

Fundamentals of Neurocomputing

Leandro Nunes de Castro

<http://www.dca.fee.unicamp.br/~lnunes>

Technical Report

DCA-RT 01/03

February, 2003

Copyright © by Leandro Nunes de Castro



State University of Campinas- UNICAMP

School of Electrical and Computer Engineering - FEEC

Department of Computer Engineering and Industrial Automation – DCA

Contents

1. INTRODUCTION.....	1
2. THE NERVOUS SYSTEM.....	2
2.1. LEVELS OF ORGANIZATION IN THE NERVOUS SYSTEM	3
2.2. BIOLOGICAL AND PHYSICAL BASIS OF LEARNING AND MEMORY	8
3. ARTIFICIAL NEURAL NETWORKS.....	9
3.1. ARTIFICIAL NEURONS.....	10
3.2. NETWORK ARCHITECTURES.....	17
3.3. LEARNING APPROACHES.....	21
4. TYPICAL ARTIFICIAL NEURAL NETWORKS	27
4.1. HEBBIAN NETWORK	27
4.2. SINGLE-LAYER PERCEPTRON.....	28
4.3. ADALINE, THE LMS ALGORITHM, AND ERROR SURFACES	34
4.4. MULTI-LAYER PERCEPTRON.....	36
4.5. SELF-ORGANIZING MAPS.....	49
4.6. DISCRETE HOPFIELD NETWORK.....	55
5. FROM NATURAL TO ARTIFICIAL NEURAL NETWORKS.....	61
6. SCOPE OF NEUROCOMPUTING.....	62
7. SUMMARY	62
8. A QUICK GUIDE TO THE LITERATURE	63
9. EXERCISES	65
9.1. QUESTIONS	65
9.2. COMPUTATIONAL EXERCISES	65
9.3. THOUGHT EXERCISES	68
9.4. PROJECTS AND CHALLENGES	68
REFERENCES	69

1. Introduction

How does the brain process information? How is it organized? What are the biological mechanisms involved in brain functioning? These form just a sample of some of the most challenging questions in science. Brains are especially good at performing functions like pattern recognition, motor control, perception, flexible inference, intuition, and guessing. But brains are also slow, imprecise, make erroneous generalizations, are prejudiced, and are usually incapable of explaining their own actions.

Neurocomputing, sometimes called *brain-like computation* or *neurocomputation* but most often referred to as *artificial neural networks* (ANN)¹, can be defined as information processing systems (computing devices) designed with inspiration taken from the nervous system, more specifically the brain, and with particular emphasis in problem solving. S. Haykin (1999) provides the following definition:

“A[n artificial] neural network is a massively parallel distributed processor made up of simple processing units, which has a natural propensity for storing experiential knowledge and making it available for use.” (Haykin, 1999; p. 2)

Many other definitions are available, such as

“A[n artificial] neural network is a circuit composed of a very large number of simple processing elements that are neurally based.” (Nigrin, 1993; p. 11)

Neurocomputing systems are distinct from what is now known as *computational neuroscience*, which is mainly concerned with the development of biologically-based computational models of the nervous system. Artificial neural networks on the other hand, take a loose inspiration from the nervous system and emphasize the problem solving capability of the systems developed. However, most books on computational neuroscience not only acknowledge the existence of artificial neural networks, but also use several ideas from them in the proposal of more biologically plausible models. They also discuss the ANN suitability as models of real biological nervous systems.

Neurons are believed to be the basic units used for computation in the brain, and their simplified abstract models are the basic processing units of neurocomputing devices or artificial neural networks. Neurons are connected to other neurons by a small junction called synapse, whose capability of being modulated is believed to be the basis for most of our cognitive abilities, such as perception, thinking and inferring. Therefore, some essential information about neurons, synapses, and their structural anatomy are relevant for the understanding of how ANNs are designed taking inspiration from biological neural networks.

The discussion to be presented here briefly introduces the main aspects of the nervous system used to devise neurocomputing systems, and then focuses on some of the most commonly used artificial neural networks, namely, single- and multi-layer perceptrons, self-organizing networks, and Hopfield networks. The description of the many algorithms uses a matrix notation particularly suitable for their software implementation of the algorithms. The biological plausibility of each model is also assessed.

¹ Although neurocomputing can be viewed as a field of research dedicated to the *design* of “intelligent” brain-like computers, this report uses the word neurocomputing as a synonym to artificial neural networks.

2. The Nervous System

All multicellular organisms have some kind of nervous system, whose complexity and organization varies according to the animal type. Even relatively simple organisms, such as worms, slugs, and insects, have the ability to learn and store information in their nervous systems. The nervous system is responsible for informing the organism through sensory input with regards to the environment in which it lives and moves, processing the input information, relating it to previous experience, and transforming it into appropriate actions or memories.

The nervous system plays the important role of processing the incoming information (signals) and providing appropriate actions according to these signals. The elementary processing units of the nervous system are the *neurons*, also called *nerve cells*. *Neural networks* are formed by the interconnection of many neurons. Each neuron in the human brain has on the order of hundreds or thousands of connections to other neurons.

Anatomically, the nervous system has two main divisions: *central nervous system* (CNS) and *peripheral nervous system* (PNS), the distinction being their different locations. Vertebrate animals have a *bony spine* (*vertebral column*) and a *skull* (*cranium*) in which the central parts of the nervous system are housed. The peripheral part extends throughout the remainder of the body. The part of the (central) nervous system located in the skull is referred to as the *brain*, and the one found in the spine is called the *spinal cord*. The brain and the spinal cord are continuous through an opening in the base of the skull; both are in contact with other parts of the body through the nerves.

The brain can be further subdivided into three main structures: the *brainstem*, the *cerebellum*, and the *forebrain*, as illustrated in Figure 1. The *brainstem* is literally the stalk of the brain through which pass all the nerve fibers relaying input and output signals between the spinal cord and higher brain centers. It also contains the cell bodies of neurons whose axons go out to the periphery to innervate the muscles and glands of the head. The structures within the brainstem are the *midbrain*, *pons*, and the *medulla*. These areas contribute to functions such as breathing, heart rate and blood pressure, vision, and hearing. The cerebellum is located behind the brainstem and is chiefly involved with skeletal muscle functions and helps to maintain posture and balance and provides smooth, directed movements. The *forebrain* is the large part of the brain remaining when the brainstem and cerebellum have been excluded. It consists of a central core, the *diencephalon*, and right and left *cerebral hemispheres* (the *cerebrum*).

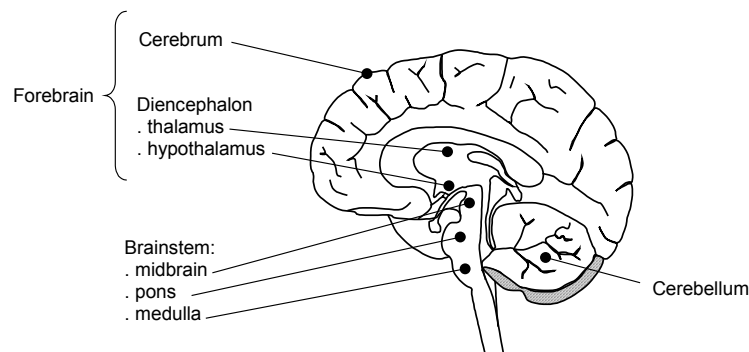


Figure 1: Structural divisions of the brain as seen in a midsagittal section.

The outer portion of the cerebral hemispheres is called *cerebral cortex*. The cortex is involved in several important functions such as thinking, voluntary movements, language, reasoning, and perception. The *thalamus* part of the diencephalon is important for integrating all sensory input (except smell) before it is presented to the cortex. The *hypothalamus*, which lies below the thalamus, is a tiny region responsible for the integration of many basic behavioral patterns, which involve correlation of neural and endocrine functions. Indeed, the hypothalamus appears to be the most important area to regulate the internal environment (homeostasis). It is also one of the brain areas associated with emotions.

2.1. Levels of Organization in the Nervous System

What structures really constitute a level of organization in the nervous system is an empirical not an a priori matter. We cannot tell, in advance of studying the nervous system, how many levels there are, nor what is the nature of the structural and functional features of any given level (Churchland & Sejnowski, 1992). Therefore, only the structures particularly interesting for the understanding, description, and implementation of artificial neural networks will be described here.

The nervous system can be organized in different levels: molecules, synapses, neurons, networks, layers, maps, and systems (Figure 2). An easily recognizable structure in the nervous system is the neuron, which is a cell specialized in signal processing. Depending on environmental conditions, the neurons are capable of generating a signal, more specifically an *electric potential*, that is used to transmit information to other cells to which it is connected. Some processes in the neuron utilize cascades of biochemical reactions that influence information processing in the nervous system. Many neuronal structures can be identified with specific functions. For instance the *synapses* are important for the understanding of signal processing in the nervous system (Trappenberg, 2002).

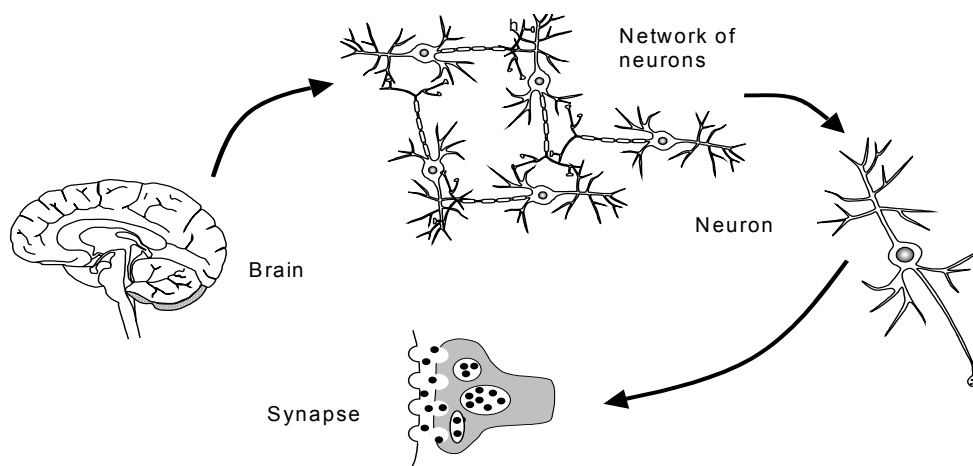


Figure 2: Some levels of organization in the nervous system.

2.1.1. Neurons and Synapses

Neurons use a variety of specialized biochemical mechanisms for information processing and transmission. These include *ion channels* that allow a controlled influx and outflux of currents, the generation and propagation of *action potentials*, and the release of *neurotransmitters*. Signal transmission between neurons is the core of the information processing capabilities of the brain. One of the most exciting discoveries in neuroscience was that the effectiveness of the signal transmission can be modulated in various ways, thus allowing the brain to adapt to different situations. It is believed to be the basis of associations, memories, and many other mental abilities (Trappenberg, 2002). *Synaptic plasticity*, that is the capability of synapses to be modified, is a key ingredient in most models described in this report.

Figure 3 shows a picture of a schematic generic neuron labeling its most important structural parts. The biological neuron is a single *cell*, thus containing a *cell body* with a *nucleus* (or *soma*) containing DNA, it is filled with fluid and cellular organelles, and is surrounded by a *cell membrane*, just like any other cell in the body. Neurons also have specialized extensions called *neurites*, that can be further distinguished into *dendrites* and *axons*. While the dendrites receive signals from other neurons, the axon propagates the output signal to other neurons.

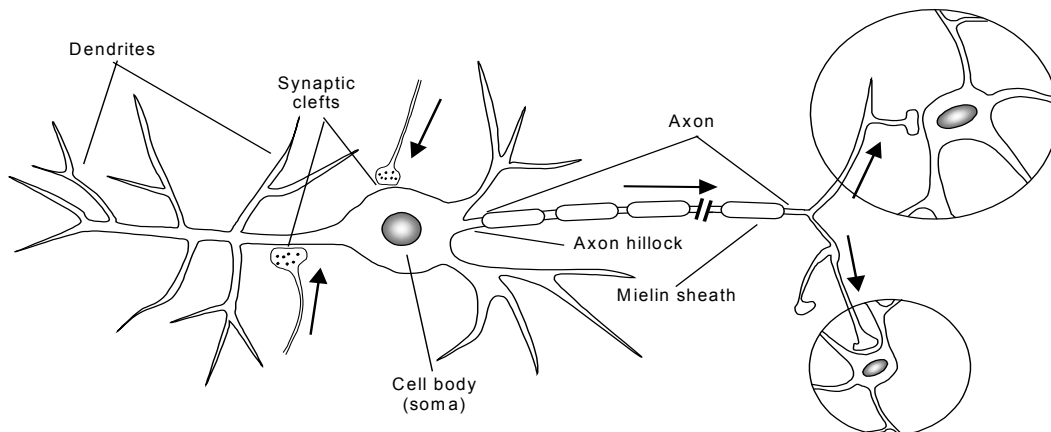


Figure 3: Schematic neuron similar in appearance to the pyramidal cells in the brain cortex. The parts outlined are the major structural components of most neurons. The direction of signal propagation between and within neurons is shown by the dark arrows.

