are ones in the table of function values. An alternative to this simple constructive method is to use harmonic analysis of logical functions, as will be shown in Sect. 2.5.

We can also consider the minimal possible set of building blocks needed to implement arbitrary logical functions when the fan-in of the units is bounded in some way. The circuits of Figure 2.14 and Figure 2.15 use decoders of n inputs. These decoders can be built of simpler cells, for example, two units capable of respectively implementing the AND function and negation. Inhibitory connections in the decoders can be replaced with a negation gate. The output of the decoders is collected at a conjunctive unit. The decoder of Figure 2.14 can be implemented as shown in Figure 2.16. The only difference from the previous decoder are the negated inputs and the higher threshold in the AND unit. All decoders for a row of the table of a logical function can be designed in a similar way. This immediately leads to the following proposition:

Proposition 3. *All logical functions can be implemented with a network composed of units which exclusively compute the AND, OR, and NOT functions.*

The three units AND, NOT and OR are called a logical basis because of this property. Since OR can be implemented using AND and NOT units, these two alone constitute a logical basis. The same happens with OR and NOT units. John von Neumann showed that through a redundant coding of the inputs (each variable is transmitted through two lines) AND and OR units alone can constitute a logical basis [326].

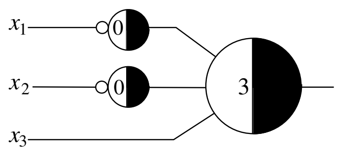


Fig. 2.16. A composite decoder for the vector $(0, 0, 1)$

2.3 Equivalent networks

We can build simpler circuits by using units with more general properties, for example weighted edges and relative inhibition. However, as we show in this section, circuits of McCulloch–Pitts units can emulate circuits built out of high-powered units by exploiting the trade-off between the complexity of the network versus the complexity of the computing units.

2.3.1 Weighted and unweighted networks

Since McCulloch–Pitts networks do not use weighted edges the question of whether weighted networks are more general than unweighted ones must be answered. A simple example shows that both kinds of networks are equivalent.

Assume that three weighted edges converge on the unit shown in Figure 2.17. The unit computes

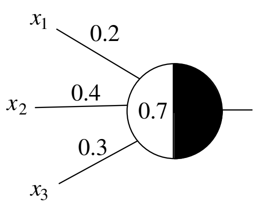


Fig. 2.17. Weighted unit

$$0.2x_1 + 0.4x_2 + 0.3x_3 \geq 0.7.$$

But this is equivalent to

$$2x_1 + 4x_2 + 3x_3 \geq 7,$$

and this computation can be performed with the network of Figure 2.18.

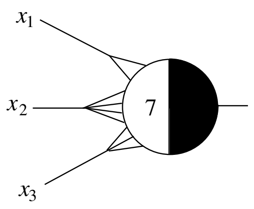


Fig. 2.18. Equivalent computing unit

The figure shows that positive rational weights can be simulated by simply fanning-out the edges of the network the required number of times. This means that we can either use weighted edges or go for a more complex topology of the network, with many redundant edges. The same can be done in the case of irrational weights if the number of input vectors is finite (see Chap. 3, Exercise 3).

2.3.2 Absolute and relative inhibition

In the last subsection we dealt only with the case of positive weights. Two classes of inhibition can be identified: *absolute* inhibition corresponds to the one used in McCulloch–Pitts units. *Relative* inhibition corresponds to the case of edges weighted with a negative factor and whose effect is to lower the firing threshold when a 1 is transmitted through this edge.

Proposition 4. *Networks of McCulloch–Pitts units are equivalent to networks with relative inhibition.*

Proof. It is only necessary to show that each unit in a network where relative inhibition is used is equivalent to one or more units in a network where absolute inhibition is used. It is clear that it is possible to implement absolute inhibition with relative inhibitory edges. If the threshold of a unit is the integer m and if n excitatory edges impinge on it, the maximum possible total excitation for this unit is $n - m$. If $m \geq n$ the unit never fires and the inhibitory edge is irrelevant. It suffices to fan out the inhibitory edge $n - m + 1$ times and make all these edges meet at the unit. When a 1 is transmitted through the inhibitory edges the total amount of inhibition is $n - m + 1$ and this shuts down the unit. To prove that relative inhibitory edges can be simulated with absolute inhibitory ones, refer to Figure 2.19. The network to the left contains a relative inhibitory edge, the network to the right absolute inhibitory ones. The reader can verify that the two networks are equivalent. Relative inhibitory edges correspond to edges weighted with -1. We can also accept any other negative weight w. In that case the threshold of the unit to the right of Figure 2.19 should be $m + w$ instead of $m + 1$. Therefore networks with negative weights can be simulated using unweighted McCulloch–Pitts elements. □