One peculiarity about neurons is that they are specialized in signal processing utilizing special electrophysical and chemical processes. They can receive and send signals to many other neurons. The neurons that send signals, usually termed *sending* or *pre-synaptic neurons*, contact the *receiving* or *post-synaptic neurons* in specialized sites named *synapses* either at the cell body or at the dendrites. The synapse is thus the junction between the pre-synaptic neuron's axon and the post-synaptic neuron's dendrite or cell body.

The general information processing feature of synapses allow them to alter the state of a post-synaptic neuron, thus eventually triggering the generation of an electric pulse, called *action potential*, in the post-synaptic neuron. The action potential is usually initiated at the *axon hillock* and travels all along the axon, which can finally branch and send

information to different regions of the nervous system. Therefore, a neuron can be viewed as a device capable of receiving diverse input stimuli from many other neurons and propagating its single output response to many other neurons.

Various mechanisms exist to transfer information (signals) among neurons. As neurons are cells encapsulated in membranes, little openings in these membranes, called *channels*, allow the transfer of information among neurons. The basic mechanisms of information processing are based on the movement of charged atoms, *ions*, in and out of the channels and within the neuron itself. Neurons live in a liquid environment in the brain containing a certain concentration of ions, and the flow of ions in and out of a neuron through channels is controlled by several factors. A neuron is capable of altering the intrinsic electrical potential, the *membrane potential*, of other neurons, which is given by the difference between the electrical potential within and in the surroundings of the cell.

When an action potential reaches the terminal end of an axon, it mobilizes some ions by opening voltage-sensitive channels that allow the flow of ions into the terminal and possibly the release of some of these stored ions. These ions then promote the release of *neurotransmitters* (chemical substances) into the *synaptic cleft*, which finally diffuse across the cleft and binds with receptors in the post-synaptic neurons. As outcomes, several chemical processes might be initiated in the post-synaptic neuron, or ions may be allowed to flow into it. Figure 4 summarizes some of the mechanisms involved in synapse transmission.

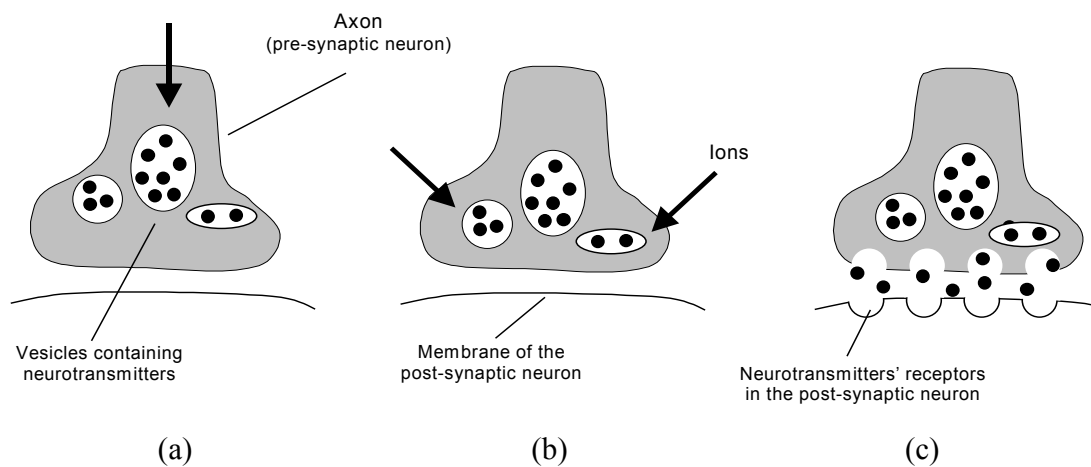


Figure 4: Diagram of a chemical synapse. The action potential arrives at the synapse (a) and causes ions to be mobilized in the axon terminal (b), thus causing vesicles to release neurotransmitters into the cleft, which in turn bind with the receptors on the post-synaptic neurons (c).

As the electrical effects of these ions (action potential) propagate through the dendrites of the receiving neuron and up to the cell body, the process of information can begin again in the post-synaptic neuron. When ions propagate up to the cell body, these signals are *integrated* (summed), and the resulting membrane potential will determine if the neuron is going to *fire*, i.e., to send an output signal to a post-synaptic neuron. This only occurs if the membrane potential of the neuron is greater than the neuron *threshold*. This is because the channels are particularly sensitive to the membrane potential; they will only open when the potential is sufficiently large (O'Reilly & Munakata, 2000). The action of neural

firing is also called *spiking*, *firing a spike*, or the *triggering of an action potential*. The spiking is a very important electrical response of a neuron; once generated it does not change its shape with increasing current. This phenomenon is called the *all-or-none* aspect of the action potential.

Different types of neurotransmitters and their associated ion channels have distinct effects on the state of the post-synaptic neuron. One class of neurotransmitters opens channels that will allow positively charged ions to enter the cell, thus triggering the increase of the membrane potential that drives the post-synaptic neurons towards their excited state. Other neurotransmitters initiate processes that drive the postsynaptic potential towards a resting state; a potential known as the *resting potential*. Therefore, neurotransmitters can promote the initiation of *excitatory* or *inhibitory* processes.

2.1.2. Networks, Layers, and Maps

Neurons can have *forward* and *feedback* connections to other neurons, meaning that they can have either one way or reciprocal connections with other neurons in the nervous system. These interconnected neurons give rise to what is known as *networks of neurons* or *neural networks*. For instance, within a cubic millimeter of cortical tissue, there are approximately 10^5 neurons and about 10^9 synapses, with the vast majority of these synapses arising from cells located within the cortex (Churchland & Sejnowski, 1992). Therefore, the degree of interconnectivity in the nervous system is quite high.

A small number of interconnected neurons (units) can exhibit complex behaviors and information processing capabilities that cannot be observed in single neurons. One important feature of neural networks is the representation of information (knowledge) in a *distributed* way, and the *parallel processing* of this information. No single neuron is responsible for storing “a whole piece” of knowledge; it has to be distributed over several neurons (connections) in the network. Networks with specific architectures and specialized information processing capabilities are incorporated into larger structures capable of performing even more complex information-processing tasks.

Many brain areas display not only networks of neurons but also *laminar organization*. Laminae are *layers of neurons* in register with other layers, and a given lamina conforms to a highly regular pattern of where it projects to and from where it receives projections. For instance, the *superior colliculus* (a particular layered midbrain structure; see Figure 1) receives visual inputs in superficial layers, and in deeper layers it receives tactile and auditory input. Neurons in an intermediate layer of the superior colliculus represent information about eye movements (Churchland & Sejnowski, 1992).

One of the most common arrangements of neurons in the vertebrate nervous systems is a layered two-dimensional structure organized with a *topographic* arrangement of unit responses. Perhaps the most well-known example of this is the mammalian cerebral cortex, though others such as the superior colliculus also use maps. The cerebral cortex is the outside surface of the brain. It is a two-dimensional structure extensively folded with fissures and hills in many larger and/or more intelligent animals. Two different kinds of cortex exist: an older form with three sub-layers called *paleocortex*, and a newer form that is most prominent in animals with more complex behavior, a structure with six or more sub-layers called *neocortex*.

Studies in human beings by neurosurgeons, neurologists, and neuropathologists have shown that different cortical areas have separate functions. Thus, it is possible to identify visible differences between different regions of the cerebral cortex, each presumably corresponding to a processing module. Figure 5 illustrates some specific functional areas in the cerebral cortex of humans. Note that regions associated with different parts of the body can have quite sharp boundaries.

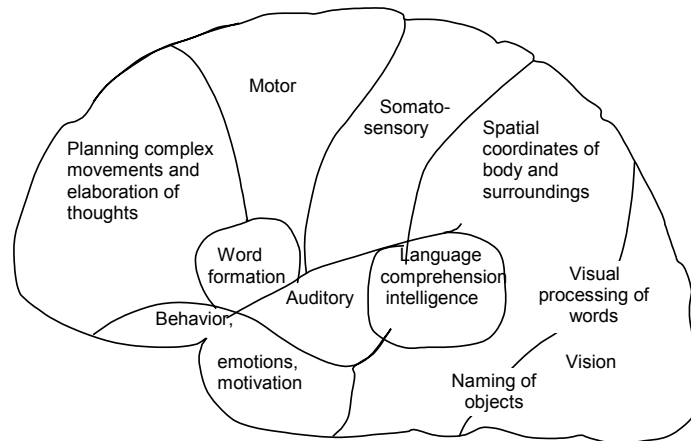


Figure 5: Map of specific functional areas in the cerebral cortex.

In general, it is known that neocortical neurons are organized into six distinct layers, which can be subdivided into *input*, *hidden*, and *output* layer. The input layer usually receives the sensory input, the output layer sends commands and outputs to other portions of the brain, and the hidden layers receive inputs locally from other cortical layers. This means that hidden layers neither directly receive sensory stimuli nor produce motor and other outputs.

One major principle of organization within many sensory and motor systems is the *topographic map*. For instance, neurons in visual areas of the cortex (in the rear end of the cortex, opposite to the eyes; see Figure 5) are arranged topographically, in the sense that adjacent neurons have adjacent visual receptive fields and collectively they constitute a map of the retina. Because neighboring processing units are concerned with similar representations, topographic mapping is an important means whereby the brain manages to save on wiring and also to share wire.

Networking, topographic mapping, and layering are all special cases of a more general principle: the exploitation of geometric and structural properties in information processing design. Evolution has shaped our brains so that these structural organizations became efficient ways for biological systems to assemble in one place information needed to solve complex problems.

2.2. Biological and Physical Basis of Learning and Memory

The nervous system is continuously modifying and updating itself. Virtually all its functions, including perception, motor control, thermoregulation, and reasoning, are modifiable by experience. The topography of the modifications appears not to be final and finished, but an ongoing process with a virtually interminable schedule. Behavioral observations indicate degrees of plasticity in the nervous system: there are fast and easy changes, slower and deeper modifiability, and a more permanent but still modifiable changes.

In general, global learning is a function of local changes in neurons. There are many possible ways a neuron could change to embody adaptations. For instance, new dendrites might sprout out, or there might be extension of existing branches, or existing synapses could change, or new synapses might be created. In the other direction, pruning could decrease the dendrites or bits of dendrites, and thus decrease the number of synapses, or the synapses on the remaining branches could be shut down altogether. These are all post-synaptic changes in the dendrites. There could also be changes in the axons; for instance, there might be changes in the membrane, or new branches might be formed, and genes might be induced to produce new neurotransmitters or more of the old ones. Pre-synaptic changes could include changes in the number of vesicles released per spike and the number of transmitter molecules contained in each vesicle. Finally, the whole cell might die, taking with it all the synapses it formerly supported (Churchland & Sejnowsky, 1992).

This broad range of structural adaptability can be conveniently condensed in the present discussion by referring simply to synapses, since every modification either involves synaptic modification directly or indirectly, or can be reasonably so represented. Learning by means of setting synaptic efficiency is thus the most important mechanism in neural networks, biological and artificial. It depends on both individual neuron-level mechanisms and network-level principles to produce an overall network that behaves appropriately in a given environment.

Two of the primary mechanisms underlying learning in the nervous system are the *long-term potentiation* (LTP) and *long-term depression* (LTD), which refers to the strengthening and weakening of weights in a non-transient form. Potentiation corresponds to an increase in the measured depolarization or excitation delivered by a controlled stimulus onto a receiving neuron, and depression corresponds to a decrease in the measured depolarization. In both cases, the excitation or inhibition of a membrane potential may trigger a complex sequence of events that ultimately result in the modification of synaptic efficiency (strength).

Such as learning, *memory* is an outcome of an adaptive process in synaptic connections. It is caused by changes in the synaptic efficiency of neurons as a result of neural activity. These changes in turn cause new pathways or facilitated pathways to develop for transmission of signals through the neural circuits of the brain. The new or facilitated pathways are called *memory traces*; once established, they can be activated by the thinking mind to reproduce the memories. Actually, one of the outcomes of a learning process can be the creation of a more permanent synaptic modification scheme, thus resulting in the memorization of an experience.

Memories can be classified in a number of ways. One common classification being into:

- *short-term memory*: lasts from a few seconds to a few minutes – e.g., one’s memory of a phone number;
- *intermediate long-term memory*: lasts from minutes to weeks – e.g., the name of a beautiful girl you met in a party;
- *long-term memory*: lasts for an indefinite period of time – e.g., your home address.

While the two first types of memories do not require many changes in the synapses, long-term memory is believed to require *structural changes* in synapses. These structural changes include an increase in number of vesicle release sites for secretion of neurotransmitter substances, an increase in the number of pre-synaptic terminals, an increase in the number of transmitter vesicles, and changes in the structures of the dendritic spines.

Therefore, the difference between learning and memory may be sharp and conceptual. Learning can be viewed as the adaptive process that results in the change of synaptic efficiency and structure, while memory is the (long-lasting) result of this adaptive process.

3. Artificial Neural Networks

Artificial neural networks (ANN) present a number of features and performance characteristics in common with the nervous system:

- The basic information processing occurs in many simple elements called (artificial) *neurons (nodes or units)*;
- These neurons are interconnected thus forming networks of neurons or *neural networks*;
- Information (signals) are transmitted between neurons via connection links called *synapses*;
- The efficiency of a synapse, represented by an associated *weight value* or *strength*, corresponds to the information stored in the neuron, thus in the network; and
- Knowledge is acquired from the environment by a process known as *learning* that is basically responsible for *adapting* the connection strengths (weight values) to the environmental stimuli.

One important feature of artificial neural networks is where the knowledge is stored. Basically, what is stored is the connection strengths (synaptic strengths) between units (artificial neurons) that allow patterns to be recreated. This feature has enormous implications, both for processing and learning. The knowledge representation is set up so that the knowledge necessarily influences the course of processing; it becomes a part of the processing itself. If the knowledge is incorporated into the strengths of the connections, then learning becomes a matter of finding the appropriate connection strengths so as to produce satisfactory patterns of activation under some circumstances.

This is an extremely important feature of ANNs, for it opens up the possibility that an information processing mechanism could *learn*, by tuning its connections, to capture the interdependencies between activations presented to it in the course of processing. Another important implication of this type of representation is that *the knowledge is distributed* over the connections among a large number of units. There is no “special” unit reserved for particular patterns.

An artificial neural network can be characterized by three main features: 1) a set of *artificial neurons*, *nodes* or *units*, 2) the pattern of connectivity among neurons, called the network *architecture* or *structure*, and 3) a method to determine the weight values, called its *training* or *learning* algorithm. Each one of these features will be discussed separately in the following sections.

3.1. Artificial Neurons

In the biological neuron, inputs come into the cell primarily through channels located in synapses, allowing ions to flow into and out of the neuron. A membrane potential appears as a result of the integration of the neural inputs, and will then determine whether a given neuron will produce a spike (action potential) or not. This spike causes neurotransmitter to be released at the end of the axon, which then forms synapses with other neuron's dendrites. The action potential only occurs when the membrane potential is above a critical threshold level. Different inputs can provide different amounts of activation depending on how much neurotransmitter is released by the sender and how many channels in the post-synaptic neuron are opened. Therefore, there are important features of the biological synapses involved in the information processing of neurons.

The net effect of these biological processes is summarized in the computational models discussed here by a *weight* (also called *synaptic strength*, *synaptic efficiency*, *connection strength* or *weight value*) between two neurons. Furthermore, the modification of one or more of these weight factors will have a major contribution for the neural network learning process. This section reviews three models of neuronal function. The first is the McCulloch-Pitts model in which the neuron is assumed to be computing a logic function. The second is a simple analog *integrate-and-fire* model. And the third is a generic connectionist neuron, which integrates its inputs and generates an output using one particular *activation function*. These neuronal models may be interchangeably called *nodes*, *units*, *artificial neurons* or simply neurons. It is important to have in mind though, that the nodes most commonly used in artificial neural networks bear a far resemblance with real neurons.

3.1.1. Elementary Logic of Propositions

Several proofs in mathematics and many algorithms in computer science use *logical expressions* such as:

“If it starts to rain then open your umbrella”

Which can be generically expressed as

“If *a* then *b*”

It is usually necessary to know the cases in which a given logical expression is TRUE (T) or FALSE (F), referred to as the *truth-value* of such expressions.

A *proposition* or *statement* is a declarative sentence that can be either true or false, but not both. For instance, consider the following sentences:

- “Brazil is in South America.” It is a true proposition.
- “Mars is the Blue Planet.” It is a false proposition.
- “What are you doing tonight?” It is not a proposition.

Many propositions are *composite*, i.e., composed of sub-propositions and various *connectives*, thus forming *compound propositions*. The fundamental property of a compound proposition is that its truth-value is completely determined by the truth-value of its sub-propositions together with the way in which they are connected to form the compound compositions.

There are three basic logical operations or *connectives*: conjunction (“AND”), disjunction (“OR”), and negation (“NOT”). These are also called Boolean function.

Let a and b be two propositions, the definition for each of the three basic logical operations are:

- a AND b : If a and b are both true, then a AND b is true; otherwise a AND b is false.
- a OR b : If a and b are both false, then a OR b is false; otherwise a OR b is true.
- NOT a : If a is true, then NOT a is false; otherwise NOT a is true.

A simple and concise way to show the relationship between the truth-value of a compound proposition and the truth-value of its sub-propositions is through a *truth table*. Figure 6 depicts the truth tables for the connectives AND, OR, and NOT; the value ‘0’ corresponds to FALSE, and ‘1’ corresponds to TRUE.

a	b	a AND b	a	b	a OR b	a	NOT a
0	0	0	0	0	0	0	1
0	1	0	0	1	1	1	0
1	0	0	1	0	1		
1	1	1	1	1	1		

Figure 6: Truth tables for the connectives AND, OR, and NOT.

3.1.2. The McCulloch and Pitts Neuron

W. McCulloch and W. Pitts (1943) wrote a famous and influential paper based on the computations that could be performed by two-state neurons. They did one of the first attempts to understand nervous activity based upon elementary neural computing units, which were highly abstract models of the physiological properties of neurons and their connections. Five physical assumptions were made for their calculus:

1. The activity of the neuron is an “all-or-none” (binary) process;
2. A certain fixed number of synapses must be excited within a period of latent addition in order to excite the neuron at any time, and this number is independent of previous activity and position of the neuron;
3. The only significant delay within the nervous system is synaptic delay;
4. The activity of any inhibitory synapse absolutely prevents excitation of the neuron at that time; and
5. The structure of the neural network does not change with time.

McCulloch and Pitts conceived the neuronal response as being equivalent to a *proposition* adequate to the neuron’s stimulation. Therefore, they studied the behavior of complicated neural networks using a notation of the symbolic *logic of propositions*. The “all-or-none”

law of nervous activity was sufficient to ensure that the activity of any neuron could be represented as a proposition.

It is important to note that according to our current knowledge of how the neuron works, based upon electrical and chemical processes, neurons are not realizing any propositions of logic. But the model of McCulloch and Pitts can be considered as a special case of the most general neuronal model to be discussed in the next sections, and is still sometimes used to study some classes of non-linear networks. Additionally, this model has caused a major impact mainly among computer scientists and engineers, encouraging the development of artificial neural networks. It has even been influential in the history of computing (cf. von Neumann, 1982).

The McCulloch and Pitts neuron is binary, i.e., it can assume only one of two states (either '0' or '1'). Each neuron has a fixed *threshold* θ , and receives inputs from synapses of identical weight values. The neuronal mode of operation is simple. At each time step t , the neuron responds to its synaptic inputs, which reflect the state of the pre-synaptic neurons. If no inhibitory synapse is active, the neuron integrates (sums up) its synaptic inputs, generating the *net input* to the neuron, u , and checks if this sum (u) is greater than the threshold θ . If it is, then the neuron becomes active, that is, responds with a '1' in its output ($y = 1$); otherwise it remains inactive, that is, responds with a '0' in its output ($y = 0$).

Although this neuron is quite simple, it already presents some important features in common with most neuron models, namely, the integration of the input stimuli to determine the *net input* u , and the presence of an activation function (threshold). Figure 7 illustrates the simple McCulloch and Pitts neuron and its activation function.

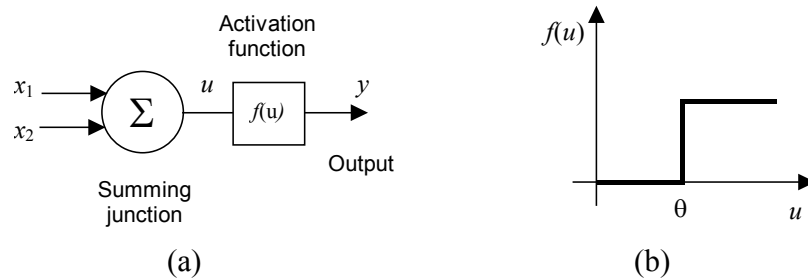


Figure 7: The McCulloch and Pitts neuron (a) and its threshold activation function (b).

To illustrate the behavior of this simple unit, assume two excitatory inputs x_1 and x_2 and a threshold $\theta = 1$. In this case, the neuron is going to fire, that is, to produce an output '1', every time x_1 or x_2 has a value '1', thus operating like the logical connective OR (see Figure 6). Assume now that the neuron threshold is increased to $\theta = 2$. In this new situation, the neuron is only going to be active (fire) if both x_1 and x_2 have value '1' simultaneously, thus operating like the logical connective AND (see Figure 6).

3.1.3. A Basic Integrate-and-Fire Neuron

Assume a noise-free neuron with the net input being a variable of time $u(t)$ corresponding to the membrane potential of the neuron. The main effects of some neuronal channels (in

particular the sodium and leakage channels) can be captured by a simple equation of an integrator (Dayan & Abbot, 2001; Trappenberg, 2002):

$$\tau_m \frac{du(t)}{dt} = u_{res} - u(t) + R_m i(t), \quad (1)$$

where τ_m is the membrane time constant of the neuron determined by the average conductance of the channels (among other things); u_{res} is the resting potential of the neuron; $i(t)$ is the input current given by the sum of the synaptic currents generated by firings of pre-synaptic neurons; and R_m is the resistance of the neuron to the flow of current (ions).

Equation (1) can be very simply understood. The rate of variation of the membrane potential of the neuron is proportional to its current membrane potential, its resting potential, and the potential generated by the incoming signals to the neuron. Note that the last term on the right side of this equation is the Ohm's law ($u = R \cdot i$) for the voltage generated by the incoming currents.

The input current $i(t)$ to the neuron is given by the sum of the incoming synaptic currents depending on the *efficiency* of individual synapses, described by the variable w_j for each synapse j . Therefore, the total input current to the neuron can be written as the sum of the individual synaptic currents multiplied by a weight value

$$i(t) = \sum_j \sum_{t_j^f} w_j f(t - t_j^f), \quad (2)$$

where the function $f(\cdot)$ parameterizes the form of the post-synaptic response. This function was termed *activation function* in the McCulloch and Pitts neuron discussed above and this nomenclature will be kept throughout this text. The variable t_j^f denotes the firing time of the pre-synaptic neuron of synapse j . The firing time of the post-synaptic neuron is defined by the time the membrane potential u reaches a threshold value θ ,

$$u(t^f) = \theta. \quad (3)$$

In contrast to the firing time of the pre-synaptic neurons, the firing time of the integrate-and-fire neuron has no index. To complete the model, the membrane potential has to be reset to the resting state after the neuron has fired. One form of doing this is by simply resetting the membrane potential to a fixed value u_{res} immediately after a spike.

3.1.4. The Generic Neurocomputing Neuron

The computing element employed in most neural networks is an integrator, such as the McCulloch and Pitts and the integrate-and-fire models of a neuron, and does detailed computation based on its connection strengths. Like in the brain, the artificial neuron is an information-processing element that is fundamental to the operation of the neural network. Figure 8 illustrates the typical artificial neuron, depicting its most important parts: the synapses, characterized by their *weight values* connecting each input to the neuron; the *summing junction* (*integrator*); and the *activation function*.

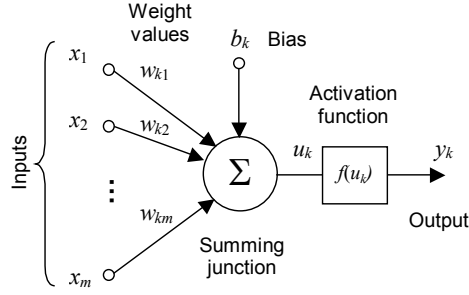


Figure 8: Non-linear model of a neuron.

Specifically, an input signal x_j at the input of synapse j connected to neuron k is multiplied by the synaptic weight w_{kj} . In this representation, the first subscript of the synaptic weight refers to the neuron, and the second subscript refers to the synapse connected to it. The summing junction adds all input signals weighted by the synaptic weight values plus the neuron's bias b_k ; this operation constitutes the dot (or inner) product (see Appendix B), that is, a linear combination of the inputs with the weight values, plus the bias b_k . Finally, an activation function is used to limit the amplitude of the output of the neuron. The activation function is also referred to as a *squashing function* (Rumelhart et al., 1986a) for it limits the permissible amplitude range of the output signal to some finite value.

The bias has the effect of increasing or decreasing the net input to the activation function depending on whether it is positive or negative, respectively. In the generic neuron it is usually used in place of a fixed threshold θ for the activation function. For instance, the McCulloch and Pitts neuron (Figure 7) will fire when its net input is greater than θ :

$$y = f(u) = \begin{cases} 1 & \text{if } u \geq \theta \\ 0 & \text{otherwise} \end{cases},$$

where $u = x_1 + x_2$ for the neuron of Figure 7. It is possible to replace the threshold θ by a bias weight b that will be multiplied by a constant input of value '1':

$$y = f(u) = \begin{cases} 1 & \text{if } u \geq 0 \\ 0 & \text{otherwise} \end{cases},$$

where $u = x_1 + x_2 + b$. Note however, that while the bias is going to be adjustable during learning, the threshold assumes a fixed value.