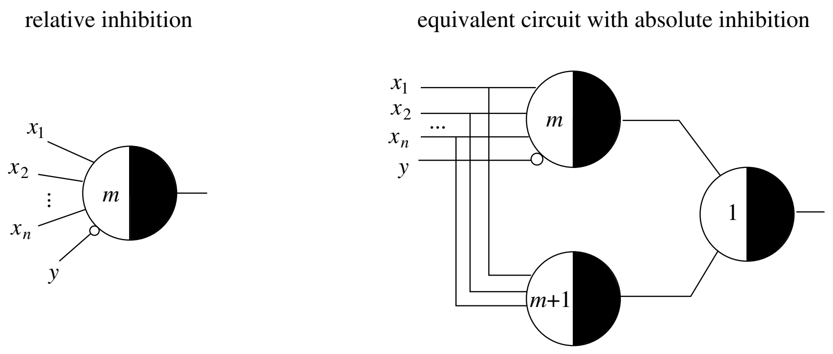


Fig. 2.19. Two equivalent networks

As shown above, we can implement any kind of logical function using unweighted networks. What we trade is the simplicity of the building blocks for a more convoluted topology of the network. Later we will always use weighted networks in order to simplify the topology.

2.3.3 Binary signals and pulse coding

An additional question which can be raised is whether binary signals are not a very limited coding strategy. Are networks in which the communication channels adopt any of ten or fifteen different states more efficient than channels which adopt only two states, as in McCulloch–Pitts networks? To give an answer we must consider that unit states have a price, in biological networks as well as in artificial ones. The transmitted information must be optimized using the number of available switching states.

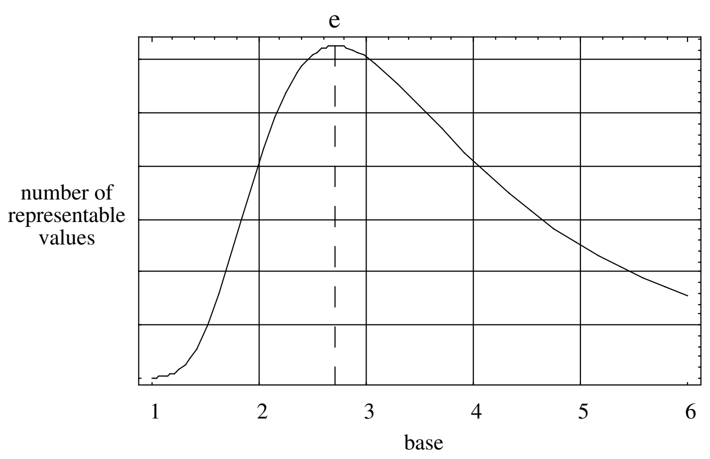


Fig. 2.20. Number of representable values as a function of the base

Assume that the number of states per communication channel is b and that c channels are used to input information. The cost K of the implementation is proportional to both quantities, i.e., $K = \gamma bc$, where γ is a proportionality constant. Using c channels with b states, b^c different numbers can be represented. This means that $c = K/\gamma b$ and, if we set $\kappa = K/\gamma$, we are seeking the numerical base b which optimizes the function $b^{\kappa/b}$. Since we assume constant cost, κ is a constant. Figure 2.20 shows that the optimal value for b is the Euler constant e. Since the number of channel states must be an integer, three states would provide a good approximation to the optimal coding strategy. However, in electronic and biological systems decoding of the signal plays such an important role that the choice of two states per channel becomes a better alternative.

Wiener arrived at a similar conclusion through a somewhat different argument [452]. The binary nature of information transmission in the nervous system seems to be an efficient way to transport signals. However, in the next chapters we will assume that the communication channels can transport *arbitrary real numbers*. This makes the analysis simpler than when we have to deal explicitly with frequency modulated signals, but does not lead to a minimization of the resources needed for a technical implementation. Some researchers prefer to work with so-called *weightless networks* which operate exclusively with binary data.