It is important to note here that the output of this generic neuron is simply a number, and the presence of discrete action potentials is ignored. As real neurons are limited in dynamic range from zero-output firing rate to a maximum of a few hundred action potentials per second, the use of an activation function could be biologically justified.

Mathematically, the neuron k can be described by a simple equation:

$$y_k = f(u_k) = f\left(\sum_{j=1}^m w_{kj}x_j + b_k\right), \quad (4)$$

where $x_j, j = 1, \dots, m$, are the input signals; $w_{kj}, j = 1, \dots, m$, are the synaptic weights of neuron k ; u_k is the net input to the activation function; b_k is the bias of neuron k ; $f(\cdot)$ is the activation function; and y_k is the output signal of the neuron.

It is possible to simplify the notation of Equation (4) so as to account for the presence of the bias by simply defining a constant input signal $x_0 = 1$ connected to the neuron k with weight value $w_{k0} = b_k$. The resulting equation becomes,

$$y_k = f(u_k) = f\left(\sum_{j=0}^m w_{kj} x_j\right), \quad (5)$$

Figure 9 illustrates the reformulated model of a neuron.

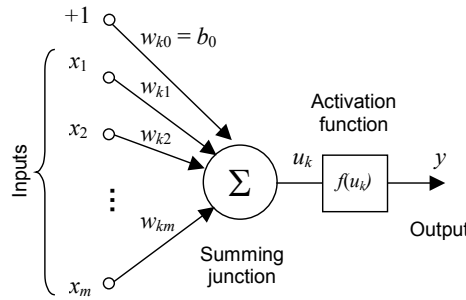


Figure 9: Reformulated model of a neuron.

Types of Activation Function

The activation function, denoted by $f(u_k)$, determines the output of a neuron k in relation to the net input u_k . It can thus assume a number of forms, of which some of the most frequently used are summarized below (see Figure 10):

- *Linear function* – relates the net input directly with its output.

$$f(u_k) = u_k. \quad (6)$$

- *Step or threshold function* – the output is ‘1’ if the net input is greater than or equal to a given threshold θ . This is the function originally used in the McCulloch and Pitts neuron:

$$f(u_k) = \begin{cases} 1 & \text{if } u_k \geq \theta \\ 0 & \text{otherwise} \end{cases}. \quad (7)$$

- *Sigmoid function* – it is a strictly increasing function that presents saturation and a graceful balance between linear and non-linear behavior. This is the most commonly used activation function in the ANN literature. It has an s-shaped form and can be obtained by several functions, such as the *logistic function*, the *arctangent function*, and the *hyperbolic tangent function*; the difference being that, as u_k ranges from $-\infty$ to $+\infty$, the logistic function ranges from 0 to 1, the arctangent ranges from $-\pi/2$ to $+\pi/2$, and the hyperbolic tangent ranges from -1 to $+1$:

$$\text{Logistic: } f(u_k) = \frac{1}{1 + \exp(-u_k)} \quad (8)$$

- *Radial basis function* – it is a non-monotonic function that is symmetric around a base value. This is the main type of function for *radial basis function neural networks*. The expression shown below is that of the Gaussian bell curve:

$$f(u_k) = \exp(-u_k^2). \quad (9)$$

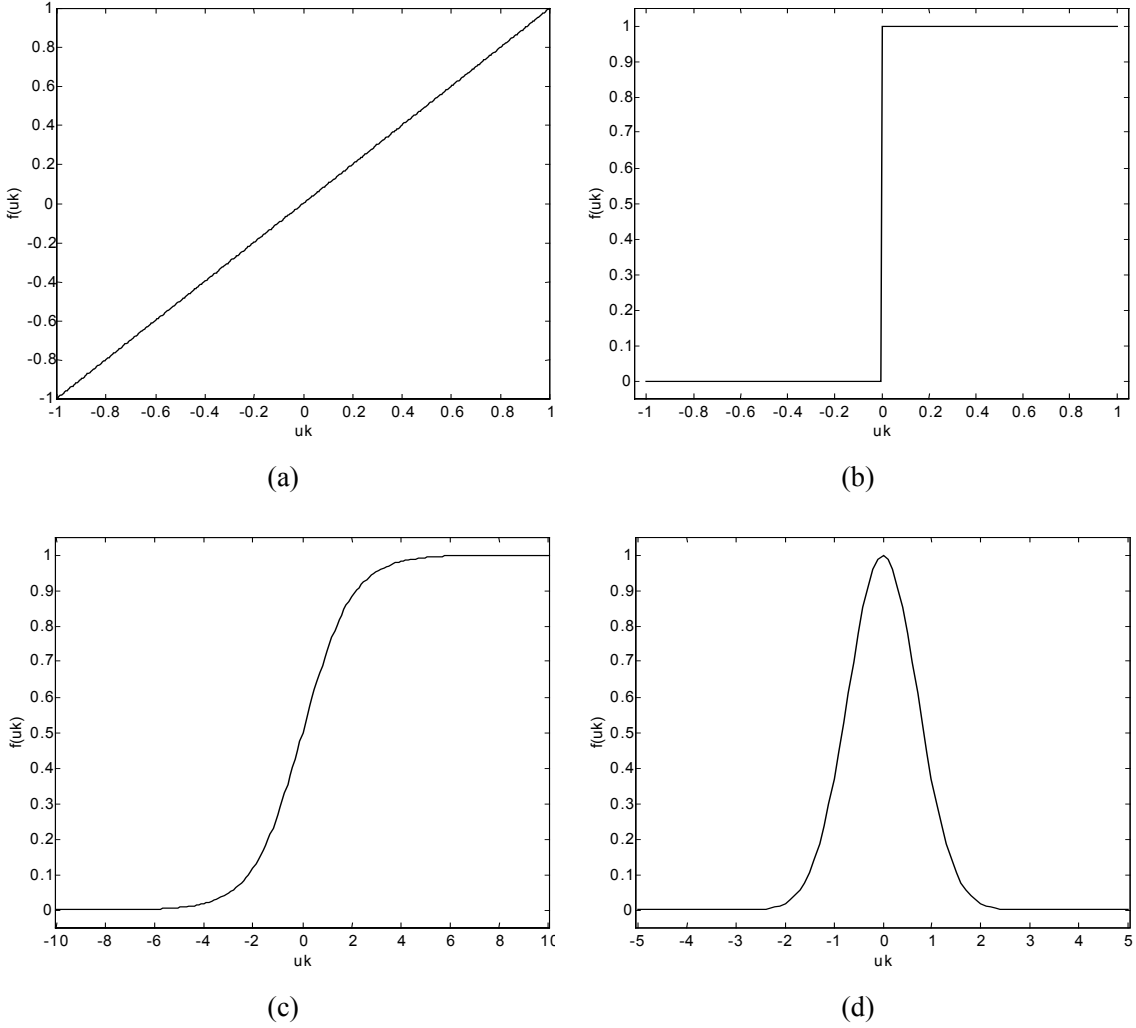


Figure 10: Most commonly used types of activation function. (a) Linear activation, $u_k \in [-1, +1]$. (b) Step function with $\theta = 0$, $u_k \in [-1, +1]$. (c) Logistic function, $u_k \in [-10, +10]$. (d) Gaussian bell curve, $u_k \in [-5, +5]$.

It is important to stress that the functions described and illustrated in Figure 10 depict only the general shape of those functions. However, it should be clear that these basic shapes can be modified. All these functions can include parameters with which some features of the functions can be changed, such as the *slope* and *offset* of a function. For instance, by multiplying u_k in the logistic function by a value β , it is possible to control the smoothness of the function:

$$f(u_k) = \frac{1}{1 + \exp(-\beta \cdot u_k)}$$

In this case, for larger values of β , this function becomes more similar to the threshold function, while for smaller values of β , the logistic function becomes more similar to a linear function.

3.2. Network Architectures

One of the basic pre-requisites for the emergence of complex behaviors is the interaction of a number of individual agents. In the nervous system, it is known that individual neurons can affect the behavior (firing rate) of others, but, as a single entity, a neuron is meaningless. This might be one reason why evolution drove our brains to a system with such an amazingly complex network of interconnected neurons.

In the human brain not much is known of how neurons are interconnected. Some particular knowledge is available for specific brain portions, but little can be said about the brain as a whole. For instance, it is known that the cortex can be divided into a number of different cortical areas specialized in different kinds of processing; some areas perform pattern recognition, others process spatial information, language processing, and so forth. In addition, it is possible to define anatomic neuronal organizations in the cortex in terms of *layers of neurons* (O'Reilly & Munakata, 2000), which are very important for the understanding of the detailed biology of the cortex.

In the domain of artificial neural networks, a layer of neurons will refer to functional layers of nodes. As not much is known about the biological layers of neurons, most neuro-computing networks employ some standardized architectures, specially designed for the engineering purposes of solving problems. There are basically three possible layers in a network: an *input layer*, one or more *intermediate* (or *hidden*) *layers*, and an *output layer*. These are so-called because the input layer receives the input stimuli directly from the environment, the output layer places the network output(s) to the environment, and the hidden layer(s) is not in direct contact with the environment.

The way in which neurons are structured (interconnected) in an artificial neural network is intimately related to the learning algorithm that is used for the network training. In addition, this interconnectivity affects the network storage and learning capabilities. In general, it is possible to distinguish three main types of network architectures (Haykin, 1999): *single-layer feedforward networks*, *multi-layer feedforward networks*, and *recurrent networks*.

3.2.1. Single-Layer Feedforward Networks

The simplest case of layered networks consists of an input layer of nodes whose output feeds the output layer. Usually, the input nodes are linear, i.e., they simply propagate the input signals to the output layer of the network. In this case, the input nodes are also called sensory units because they only sense (receive information from) the environment and propagate the received information to the next layer. In contrast, the output units are usually processing elements, such as the neuron depicted in Figure 9, with a non-linear type of activation function. The signal propagation in this network is purely positive or *feed-forward*, that is, signals are propagated from the network input to its outputs and never the

opposite way (*backward*). This architecture is illustrated in Figure 11(a) and the direction of signal propagation in Figure 11(b). In order not to overload the picture, very few connection strengths were depicted in Figure 11(a), but these might be sufficient to give a general idea of how the weights are assigned in the network.

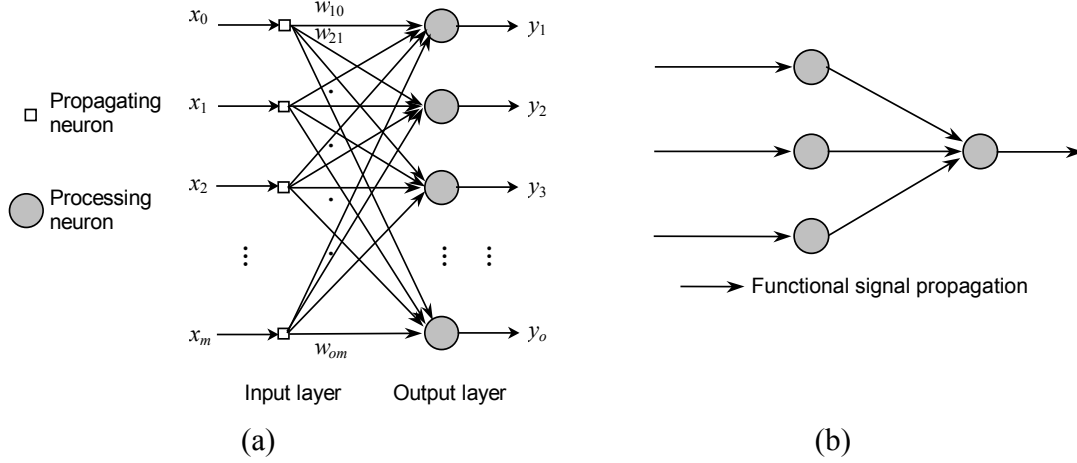


Figure 11: Single-layer feedforward neural network. (a) Network architecture. (b) Direction of propagation of the functional signal.

The neurons presented in Figure 11 are of the generic neuron type illustrated in Figure 9. (Note that input x_0 is assumed to be fixed in '1' and the weight vector that connects it to all output units w_{i0} , $i = 1, \dots, o$, corresponds to the bias of each output node b_i , $i = 1, \dots, o$.) The weight values between the inputs and the output nodes can be written using matrix notation (Appendix B) as

$$\mathbf{W} = \begin{bmatrix} w_{10} & w_{11} & \cdots & w_{1m} \\ \vdots & \vdots & \ddots & \vdots \\ w_{o0} & w_{o1} & \cdots & w_{om} \end{bmatrix}, \quad (10)$$

where the first index i , $i = 1, \dots, o$, of each element corresponds to the post-synaptic node and the second index j , $j = 1, \dots, m$, corresponds to the pre-synaptic node (just remember 'to-from' while reading from left to right). Therefore, each row of matrix \mathbf{W} corresponds to the weight vector of an output unit, and each element of a given column j , $j = 1, \dots, m$, corresponds to the strength with which an input j is connected to each output unit i .