2.4 Recurrent networks

We have already shown that feed-forward networks can implement arbitrary logical functions. In this case the dimension of the input and output data is predetermined. In many cases, though, we want to perform computations on an input of variable length, for example, when adding two binary numbers being fed bit for bit into a network, which in turn produces the bits of the result one after the other. A feed-forward network cannot solve this problem because it is not capable of keeping track of previous results and, in the case of addition, the carry bit must be stored in order to be reused. This kind of problem can be solved using recurrent networks, i.e., networks whose partial computations are recycled through the network itself. Cycles in the topology of the network make storage and reuse of signals possible for a certain amount of time after they are produced.

2.4.1 Stored state networks

McCulloch–Pitts units can be used in recurrent networks by introducing a temporal factor in the computation. We will assume that computation of the activation of each unit consumes a time unit. If the input arrives at time t the result is produced at time $t+1$. Up to now, we have been working with units which produce results without delay. The numerical capabilities of any feed-forward network with instantaneous computation at the nodes can be reproduced by networks of units with delay. We only have to take care to coordinate the arrival of the input values at the nodes. This could make the introduction of additional computing elements necessary, whose sole mission is to insert the necessary delays for the coordinated arrival of information. This is the same problem that any computer with clocked elements has to deal with.

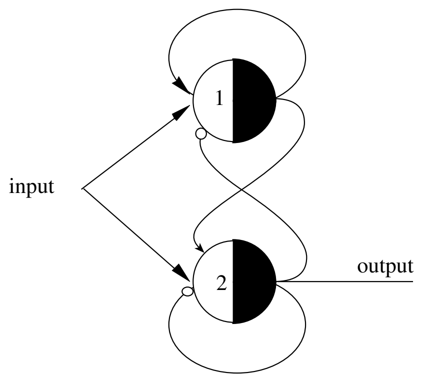


Fig. 2.21. Network for a binary scaler

Figure 2.21 shows a simple example of a recurrent circuit. The network processes a sequence of bits, giving off one bit of output for every bit of input, but in such a way that any two consecutive ones are transformed into the sequence 10. The binary sequence 00110110 is transformed for example into the sequence 00100100. The network recognizes only two consecutive ones separated by at least a zero from a similar block.

2.4.2 Finite automata

The network discussed in the previous subsection is an example of an automaton. This is an abstract device capable of assuming different states which change according to the received input. The automaton also produces an output according to its momentary state. In the previous example, the state of the automaton is the specific combination of signals circulating in the network at any given time. The set of possible states corresponds to the set of all possible combinations of signals traveling through the network.

state transitions

		state	
		Q_0	Q_1
input	0	Q_0	Q_0
	1	Q_1	Q_1

output table

		state	
		Q_0	Q_1
input	0	0	1
	1	0	1

Fig. 2.22. State tables for a binary delay

Finite automata can take only a finite set of possible states and can react only to a finite set of input signals. The state transitions and the output of an automaton can be specified with a table, like the one shown in Figure 2.22. This table defines an automaton which accepts a binary signal at time t and produces an output at time $t+1$. The automaton has two states, Q_0 and Q_1, and accepts only the values 0 or 1. The first table shows the state transitions, corresponding to each input and each state. The second table shows the output values corresponding to the given state and input. From the table we can see that the automaton switches from state Q_0 to state Q_1 after accepting the input 1. If the input bit is a 0, the automaton remains in state Q_0. If the state of the automaton is Q_1 the output at time $t+1$ is 1 regardless of whether 1 or 0 was given as input at time t. All other possibilities are covered by the rest of the entries in the two tables.

The diagram in Figure 2.23 shows how the automaton works. The values at the beginning of the arrows represent an input bit for the automaton. The values in the middle of the arrows are the output bits produced after each new input. An input of 1, for example, produces the transition from state Q_0 to state Q_1 and the output 0. The input 0 produces a transition to state Q_0. The automaton is thus one that stores only the last bit of input in its current state.

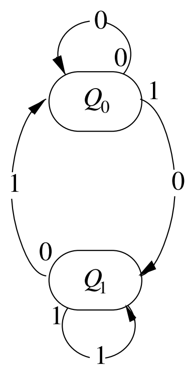


Fig. 2.23. Diagram of a finite automaton

Finite automata without input from the outside, i.e., free-wheeling automata, unavoidably fall in an infinite loop or reach a final constant state. This is why finite automata cannot cover all computable functions, for whose computation an infinite number of states are needed. A Turing machine achieves this through an infinite storage band which provides enough space for all computations. Even a simple problem like the multiplication of two arbitrary binary numbers presented sequentially cannot be solved by a finite automaton. Although our computers are finite automata, the number of possible states is so large that we consider them as universal computing devices for all practical purposes.

2.4.3 Finite automata and recurrent networks

We now show that finite automata and recurrent networks of McCulloch–Pitts units are equivalent. We use a variation of a constructive proof due to Minsky [311].

Proposition 5. *Any finite automaton can be simulated with a network of McCulloch–Pitts units.*

Proof. Figure 2.24 is a diagram of the network needed for the proof. Assume that the input signals are transmitted through the input lines I_1 to I_m and at each moment t only one of these lines is conducting a 1. All other input lines are passive (set to 0). Assume that the network starts in a well-defined

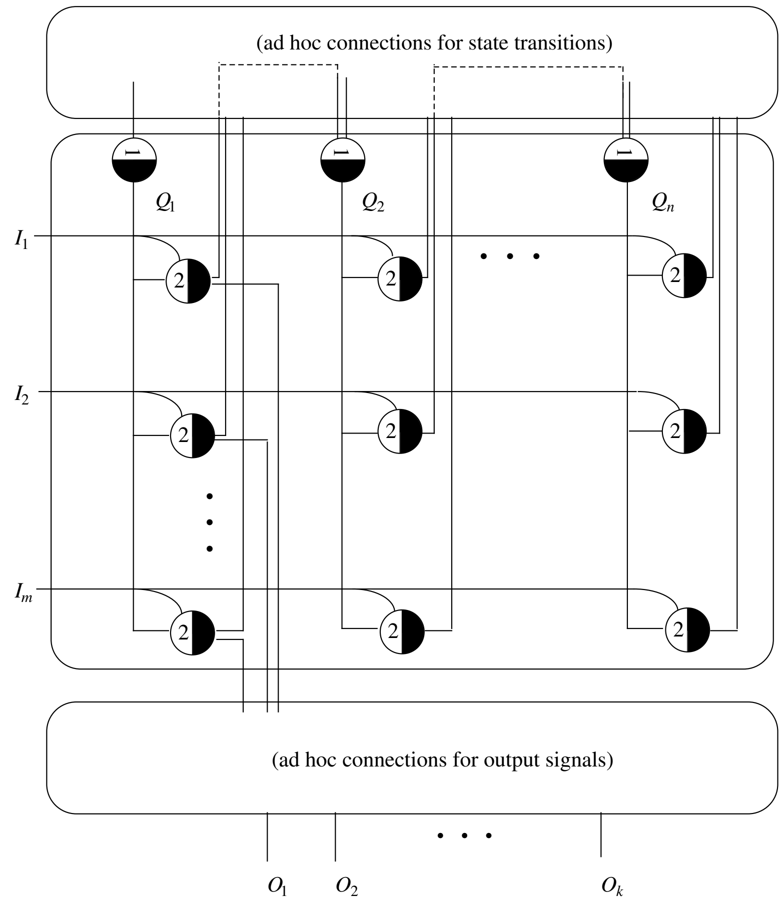

(ad hoc connections for state transitions)
1
1
1
Q_1
Q_2
Q_n
I_1
I_2
I_m
2
(ad hoc connections for output signals)
O_1
O_2
O_k

Fig. 2.24. Implementation of a finite automaton with McCulloch–Pitts units

state Q_i. This means that one, and only one, of the lines labeled $Q_1, \ldots, Q_n$ is set to 1 and the others to 0. At time $t + 1$ only one of the AND units can produce a 1, namely the one in which both input and state line are set to 1. The state transition is controlled by the ad hoc connections defined by the user in the upper box. If, for example, the input I_1 and the state Q_1 at time t produce the transition to state Q_2 at time $t + 1$, then we have to connect the output of the upper left AND unit to the input of the OR unit with the output line named Q_2 (dotted line in the diagram). This output will become active at time $t + 2$. At this stage a new input line must be set to 1 (for example I_2) and a new state transition will be computed (Q_n in our example). The connections required to produce the desired output are defined in a similar way. This can be controlled by connecting the output of each AND unit to the corresponding output line $O_1, \ldots, O_k$ using a box of ad hoc connections similar to the one already described. □

A disadvantage of this constructive method is that each simulated finite automaton requires a special set of connections in the upper and lower boxes. It is better to define a universal network capable of simulating any other finite automaton without having to change the topology of the network (under the assumption of an upper bound for the size of the simulated automata). This is indeed an active field of research in which networks learn to simulate automata [408]. The necessary network parameters are found by a learning algorithm. In the case of McCulloch–Pitts units the available degrees of freedom are given by the topology of the network.