The output of each post-synaptic neuron i in Figure 11(a), $i = 1, \dots, o$, is computed by applying an activation function $f(\cdot)$ to its net input, which is given by the linear combination of the network inputs and the weight vector,

$$y_i = f(\mathbf{w}_i \cdot \mathbf{x}) = f(\sum_j w_{ij} \cdot x_j), \quad j = 1, \dots, m. \quad (11)$$

The output vector of the whole network \mathbf{y} is given by the activation function applied to the product of the weight matrix \mathbf{W} by the input vector \mathbf{x} ,

$$\mathbf{y} = f(\mathbf{W} \cdot \mathbf{x}) \quad (12)$$

Consider the general case where the weight and input vectors can assume any real value, thus the dimension of the weight matrix and each vector is $\mathbf{W} \in \mathbb{R}^{o \times m}$, $\mathbf{w}_i \in \mathbb{R}^{1 \times m}$, $i = 1, \dots, o$, $\mathbf{x} \in \mathbb{R}^{m \times 1}$, and $\mathbf{y} \in \mathbb{R}^{o \times 1}$.

3.2.2. Multi-Layer Feedforward Networks

The second class of feedforward neural networks is known as multi-layer networks. These are distinguished from the single-layered networks by the presence of one or more *intermediate* or *hidden* layers. By adding one or more hidden layers, the non-linear computational processing and storage capability of the network is increased, for reasons that will become clearer further in the text. The output of each network layer is used as input to the following layer. A multi-layer feedforward network can be thought of as an assembly line: some basic material is introduced into the production line and passed on, the second stage components are assembled, and then the third stage, up to the last or output stage, which delivers the final product.

The learning algorithm typically used to train this type of network requires the *back-propagation* of an error signal calculated between the network output and a desired output. This architecture is illustrated in Figure 12(a) and the direction of signal propagation is depicted in Figure 12(b).

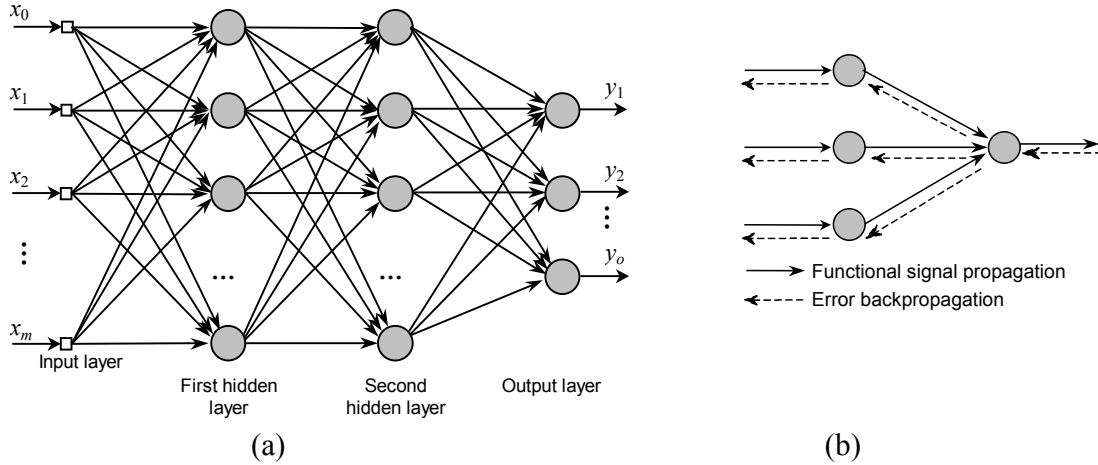


Figure 12: Multi-layer feedforward neural network. (a) Network architecture (legend as depicted in Figure 11). (b) Direction of propagation of the functional and error signals.

The weight values were suppressed from the picture not to overload it. In such networks, there is one weight matrix for each layer in the net, and these are going to be denoted by the letter \mathbf{W}^k with a superscript k indicating the layer. Layers are counted from left to right and the subscripts of this matrix notation remain the same as previously. Therefore, w_{ij}^k corresponds to the weight value connecting the post-synaptic neuron i to the pre-synaptic neuron j at layer k .

In the network of Figure 12(a), \mathbf{W}^1 indicates the matrix of connections between the input layer and the first hidden layer; matrix \mathbf{W}^2 contains the connections between the first hidden layer and the second hidden layer; and matrix \mathbf{W}^3 contains the connections be-

tween the second hidden layer and the output layer. In feedforward networks the signals are propagated from the inputs to the network output in a layer-by-layer form (from left to right). Therefore, the network output is given by (in matrix notation):

$$\mathbf{y} = \mathbf{f}^3(\mathbf{W}^3 \mathbf{f}^2(\mathbf{W}^2 \mathbf{f}^1(\mathbf{W}^1 \mathbf{x}))), \quad (13)$$

where \mathbf{f}^k is the vector of activation functions of layer k assuming the nodes in a given layer can have different activation functions; \mathbf{W}^k is the weight matrix of layer k and \mathbf{x} is the input vector.

It can be observed from Equation (13) that the network output is computed by recursively multiplying the weight matrix of a given layer by the output produced by each previous layer. The expression to calculate the output of each layer is given by Equation (11) using the appropriate weight matrix and inputs.

It is important to note that if the intermediate nodes have linear activation functions there is no point in adding more layers to this network, because $f(x) = x$ for linear functions, and thus the network output would be given by

$$\mathbf{y} = \mathbf{f}^3(\mathbf{W}^3 \mathbf{W}^2 \mathbf{W}^1 \mathbf{x}).$$

and this expression can be reduced to,

$$\mathbf{y} = \mathbf{f}^3(\mathbf{W} \mathbf{x}),$$

where $\mathbf{W} = \mathbf{W}^3 \mathbf{W}^2 \mathbf{W}^1$.

Therefore, if one wants to increase the computational capabilities of a multi-layer neural network by adding more layers to the network, non-linear activation functions have to be used in the hidden layers.

3.2.3. Recurrent Networks

The third class of networks is known as recurrent networks, distinguished from feedforward networks for they have at least one *recurrent* (or *feedback*) loop. For instance, a recurrent network may consist of a single layer of neurons with each neuron feeding its output signal back to the input of other neurons, as illustrated in Figure 13. In a feedback arrangement, there is communication between neurons; there can be cross talk and plan revision; intermediate decisions can be taken; and a mutually agreeable solution can be found.

Note that the type of feedback loop illustrated in Figure 13 is distinguished from the backpropagation of error signals briefly mentioned in the previous section. The recurrent loop has an impact on the network learning capability and performance because it involves the use of particular branches composed of retard units (Z^{-1}) resulting in a non-linear dynamic behavior (assuming that the network has non-linear units).

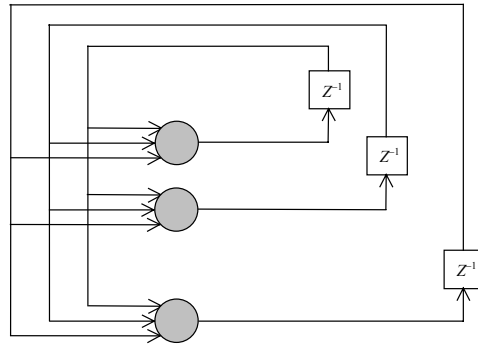


Figure 13: Recurrent neural network with no hidden layer.

3.3. Learning Approaches

It has been discussed that one of the most exciting discoveries in neuroscience was that synaptic efficiency could be modulated to the input stimulation. Also, this is believed to be the basis for learning and memory in the brain. Learning thus involves the adaptation of synaptic strengths to environmental stimuli. Biological neural networks are known not to have their architectures much altered throughout life. New neurons cannot routinely be created to store new knowledge, and the number of neurons is roughly fixed at birth, at least in mammals. The alteration of synaptic strengths may thus be the most relevant factor for learning. This change could be the simple modification in strength of a given synapse, the formation of new synaptic connections or the elimination of pre-existing synapses. However, the way learning is accomplished in the biology of the brain is still not much clear.

In the neurocomputing context, *learning* (or *training*) corresponds to the process by which the network's "free parameters" are adapted (adjusted) through a mechanism of presentation of environmental (or input) stimuli. In the standard neural network learning algorithms to be discussed here, these free parameters correspond basically to the connection strengths (weights) of individual neurons. It is important to have in mind though, that more sophisticated learning algorithms are capable of dynamically adjusting several other artificial neural network parameters, such as the network architecture and the activation function of individual neurons. The environmental or input stimuli correspond to a set of *input data* (or *patterns*) that is used to *train* the network.

Neural network learning basically implies in the following sequence of events:

- Presentation of the input patterns to the network;
- Adaptation of the network free parameters so as to produce an altered pattern of response for the input data.

In most neural network applications, the network weights are first adjusted according to a given learning rule or algorithm, and then the network is applied to a new set of input data. In this case there are two steps involved in the use of a neural network: 1) network training, and 2) network application.

With standard learning algorithms a neural network learns through an iterative process of weight adjustment. The type of learning is defined by the way in which the weights

are adjusted. The three main learning approaches are: 1) *supervised learning*, 2) *unsupervised learning*, and 3) *reinforcement learning*.

3.3.1. Supervised Learning

This learning strategy embodies the concept of a *supervisor* or *teacher*, who has the knowledge about the environment in which the network is operating. This knowledge is represented in the form of a set of *input-output samples* or *patterns*. Supervised learning is typically used when the class of data is known *a priori* and this can be used as the supervisory mechanism, as illustrated in Figure 14. The network free parameters are adjusted through the combination of the input and error signals, where the error signal is the difference between the desired output and the current network output.

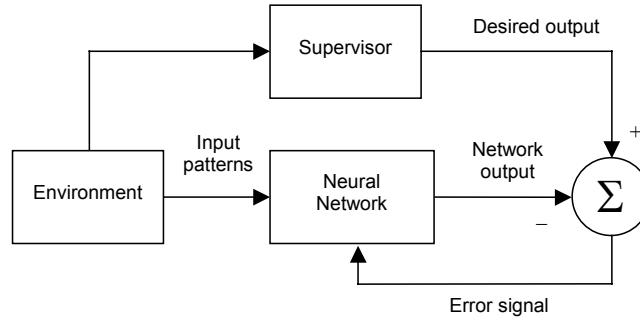


Figure 14: In supervised learning, the environment provides input patterns to train the network and a supervisor has the knowledge about the environment in which the network is operating.

Consider a simple example, let neuron j be the only output unit of a feedforward neural network, as illustrated in Figure 15. Neuron j is stimulated by a signal vector $\mathbf{x}(t)$ produced by one or more hidden layers, that are also stimulated by an input vector. Index t is the discrete time index or, more precisely, the time interval of an iterative process that will be responsible for adjusting the weights of neuron j . The only output signal $y_j(t)$, from neuron j , is compared with a *desired output*, $d_j(t)$. An error signal $e_j(t)$ is produced:

$$e_j(t) = d_j(t) - y_j(t). \quad (14)$$

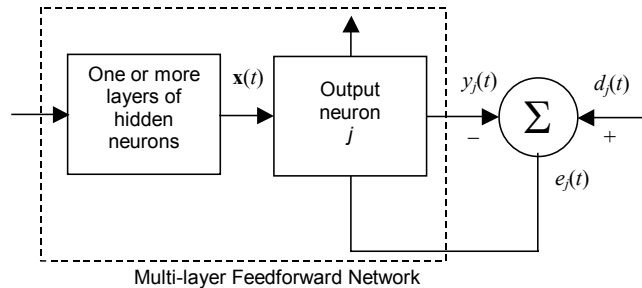


Figure 15: Error-correction learning. At each time step t , the network output is compared with the desired output, and the error is calculated and used to adjust the network response.

The error signal acts as a control mechanism responsible for the application of corrective adjustments in the weights of neuron j . The goal of supervised learning is to make the network output more similar to the desired output at each time step, i.e., to *correct the error* between the network output and the desired output. This objective can be achieved by minimizing a *cost function* (also called *performance index*, *error function*, or *objective function*), $\mathfrak{J}(t)$, which represents the instant value of the error measure:

$$\mathfrak{J}(t) = \frac{1}{2} e_j^2(t). \quad (15)$$

Generalization Capability

Once the supervised learning process is complete, it is possible to present an *unseen* pattern to the network, which will then classify that pattern, with a certain degree of accuracy, into one of the classes used in the training process. This is an important aspect of an artificial neural network, its *generalization capability*. In the present context, *generalization* refers to the performance of the network on new patterns that were not used in the network learning process.

This can be illustrated with the help of the following example. Assume we want to use a multi-layer feedforward neural network to approximate the function $\sin(x).\cos(2x)$ depicted by a solid line in Figure 16(a). It is only necessary a few training samples to adjust the weight vectors of a multi-layer feedforward neural network with a single hidden layer composed of five sigmoidal units. The network contains a single input and a single output unit, but these details are not relevant for the present discussion.