2.4.4 A first classification of neural networks

The networks described in this chapter allow us to propose a preliminary taxonomy of the networks we will discuss in this book. The first clear separation line runs between weighted and unweighted networks. It has already been shown that both classes of models are equivalent. The main difference is the kind of learning algorithm that can be used. In unweighted networks only the thresholds and the connectivity can be adapted. In weighted networks the topology is not usually modified during learning (although we will see some algorithms capable of doing this) and only an optimal combination of weights is sought.

The second clear separation is between synchronous and asynchronous models. In synchronous models the output of all elements is computed instantaneously. This is always possible if the topology of the network does not contain cycles. In some cases the models contain layers of computing units and the activity of the units in each layer is computed one after the other, but in each layer simultaneously. Asynchronous models compute the activity of each unit independently of all others and at different stochastically selected times (as in Hopfield networks). In these kinds of models, cycles in the underlying connection graph pose no particular problem.

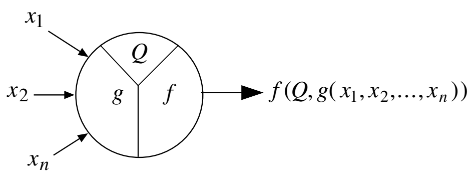


Fig. 2.25. A unit with stored state Q

Finally, we can distinguish between models with or without stored unit states. In Figure 2.5 we gave an example of a unit without stored state. Figure 2.25 shows a unit in which a state Q is stored after each computation.

The state Q can modify the output of the unit in the following activation. If the number of states and possible inputs is finite, we are dealing with a finite automaton. Since any finite automaton can be simulated by a network of computing elements without memory, these units with a stored state can be substituted by a network of McCulloch–Pitts units. Networks with stored-state units are thus equivalent to networks *without* stored-state units. Data is stored in the network itself and in its pattern of recursion.

It can be also shown that time varying weights and thresholds can be implemented in a network of McCulloch–Pitts units using cycles, so that networks with time varying weights and thresholds are equivalent to networks with constant parameters, whenever recursion is allowed.

2.5 Harmonic analysis of logical functions

An interesting alternative for the automatic synthesis of logic functions and for a quantification of their implementation complexity is to do an analysis of the distribution of its non-zero values using the tools of harmonic analysis. Since we can tabulate the values of a logical function in a sequence, we can think of it as a one-dimensional function whose fundamental "frequencies" can be extracted using the appropriate mathematical tools. We will first deal with this problem in a totally general setting and then show that the Hadamard–Walsh transform is the tool we are looking for.

2.5.1 General expression

Assume that we are interested in expressing a function $f : \mathbb{R}^m \to \mathbb{R}$ as a linear combination of n functions $f_1, f_2, \ldots, f_n$ using the n constants $a_1, a_2, \ldots, a_n$ in the following form

$$f = a_1 f_1 + a_2 f_2 + \cdots + a_n f_n.$$

The domain and range of definition are the same for f and the base functions.

We can determine the quadratic error E of the approximation in the whole domain of definition V for given constants $a_1, \ldots, a_n$ by computing

$$E = \int_V (f - (a_1 f_1 + a_2 f_2 + \cdots + a_n f_n))^2 dV.$$

Here we are assuming that f and the functions f_i, $i = 1, \ldots, n$, are integrable in its domain of definition V. Since we want to minimize the quadratic error E we compute the partial derivatives of E with respect to $a_1, a_2, \ldots, a_n$ and set them to zero:

$$\frac{dE}{da_i} = -2 \int_V f_i (f - a_1 f_1 - a_2 f_2 - \cdots - a_n f_n) dV = 0, \text{ for } i = 1, \ldots, n$$

This leads to the following set of n equations expressed in a simplified notation:

$$a_1 \int f_i f_1 + a_2 \int f_i f_2 + \cdots + a_n \int f_i f_n = \int f_i f, \text{ for } i = 1, \ldots, n.$$

Expressed in matrix form the set of equations becomes:

$$\begin{pmatrix} \int f_1 f_1 & \int f_1 f_2 & \cdots & \int f_1 f_n \\ \int f_2 f_1 & \int f_2 f_2 & & \int f_2 f_n \\ \vdots & & \ddots & \vdots \\ \int f_n f_1 & \int f_n f_2 & \cdots & \int f_n f_n \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} \int f_1 f \\ \int f_2 f \\ \vdots \\ \int f_n f \end{pmatrix}$$

This expression is very general. The only assumption we have used is the integrability of the partial products of the form $f_i f_j$ and $f_i f$. Since no special assumptions on the integral were used, it is also possible to use a discrete version of this equation. Assume that the function f has been defined at m points and let the symbol $\sum f_i f_j$ stand for $\sum_{k=1}^m f_i(x_k) f_j(x_k)$. In this case the above expression transforms to

$$\begin{pmatrix} \sum f_1 f_1 & \sum f_1 f_2 & \cdots & \sum f_1 f_n \\ \sum f_2 f_1 & \sum f_2 f_2 & & \sum f_2 f_n \\ \vdots & & \ddots & \vdots \\ \sum f_n f_1 & \sum f_n f_2 & \cdots & \sum f_n f_n \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} \sum f_1 f \\ \sum f_2 f \\ \vdots \\ \sum f_n f \end{pmatrix}$$

The general formula for the polynomial approximation of m data points $(x_1, y_1), \ldots, (x_m, y_m)$ using the primitive functions $x^0, x^1, x^2, \ldots, x^{n-1}$ translates directly into

$$\begin{pmatrix} m & \sum x_i & \cdots & \sum x_i^{n-1} \\ \sum x_i & \sum x_i^2 & & \sum x_i^n \\ \vdots & & \ddots & \vdots \\ \sum x_i^{n-1} & \sum x_i^n & \cdots & \sum x_i^{2n-2} \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} \sum y_i \\ \sum x_i y_i \\ \vdots \\ \sum x_i^{n-1} y_i \end{pmatrix}$$

In the case of base functions that are mutually orthogonal, the integrals $\int f_k f_j$ vanish when $k \neq j$. In this case the $n \times n$ matrix is diagonal and the above equation becomes very simple. Assume, as in the case of the Fourier transform, that the functions are sines and cosines of the form $\sin(k_i x)$ and $\cos(k_j x)$. Assume that no two sine functions have the same integer wave number k_i and no two cosine functions the same integer wave number k_j. In this case the integral $\int_0^{2\pi} \sin(k_i x) \sin(k_j x)$ is equal to π, whenever $i = j$, otherwise it vanishes. The same is true for the cosine functions. The expression transforms then to

$$\pi \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} \int f_1 f \\ \int f_2 f \\ \vdots \\ \int f_n f \end{pmatrix}$$

which is just another way of computing the coefficients for the Fourier approximation of the function f.

2.5.2 The Hadamard–Walsh transform

In our case we are interested in expressing Boolean functions in terms of a set of primitive functions. We adopt bipolar coding, so that now the logical value false is represented by -1 and the logical value true by 1. In the case of n logical variables $x_1, \ldots, x_n$ and the logical functions defined on them, we can use the following set of 2^n primitive functions:

- The constant function $(x_1, \ldots, x_n) \mapsto 1$
- The $\binom{n}{k}$ monomials $(x_1, \ldots, x_n) \mapsto x_{l_1} x_{l_2} \cdots x_{l_k}$, where $k = 1, \ldots, n$ and $l_1, l_2, \ldots, l_k$ is a set of k different indices in $\{1, 2, \ldots, n\}$

All these functions are mutually orthogonal. In the case of two input variables the transformation becomes:

$$4 \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ -1 & 1 & -1 & 1 \\ -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 \end{pmatrix} \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{pmatrix}$$