It is known that most real-world data contains noise, i.e., irrelevant or meaningless data, or data with disturbances. To investigate the relationship between noisy data and the network capability of approximating these data, some noise was added to the input data by simply adding a uniform distribution of zero mean and variance 0.15 to each input datum. Figure 16(a) shows the input data and the curve representing the original noise-free function $\sin(x).\cos(2x)$. Note that the noisy data generated deviates a little from the function to be approximated.

In this case, if the neural network is not trained sufficiently, the approximation it will provide for the input data will not be satisfactory; an *underfitting* will occur (Figure 16(b)). In the opposite case, when the network is trained until it perfectly approximates the input data, some *overfitting* will occur, meaning that the approximation is too accurate for this noisy data set (Figure 16(c)). A better training is the one that establishes a compromise between the approximation accuracy and a good generalization for unseen data (Figure 16(d)).

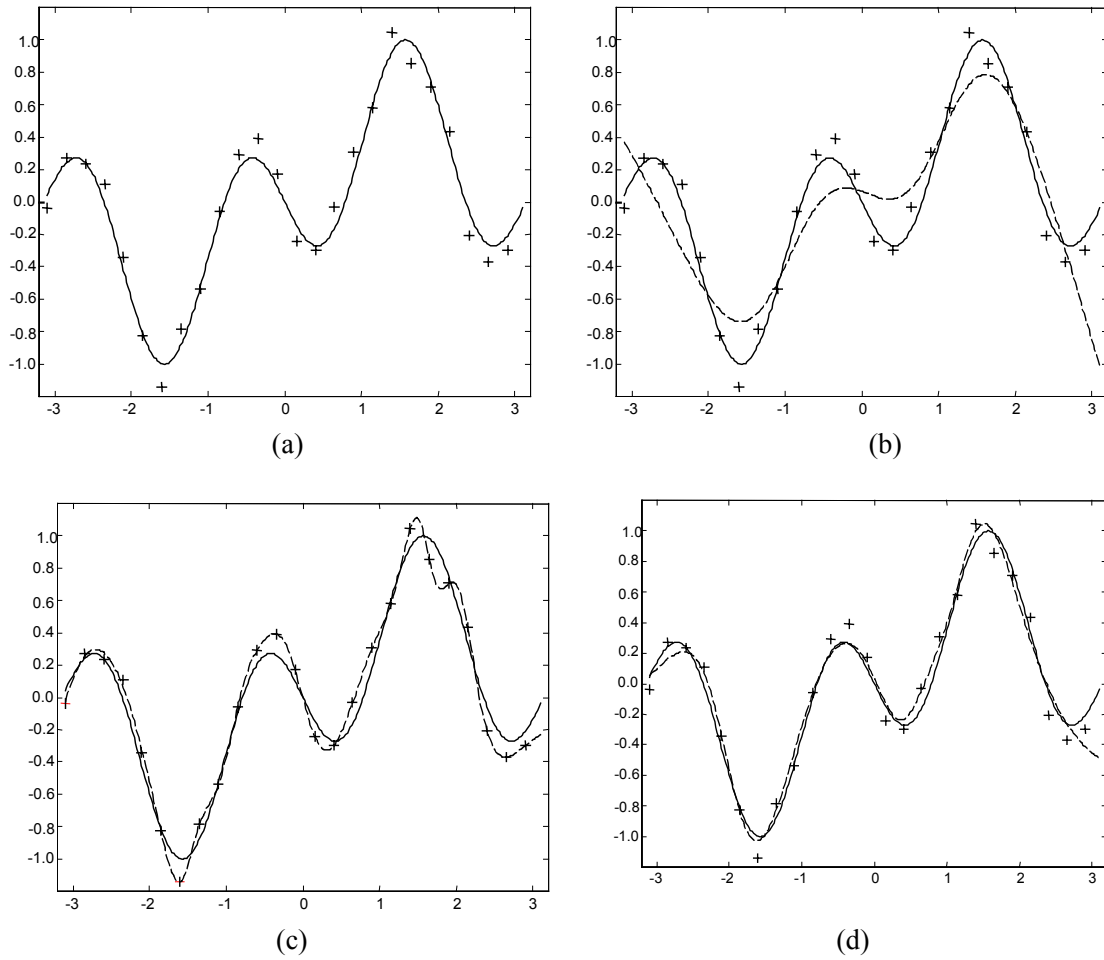


Figure 16: Function $\sin(x)\cos(2x)$ with uniformly distributed noise over the interval $[-0.15, 0.15]$. Legend: + input patterns; — desired output; - - network output. (a) Training patterns with noise and the original function to be approximated (solid line). (b) Training process interrupted too early, *underfitting* (dashed line). (c) Too much training, *overfitting* (dashed line). (d) Better trade-off between quality of approximation and generalization capability.

From a biological perspective, generalization is very important for our creation of models of the world. Think of generalization according to the following example. If you just memorize some specific facts about the world instead of trying to extract some simple essential regularity underlying these facts, then you would be in trouble when dealing with novel situations where none of the specifics appear. For instance, if you memorize a situation where you almost drowned in seawater (but you survived) as “stay away from seawater”, then you may be in trouble if you decide to swim in a pool or a river. Of course, if the problem is that you don’t know how to swim, then knowing that seawater in particular is dangerous might not be a good enough “model” of the world to prevent you from drowning in other types of water. Your internal model has to be general enough so as to suggest “stay away from water”.

3.3.2. Unsupervised Learning

In the *unsupervised* or *self-organized* learning approach, there is no supervisor to evaluate the network performance in relation to the input data set, as illustrated in Figure 17. Note that in this figure there is no error information being fed back into the network, the classes of the data are *unknown* or *unlabelled* and the presence of the supervisor no longer exists. The network adapts itself to statistical regularities in the input data, developing an ability to create internal representations that encode the features of the input data and thus, generate new classes automatically. Usually, self-organizing algorithms employ a *competitive learning* scheme.

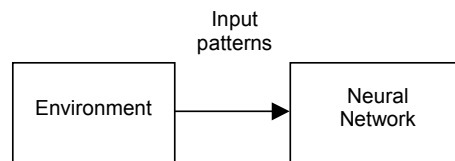


Figure 17: Unsupervised learning.

In *competitive learning*, the network output neurons compete with each other to become activated, with a single output neuron being activated at each iteration. This property makes the algorithm appropriate to discover salient statistical features within the data that can then be used to classify a set of input patterns.

There are three basic elements in a competitive learning strategy for neural networks:

- A set of neurons with the same characteristics, except for their associated weights;
- A limit imposed in the weight of each neuron; and
- A competition mechanism among neurons. The neuron that ‘wins’ the competition is called *winner*.

Individual neurons learn to specialize on ensembles (*clusters*) of similar patterns; in effect they become *feature extractors* or *feature detectors* for different classes of input patterns. In its simplest form, a *competitive neural network*, i.e., a neural network trained using a competitive learning scheme, has a single layer of output neurons that is fully connected. There are also lateral connections among neurons, as illustrated in Figure 18, capable of inhibiting neighbor neurons.

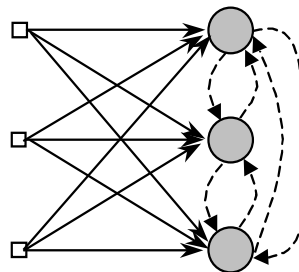


Figure 18: Simple competitive network architecture with direct excitatory (feedforward) connections (solid arrows) from the network inputs to the network outputs and inhibitory lateral connections among the output neurons (dashed arrows).

For a neuron i to be the winner, the distance between its corresponding weight vector \mathbf{w}_i and a certain input pattern \mathbf{x} must be the smallest measure (among all the network output units), given a certain metric $\|\cdot\|$ usually taken to be the Euclidean distance. Therefore, the idea is to find the output neuron whose weight vector is most similar (has the shortest distance) to the input pattern presented.

$$i = \arg \min_i \|\mathbf{x} - \mathbf{w}_i\|, \quad \forall i. \quad (16)$$

If a neuron does not respond to a determined input pattern (i.e. is not the winner), no learning takes place for this neuron. However, if a neuron i wins the competition, then an adjustment $\Delta \mathbf{w}_i$ is applied to the weight vector \mathbf{w}_i associated with the winning neuron i ,

$$\Delta \mathbf{w}_i = \begin{cases} \alpha(\mathbf{x} - \mathbf{w}_i) & \text{if } i \text{ wins the competition} \\ 0 & \text{if } i \text{ loses the competition} \end{cases} \quad (17)$$

where α is a *learning rate* that controls the step size given by \mathbf{w}_i in the direction of the input vector \mathbf{x} .

Note that this learning rule works by simply moving the weight vector of the winning neuron in the direction of the input pattern presented. More on these subjects will be presented shortly.

3.3.3. Reinforcement Learning

Reinforcement learning is distinguished from the other approaches as it relies on learning from direct interaction with the environment but does not rely on explicit supervision or complete models of the environment. Often, the only information is a scalar evaluation that indicates how well the artificial neural network is performing. This is based on a framework that defines the interaction between the neural network and its environment in terms of the current values of the network's free parameters (weights), network response (actions), and rewards. Situations are mapped into actions so as to maximize a numerical reward signal (Sutton & Barto, 1998).

Reinforcement learning is also distinguishable by the use of trial and error search and delayed reward; there is a trade-off between *exploration* and *exploitation*. To obtain more reward, a reinforcement learning network must prefer actions that it has tried in the past and found to be effective in producing reward. In contrast, to discover such actions it has to try actions that it has not selected previously. Thus, the network has to exploit what it already knows in order to obtain reward, but it also has to explore in order to make better action selections in the future. Figure 19 illustrates the network-environment interaction in a reinforcement learning system, highlighting that the network output is fed into the environment that provides it with a reward signal according to how well the network is performing.

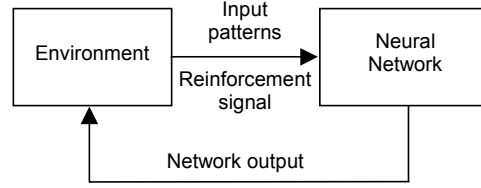


Figure 19: In reinforcement learning, the network output provides the environment with information about how well the neural network is performing.

4. Typical Artificial Neural Networks

The previous section introduced the fundamentals of neurocomputing. It was argued that an artificial neural network can be designed by 1) choosing some abstract models of neurons, 2) defining a network architecture, and 3) choosing an appropriate learning algorithm. In the present section, some of the most commonly used neural networks will be described, focusing on the main three aspects above: type of neuron, network architecture, and learning algorithm.

4.1. Hebbian Network

After the 1943 McCulloch and Pitts' paper describing a logical calculus of neuronal activity, N. Wiener published a famous book named *Cybernetics* in 1948, followed by the publication of Hebb's book *The Organization of Behavior*. These were some landmarks in the history of neurocomputing.

In Hebb's book, an explicit statement of a physiological learning rule for synaptic modification was presented for the first time. Hebb proposed that the connectivity of the brain is continually changing as an organism learns distinct functional tasks, and that neural assemblies are created by such changes. Hebb stated that the effectiveness of a variable synapse between two neurons is increased by the repeated activation of one neuron by the other across that synapse. Quoting from Hebb's book *The Organization of Behavior*:

“When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency as one of the cells firing B, is increased.” (Hebb, 1949; p. 62)

This postulate requires that change occur in the synaptic strength between cells when the pre-synaptic and post-synaptic cells are active at the same time. Hebb suggested that this change is the basis for *associative learning* that would result in a lasting modification in the activity pattern of a spatially distributed assemble of neurons.

This rule is often approximated in artificial neural networks by the *generalized Hebb rule*, where changes in connection strengths are given by the product of pre-synaptic and post-synaptic activity. The main difference being that change is now a result of both stimulatory and inhibitory synapses, not only excitatory synapses. In mathematical terms,

$$\Delta w_{ij}(t) = \alpha y_i(t) x_j(t), \quad (18)$$

where Δw_{ij} is the change to be applied to neuron i , α is a *learning rate* parameter that determines how much change should be applied, y_i is the output of neuron i , x_j is the input to neuron j , and t is the time index. Equation (18) can be expressed generically as

$$\Delta w_{ij}(t) = g(y_i(t), x_j(t)) \quad (19)$$

where $g(\cdot, \cdot)$ is a function of both pre- and post-synaptic signals. The weight of a given neuron i is thus updated according to the following rule:

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t). \quad (20)$$

Equation (18) clearly emphasizes the correlational or associative nature of Hebb's updating rule. It is known that much of human memory is *associative*, just as the mechanism suggested by Hebb. In an associative memory, an event is linked to another event, so that the presentation of the first event gives rise to the linked event. In the most simplistic version of association, a *stimulus* is linked to a *response*, so that a later presentation of the stimulus evokes the response.

4.1.1. Biological Basis of Hebbian Synaptic Modification

Physiological evidence suggests the existence of Hebb synapses at various locations in the brain. Advances in modern neurophysiological techniques have allowed us to see what appears to be Hebbian modification in several parts of the mammalian brain (Anderson, 1995). A part of the cerebral cortex, the *hippocampus*, shows an effect called *long-term potentiation*, in which its neurons can be induced to display long-term, apparently permanent, increases in activity with particular patterns of stimulation.