In the general case we compute 2^n coefficients using the formula

$$\begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_{2^n} \end{pmatrix} = H_n \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_{2^n} \end{pmatrix}$$

where the matrix H_n is defined recursively as

$$H_n = \frac{1}{2} \begin{pmatrix} H_{n-1} & H_{n-1} \\ -H_{n-1} & H_{n-1} \end{pmatrix}$$

whereby

$$H_1 = \frac{1}{2} \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}.$$

The AND function can be expressed using this simple prescription as

$$x_1 \wedge x_2 = \frac{1}{4}(-2 + 2x_1 + 2x_2 + 2x_1 x_2).$$

The coefficients are the result of the following computation:

$$\begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{pmatrix} = \frac{1}{4} \begin{pmatrix} 1 & 1 & 1 & 1 \\ -1 & 1 & -1 & 1 \\ -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 \end{pmatrix} \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{pmatrix}.$$

The expressions obtained for the logical functions can be wired as networks using weighted edges and only two operations, addition and binary multiplication. The Hadamard–Walsh transform is consequently a method for the synthesis of Boolean functions. The next step, that is, the optimization of the number of components, demands additional techniques which have been extensively studied in the field of combinatorics.

2.5.3 Applications of threshold logic

Threshold units can be used in any application in which we want to reduce the execution time of a logic operation to possibly just two layers of computational delay without employing a huge number of computing elements. It has been shown that the *parity* and *majority* functions, for example, cannot be implemented in a fixed number of layers of computation without using an exponentially growing number of conventional logic gates [148, 464], even when unbounded fan-in is used. The majority function k out of n is a threshold function implementable with just a single McCulloch–Pitts unit. Although circuits built from n threshold units can be built using a polynomial number $P(n)$ of conventional gates the main difference is that conventional circuits cannot guarantee a *constant* delay. With threshold elements we can build multiplication or division circuits that guarantee a constant delay for 32 or 64-bit operands. Any symmetric Boolean function of n bits can in fact be built from two layers of computing units using $n+1$ gates [407]. Some authors have developed circuits of threshold networks for fast multiplication and division, which are capable of operating with constant delay for a variable number of data bits [405]. Threshold logic offers thus the possibility of harnessing parallelism at the level of the basic arithmetic operations.

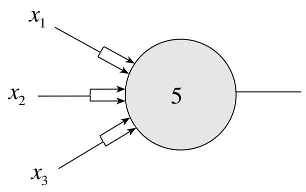


Fig. 2.26. Fault-tolerant gate

Threshold logic also offers a simpler way to achieve fault-tolerance. Figure 2.26 shows an example of a unit that can be used to compute the conjunction of three inputs with inherent fault tolerance. Assume that three inputs x_1, x_2, x_3 can be transmitted, each with probability p of error. The probability of a false result when x_1, x_2 and x_3 are equal, and we are computing the conjunction of the three inputs, is $3p$, since we assume that all three values are transmitted independently of each other. But assume that we transmit each value using two independent lines. The gate of Figure 2.26 has a threshold of 5, that is, it will produce the correct result even in the case where an input value is transmitted with an error. The probability that exactly two ones arrive as zeros is p^2 and, since there are 15 combinations of two out of six lines, the probability of getting the wrong answer is $15p^2$ in this case. If p is small enough then $15p^2 < 3p$ and the performance of the gate is improved for this combination of input values. Other combinations can be analyzed in a similar way. If threshold units are more reliable than the communication channels, redundancy can be exploited to increase the reliability of any computing system.

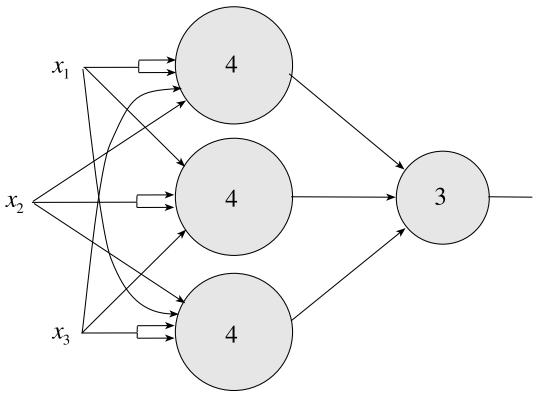


Fig. 2.27. A fault-tolerant AND built of noisy components

When the computing units are unreliable, fault tolerance is achieved using redundant networks. Figure 2.27 is an example of a network built using four units. Assume that the first three units connected directly to the three bits of input x_1, x_2, x_3 all fire with probability 1 when the total excitation is greater than or equal to the threshold θ but also with probability p when it is $\theta - 1$. The duplicated connections add redundancy to the transmitted bit, but in such a way that all three units fire with probability one when the three bits are 1. Each unit also fires with probability p if two out of three inputs are 1. However each unit reacts to a different combination. The last unit, finally, is also noisy and fires any time the three units in the first level fire and also with probability p when two of them fire. Since, in the first level, at most one unit fires when just two inputs are set to 1, the third unit will only fire when all three inputs are 1. This makes the logical circuit, the AND function of three inputs, built out of unreliable components error-proof. The network can be simplified using the approach illustrated in Figure 2.26.