Kelso et al. (1986) have presented an experimental demonstration of Hebbian modification using a slice of rat hippocampus maintained outside the animal. They showed that when a pre-synaptic cell was excited and the post-synaptic cells inhibited, little or no change was seen in the efficiency of a synapse. If the post-synaptic cell was excited by raising the membrane potential at the same time the pre-synaptic cell was active, the excitatory post-synaptic potential from the pre-synaptic cell was significantly increased in a stable and long-lasting form.

4.2. Single-Layer Perceptron

Rosenblatt (1958, 1962) introduced the *perceptron* as the simplest form of a neural network used for the classification of *linearly separable* patterns. Perceptrons constituted the first model of supervised learning, though some perceptrons were self-organized. Before describing the perceptron learning algorithm, the next section discusses in some more detail the problem of linear separability.

4.2.1. Linear Separability

Linear separability can be easily understood with a simple example. Without loss of generality, assume there is a set of input patterns to be classified into a single class. Assume also there is a classifier system that has to respond TRUE if a given input pattern is a member of a certain class, and FALSE if it is not. A TRUE response is represented by an output '1' and a FALSE response by an output '0' of the classifier.

As one of two responses is required, there is a *decision boundary* that divides one response from the other. Depending on the number m of variables of the input patterns, the decision boundary can have any shape in the space of dimension m . If $m = 2$ and the classifier can be set up so that there is a line dividing all input patterns that produce output '1' from those patterns that produce output '0', then the problem is *linearly separable*; else it

is *non-linearly separable*. This holds true for a space of any dimension, but for higher dimensions, a plane or hyper-plane should exist so as to divide the space into a ‘1’-class and a ‘0’-class region.

Figure 20 illustrates one linearly separable function and one non-linearly separable function. Note that, in Figure 20(a), several lines could be drawn so as to separate the data into class ‘1’ or ‘0’. The regions that separate the classes to which each of the input patterns belong to are often called *decision regions*, and the (hyper-)surface that defines these regions are called *decision surfaces*. Therefore, a classification problem can be viewed as the problem of finding decision surfaces that correctly classify the input data. The difficulty lying in that it is not always trivial to automatically define such surfaces.

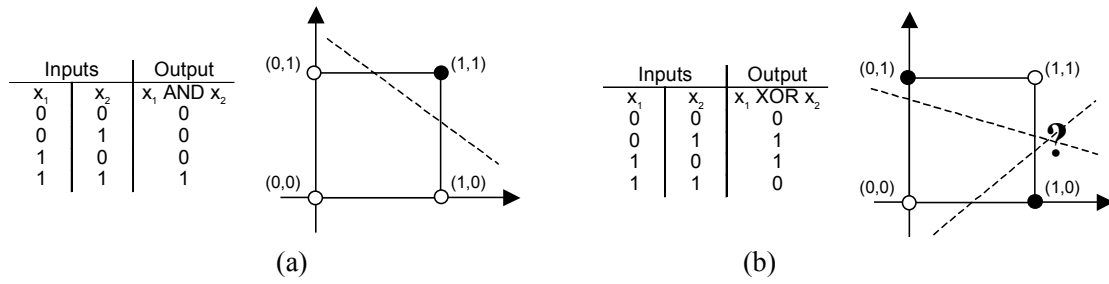


Figure 20: Examples of linear and non-linear separability. (a) Logic function AND. (b) Logic function XOR.

4.2.2. Simple Perceptron for Pattern Classification

Basically, the perceptron consists of a single layer of neurons with adjustable synaptic weights and biases. Under suitable conditions, in particular if the training patterns belong to linearly separable classes, the iterative procedure of adaptation for the perceptron can be proved to converge to the correct weight set. These weights are such that the perceptron algorithm converges and positions the decision surfaces in the form of (hyper-)planes between the classes. This proof of convergence is known as the *perceptron convergence theorem*.

The simple perceptron has a single layer of feedforward neurons as illustrated in Figure 11(a). The neurons in this network are similar to those of McCulloch and Pitts with a threshold activation function, but include a bias. The key contribution of Rosenblatt and his collaborators was to introduce a learning rule to train the perceptrons to solve pattern recognition and classification problems.

Their algorithm works as follows. For each training pattern \mathbf{x}_i , the network output response y_i is calculated. Then, the network determines if an error e_i occurred for this pattern by comparing the calculated output y_i with a *desired value* d_i for that pattern, $e_i = d_i - y_i$. Note that the desired output value for each training pattern is known. The weight vector connecting the inputs (pre-synaptic neuron) to each output (post-synaptic) neuron and the neuron’s bias are updated according to the following rules:

$$\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + \alpha e_i \mathbf{x}_i, \quad (21)$$

$$b(t+1) = b(t) + \alpha e_i, \quad (22)$$

where $\mathbf{w} \in \mathbb{R}^{1 \times m}$, $\mathbf{x} \in \mathbb{R}^{1 \times m}$, and $b \in \mathbb{R}^{1 \times 1}$.

Consider now the simplest perceptron case with a single neuron, as illustrated in Figure 21. The goal of this network, more specifically neuron, is to classify a number of input patterns as belonging or not belonging to a particular class. Assume that the input data set is given by the N pairs of samples $\{\mathbf{x}_1, d_1\}, \{\mathbf{x}_2, d_2\}, \dots, \{\mathbf{x}_N, d_N\}$, where \mathbf{x}_j is the input vector j and d_j its corresponding *target* or *desired* output. In this case, the target value is ‘1’ if the pattern belongs to the class, and ‘0’ otherwise.

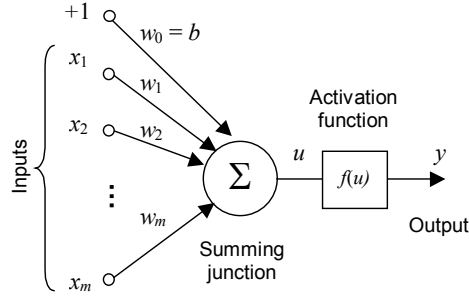


Figure 21: The simplest perceptron to perform pattern classification.

Let $\mathbf{X} \in \mathbb{R}^{m \times N}$, be the matrix of N input patterns of dimension m each (the patterns are placed column wise in matrix \mathbf{X}), and $\mathbf{d} \in \mathbb{R}^{1 \times N}$ be the vector of desired outputs. The learning algorithm for this simple perceptron network is presented in Algo 1, where $f(\cdot)$ is the threshold function (Figure 10(b)). The stopping criterion for this algorithm is either a fixed number of iteration steps – `max_it` – or the sum E of the errors e_i , $i = 1, \dots, N$, for each input pattern being equal to zero. The parameter α is a *learning rate* that determines the step size of the adaptation in the weight vector and bias term.

```

procedure [ $\mathbf{w}$ ] = perceptron(max_it,  $E$ ,  $\alpha$ ,  $\mathbf{X}$ ,  $\mathbf{d}$ )
    initialize  $\mathbf{w}$     //for simplicity set it to zero
    initialize  $b$     //for simplicity set it to zero
     $t \leftarrow 1$ 
    while  $t < \text{max\_it} \ \& \ E > 0$  do,
        for  $i$  from 1 to  $N$  do,    //for each training pattern
             $y_i \leftarrow f(\mathbf{w}\mathbf{x}_i + b)$     //determine the network output for  $\mathbf{x}_i$ 
             $e_i \leftarrow d_i - y_i$     //determine the network error for  $\mathbf{x}_i$ 
             $\mathbf{w} \leftarrow \mathbf{w} + \alpha e_i \mathbf{x}_i$     //update the weight vector
             $b \leftarrow b + \alpha e_i$     //update the bias term
        end for
         $E \leftarrow \text{sum}(e_i)$ 
         $t \leftarrow t + 1$ 
    end while
end procedure

```

Algo 1: Simple perceptron learning algorithm. (The function $f(\cdot)$ is the threshold function)

4.2.3. Multiple Output Perceptron for Pattern Classification

Note that the perceptron updating rule uses the error-correction learning of most supervised learning techniques, as discussed in Section 3.3.1. This learning rule, proposed to update the weight vector of a single neuron, can be easily extended to deal with networks with more than one output neuron, such as the network presented in Figure 11(a). In this case, for each input pattern \mathbf{x}_i , the vector of network outputs is given by Equation (12) explicitly including the bias vector \mathbf{b} as follows:

$$\mathbf{y} = f(\mathbf{W}\mathbf{x}_i + \mathbf{b}), \quad (23)$$

where $\mathbf{W} \in \mathbb{R}^{o \times m}$, $\mathbf{x}_i \in \mathbb{R}^{m \times 1}$, $i = 1, \dots, N$, $\mathbf{y} \in \mathbb{R}^{o \times 1}$, and $\mathbf{b} \in \mathbb{R}^{o \times 1}$. The function $f(\cdot)$ is the threshold function presented in Figure 10(b).

Let the matrix of desired outputs be $\mathbf{D} \in \mathbb{R}^{o \times N}$, where each column of \mathbf{D} corresponds to the desired output for one of the input patterns. Therefore, the error signal for each input pattern \mathbf{x}_i is calculated by simply subtracting the vectors \mathbf{d}_i and \mathbf{y}_i that are of same dimension: $\mathbf{e}_i = \mathbf{d}_i - \mathbf{y}_i$, where $\mathbf{e}_i, \mathbf{d}_i, \mathbf{y}_i \in \mathbb{R}^{o \times 1}$. The algorithm to implement the perceptron with multiple output neurons is presented in Algo 2.

```

procedure [W] = perceptron(max_it,  $\alpha$ ,  $\mathbf{X}$ ,  $\mathbf{D}$ )
    initialize  $\mathbf{W}$     //for simplicity set it to zero
    initialize  $\mathbf{b}$     //for simplicity set it to zero
    t  $\leftarrow$  1
    while t < max_it do,
        for i from 1 to N do,    //for each training pattern
             $\mathbf{y}_i \leftarrow f(\mathbf{W}\mathbf{x}_i + \mathbf{b})$     //determine the network outputs for  $\mathbf{x}_i$ 
             $\mathbf{e}_i \leftarrow \mathbf{d}_i - \mathbf{y}_i$     //determine the network error for  $\mathbf{x}_i$ 
             $\mathbf{W} \leftarrow \mathbf{W} + \alpha \mathbf{e}_i \mathbf{x}_i^T$     //update the weight matrix
             $\mathbf{b} \leftarrow \mathbf{b} + \alpha \mathbf{e}_i$     //update the bias vector
        end for
        t  $\leftarrow$  t + 1
    end while
end procedure

```

Algo 2: Learning algorithm for the perceptron with multiple outputs.

4.2.4. Examples of Application

To illustrate the applicability of the perceptron network, consider the two problems below. The first example – simple classification problem – illustrates the potentiality of the perceptron to represent Boolean functions, and the second example – character recognition – illustrates its capability to recognize a simple set of binary characters.

Simple Classification Problem

Consider the problem of using the simple perceptron with a single neuron to represent the AND function. The training data and its graphical interpretation are presented in Figure 22. The training patterns to be used as inputs to Algo 1 are, in matrix notation:

$$\mathbf{X} = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} \quad \mathbf{d} = [0 \quad 0 \quad 0 \quad 1].$$

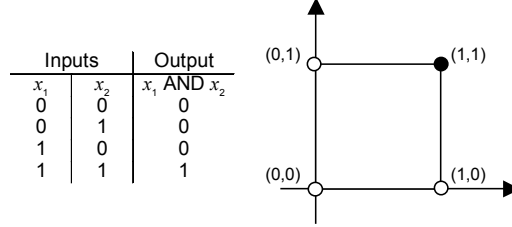


Figure 22: The AND function and its graphical representation.

The boundary between the values of x_1 and x_2 for which the network provides a response ‘0’ (not belonging to the class) and the values for which the network responds ‘1’ (belonging to the class) is the separating line given by

$$x_1 w_1 + x_2 w_2 + b = 0. \quad (24)$$

As the activation function of the network is the threshold with $\theta = 0$, the requirement for a positive response from the output unit is that the net input it receives be greater than or equal to zero, that is, $x_1 w_1 + x_2 w_2 + b \geq 0$. During training, the values of w_1 , w_2 and b are determined so that the network will present the correct response for all training data.

For simplicity, the learning rate is set to 1, $\alpha = 1$, and the initial values for the weights and biases are taken to be zero, $\mathbf{w} = [0 \ 0]$ and $b = 0$. By running Algo 1 with \mathbf{X} , \mathbf{d} and the other parameters given above, the following values are obtained for the perceptron network with a single output.

$$w_1 = 2; w_2 = 1; b = -3.$$

By replacing these values in the line equation for this neuron we obtain

$$2x_1 + 1x_2 - 3 = 0.$$

For each pair of training data, the following net input u_i , $i = 1, \dots, N$, is calculated: $\mathbf{u} = [-3, -2, -1, 0]$. Therefore, the network response, $\mathbf{y} = f(\mathbf{u})$, for each input pattern is $\mathbf{y} = [0, 0, 0, 1]$, realizing the AND function as desired.