2.6 Historical and bibliographical remarks

Warren McCulloch started pondering networks of artificial neurons as early as 1927 but had problems formulating a general, biologically plausible model since at that time inhibition in neurons had not yet been confirmed. He also had problems with the mathematical treatment of recurrent networks. Inhibition and recurrent pathways in the brain were confirmed in the 1930s and this cleared the way for McCulloch's investigations.

The McCulloch–Pitts model was proposed at a time when Norbert Wiener and Arturo Rosenblueth had started discussing the important role of feedback in biological systems and servomechanisms [301]. Wiener and his circle had published some of these ideas in 1943 [297]. Wiener's book *Cybernetics*, which was the first best-seller in the field of Artificial Intelligence, is the most influential work of this period. The word *cybernetics* was coined by Wiener and was intended to underline the importance of adaptive control in living and artificial systems. Wiener himself was a polymath, capable of doing first class research in mathematics as well as in other fields, such as physiology and medicine.

McCulloch and Pitts' model was superseded rapidly by more powerful approaches. Although threshold elements are, from the combinatorial point of view, more versatile than conventional logic gates, there is a problem with the assumed unlimited fan-in. Current technology has been optimized to handle a limited number of incoming signals into a gate. A possible way of circumventing the electrical difficulties could be the use of optical computing elements capable of providing unlimited fan-in [278]. Another alternative is the definition of a maximal fan-in for threshold elements that could be produced using conventional technology. Some experiments have been conducted in this direction and computers have been designed using exclusively threshold logic elements. The *DONUT* computer of Lewis and Coates was built in the 1960s using 614 gates with a maximal fan-in of 15. The same processor built with NOR gates with a maximal fan-in of 4 required 2127 gates, a factor of approximately 3.5 more components than in the former case [271].

John von Neumann [326] extensively discussed the model of McCulloch and Pitts and looked carefully at its fault tolerance properties. He examined the question of how to build a reliable computing mechanism built of unreliable components. However, he dealt mainly with the redundant coding of the units' outputs and a canonical building block, whereas McCulloch and his collaborator Manuel Blum later showed how to build reliable networks out of general noisy threshold elements [300]. Winograd and Cowan generalized this approach by replicating modules according to the requirements of an error-correcting code [458]. They showed that sparse coding of the input signals, coupled with error correction, makes possible fault-tolerant networks even in the presence of transmission or computational noise [94].

Exercises

1. Design a McCulloch–Pitts unit capable of recognizing the letter "T" digitized in a 10×10 array of pixels. Dark pixels should be coded as ones, white pixels as zeroes.
2. Build a recurrent network capable of adding two sequential streams of bits of arbitrary finite length.
3. Show that no finite automaton can compute the product of two sequential streams of bits of arbitrary finite length.
4. The parity of n given bits is 1 if an odd number of them is equal to 1, otherwise it is 0. Build a network of McCulloch–Pitts units capable of computing the parity function of two, three, and four given bits.
5. How many possible states can assume the binary scaler in Figure 2.21? Write the state and output tables for an equivalent finite automaton.
6. Design a network like the one shown in Figure 2.24 capable of simulating the finite automaton of the previous exercise.
7. Find polynomial expressions corresponding to the OR and XOR Boolean functions using the Hadamard–Walsh transform.
8. Show that the Hadamard–Walsh transform can be computed recursively, so that the number of multiplications becomes $O(n \log n)$, where n is the dimension of the vectors transformed (with n a power of two).
9. What is the probability of error in the case of the fault-tolerant gate shown in Figure 2.26? Consider one, two, and three faulty input bits.
10. The network in Figure 2.27 consists of unreliable computing units. Simplify the network. What happens if the units *and* the transmission channels are unreliable?