The decision line between the two classes can be determined by isolating the variable x_2 as a function of the other variable in Equation (24),

$$x_2 = -\frac{w_1}{w_2}x_1 - \frac{b}{w_2} = -2x_1 + 3.$$

Note that this line passes exactly on top of class ‘1’, as illustrated in Figure 23.

An alternative form of initializing the weights for the perceptron network is by choosing random values within a given domain. For instance, a uniform or normal distribution with zero mean and pre-defined variance (usually less than or equal to one) could be used. Assume now that the following initial values for \mathbf{w} and b were chosen using a uniform distribution of zero mean and variance one:

$$w_1 = 0.015; w_2 = 0.768; b = 0.971.$$

After training the network, the following values for the weights were determined:

$$w_1 = 2.015; w_2 = 0.768; b = -2.029.$$

These values define the decision line below, as depicted in Figure 23.

$$x_2 = -\frac{w_1}{w_2}x_1 - \frac{b}{w_2} = -2.624x_1 + 2.642.$$

Two important issues can be noticed here. First, the network response is dependent upon the initial values chosen for its weights and biases. Second, there is not a single correct response for a given training data set. These and other important issues in neural network training will be discussed later.

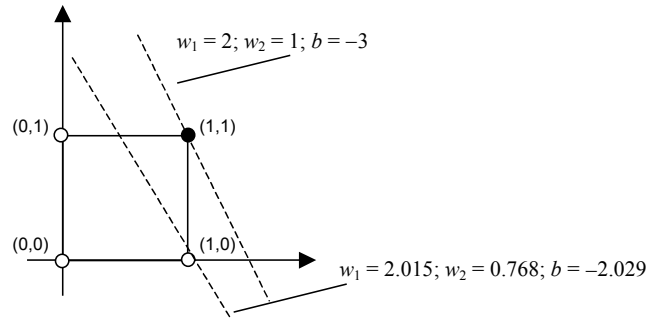


Figure 23: Decision lines for the simple perceptron network realizing the logic function AND.

Character Recognition

As a second example of using the perceptron for pattern classification, consider the 8 input patterns presented in Figure 24. Each input vector is a 120-tuple representing a letter expressed as a pattern on a 12×10 grid of pixels. Each character is assumed to have its own class. One output unit is assigned to each character/class, thus there are eight categories to which each character could be assigned. Matrix $\mathbf{X} \in \mathfrak{R}^{120 \times 8}$, where each row of \mathbf{X} corresponds to an input pattern, and matrix $\mathbf{D} \in \mathfrak{R}^{8 \times 8}$ is a diagonal matrix, meaning that input pattern number 1 (character '0') should activate only output 1, input pattern number 2 (character '1') should activate only output 2, and so on.

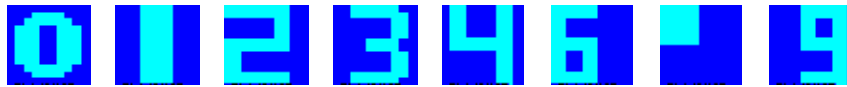


Figure 24: Training patterns for the perceptron.

After training, the network correctly classifies each of the training patterns. Algo 3 can be used to run the network in order to evaluate if it is correctly classifying the training patterns. Note that in this case, the network is first trained and then applied to classify novel data, that is, its generalization capability for unseen data can be evaluated.

```

procedure [y] = perceptron(W,b,z)
  for i from 1 to N do,           //for each training pattern
     $y_i \leftarrow f(Wx_i + b)$       //determine the network outputs for  $x_i$ 
  end for
end procedure

```

Algo 3: Algorithm used to run the trained single-layer perceptron network.

To test how this network generalizes, some random *noise* can be inserted into the training patterns by simply changing a value ‘0’ by ‘1’ or a value ‘1’ by ‘0’, respectively, with a given probability. Figure 25 illustrates the training patterns with 5% noise.

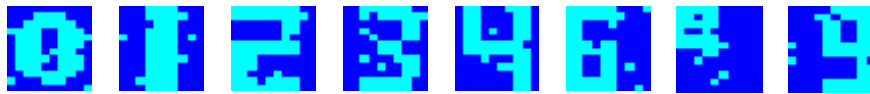


Figure 25: Noisy patterns used to test the generalization capability of the single-layer perceptron network trained to classify the patterns presented in Figure 24.

4.3. ADALINE, the LMS Algorithm, and Error Surfaces

Almost at the same time Rosenblatt introduced the perceptron learning rule, B. Widrow and his student M. Hoff (Widrow & Hoff, 1960) developed the Widrow-Hoff learning rule, also called the *least mean squared (LMS) algorithm* or *delta rule*. They introduced the ADALINE (ADAPtive LInear NEuron) network that is very similar to the perceptron, except that its activation function is linear instead of threshold. Although both networks, ADALINE and perceptron, suffer from only being capable of solving linear separable problems, the LMS algorithm is more powerful than the perceptron learning rule and has found many more practical uses than the perceptron.

As the ADALINE employs neurons with a linear activation function, the neuron’s output is equal to its net input. The goal of the learning algorithm is to minimize the *mean squared error* (MSE) between the network output and the desired output. This allows the network to perform a continuous learning even after a given input pattern has been learnt. One advantage of the LMS algorithm over the perceptron learning rule is that the resultant network after training is not too sensitive to noise. It could be observed in Figure 23, that the decision surfaces generated by the perceptron algorithm usually lie very close to some input data. This is very much the case for the perceptron algorithm, but is not the case for the LMS algorithm, making the latter more robust to noise. In addition, the LMS algorithm has been broadly used in signal processing applications.

4.3.1. LMS Algorithm (Delta Rule)

The LMS algorithm or delta rule is another example of supervised learning in which the learning rule is provided with a set of inputs and corresponding desired outputs $\{x_1, d_1\}$, $\{x_2, d_2\}$, ..., $\{x_N, d_N\}$, where x_j is the input vector j and d_j its corresponding *target* or *desired* output vector.

The delta rule adjusts the neural network weights and biases so as to minimize the difference (error) between the network output and the desired output over all training pat-

terns. This is accomplished by reducing the error for each pattern, one at a time, though weight corrections can also be accumulated over a number of training patterns.

The *sum-squared error* (SSE) for a particular training pattern is given by

$$\mathfrak{I} = \sum_{i=1}^o e_i^2 = \sum_{i=1}^o (d_i - y_i)^2 . \quad (25)$$

The gradient of \mathfrak{I} , also called the *performance index* or *cost function*, is a vector consisting of the partial derivatives of \mathfrak{I} with respect to each of the weights. This vector gives the direction of most rapid increase of \mathfrak{I} ; thus, its opposite direction is the one of most rapid decrease in the error value. Therefore, the error can be reduced most rapidly by adjusting the weight w_{IJ} in the following manner

$$w_{IJ} = w_{IJ} - \alpha \frac{\partial \mathfrak{I}}{\partial w_{IJ}} ,$$

where w_{IJ} is the weight from the J -th pre-synaptic neuron to the I -th post-synaptic neuron, and α is a learning rate.

It is necessary now to explicitly determine the gradient of the error with respect to the arbitrary weight w_{IJ} . As the weight w_{IJ} only influences the output (post-synaptic) unit I , the gradient of the error is

$$\begin{aligned} \frac{\partial \mathfrak{I}}{\partial w_{IJ}} &= \frac{\partial}{\partial w_{IJ}} \sum_{i=1}^o (d_i - y_i)^2 \\ &= \frac{\partial}{\partial w_{IJ}} (d_I - y_I)^2 \end{aligned}$$

It is known that

$$y_I = f(\mathbf{w}_I \cdot \mathbf{x}) = f(\sum_j w_{Ij} x_j)$$

Then, we obtain

$$\begin{aligned} \frac{\partial \mathfrak{I}}{\partial w_{IJ}} &= -2(d_I - y_I) \frac{\partial y_I}{\partial w_{IJ}} . \\ &= -2(d_I - y_I) x_J \end{aligned}$$

Therefore, similarly to the updating rule of Hebb's network, the LMS also assumes an updating value Δw_{IJ} to be added to the weight w_{IJ} . The delta rule for updating the weight from the J -th pre-synaptic neuron to the I -th post-synaptic neuron is given by

$$\Delta w_{IJ} = \alpha (d_I - y_I) x_J . \quad (26)$$

$$w_{IJ} = w_{IJ} + 2\alpha (d_I - y_I) x_J . \quad (27)$$

The following equation for updating the bias b_I can be obtained by calculating the gradient of the error in relation to the bias b_I

$$b_I = b_I + 2\alpha (d_I - y_I) . \quad (28)$$

For each input pattern \mathbf{x}_i , the LMS algorithm can be conveniently written in matrix notation as

$$\begin{aligned} \mathbf{W} &= \mathbf{W} + \alpha \mathbf{e}_i \mathbf{x}_i^T \\ \mathbf{b} &= \mathbf{b} + \alpha \mathbf{e}_i \end{aligned}$$

where $\mathbf{W} \in \mathbb{R}^{o \times m}$, $\mathbf{x}_i \in \mathbb{R}^{m \times 1}$, $i = 1, \dots, N$, $\mathbf{e}_i \in \mathbb{R}^{o \times 1}$, and $\mathbf{b} \in \mathbb{R}^{o \times 1}$. (Note that 2α was replaced by α .)

The beauty of this algorithm is that at each iteration it calculates the approximate of the gradient vector by simply multiplying the error by the input vector. And this approximation to the gradient can be used in a steepest descent-like algorithm with fixed learning rate.

4.3.2. Error Surfaces

Assume a neural network with n weights. This set of weights can be viewed as a point in an n -dimensional space, called *weight space*. If the neural network is used to classify a set of patterns, for each of these patterns the network will generate an error signal. This means that every set of weights (and biases) has an associated scalar error value; if the weights are changed, a new error value is determined. The error values for every set of weights define a surface in the weight space, called the *error surface* (Anderson, 1995; Hagan et al., 1996).

The question that arises now is what is the function of learning? The way the LMS algorithm, and the backpropagation algorithm to be explained in the next section, view learning is as the minimization of the error to its smallest value possible. This corresponds to finding an appropriate weight set that leads to the smallest error in the error surface. Figure 26 depicts an error surface in weight space. Note that this surface potentially has many *local minima* and one or more *global minimum*. Any gradient-like method, such as the LMS and the backpropagation algorithm can only converge to local optima solutions. Therefore, the choice of an appropriate initial weight set for the network is crucial.

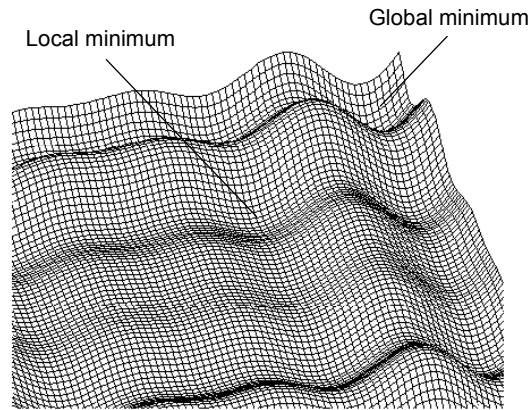


Figure 26: An error surface in weight space. The total error is a function of the values of the weights, so every set of weights has an associated error.

4.4. Multi-Layer Perceptron

The multi-layer perceptron is a kind of multi-layer feedforward network such as the one illustrated in Figure 12. Typically, the network consists of a set of input units that constitute the input layer, one or more hidden layers, and an output layer. The input signal

propagates through the network in a forward direction, layer by layer. This network is a generalization of the single-layer perceptron discussed previously.

In the late 1960's, M. Minsky and S. Papert (1969) released a book called *Perceptrons* demonstrating the limitations of single-layered feedforward networks, namely, the incapability of solving non-linearly separable problems. This caused a significant impact in the interest in neural network research during the 1970s. Both, Rosenblatt and Widrow were aware of the limitations of the perceptron network and proposed multi-layer networks to overcome this limitation. However, they were unable to generalize their algorithms to train these more powerful networks.

It has already been discussed that a multi-layer network with linear nodes is equivalent to a single-layered network with linear units. Therefore, one necessary condition for the multi-layer network to be more powerful than the single-layer networks is the use of non-linear activation functions. More specifically, multi-layer networks typically employ a sigmoidal activation function (Figure 10).

Multi-layer perceptrons (MLP) have been applied successfully to a number of problems by training them with a supervised learning algorithm known as the *error backpropagation* or simply *backpropagation*. This algorithm is also based on an error-correction learning rule and, thus, can be viewed as a generalization of the LMS algorithm or delta rule.

The backpropagation algorithm became very popular with the publication of the *Parallel Distributed Processing* volumes by Rumelhart and collaborators (Rumelhart et al., 1986; McClelland et al., 1986). These books were also crucial for the re-emergence of interest for the research on neural networks. Basically, the error backpropagation learning consists of two passes of computation, a *forward* and a *backward* pass (see Figure 12(b)). In the forward pass, an input pattern is presented to the network and its effects are propagated through the network until they produce the network output(s). In the backward pass, the synaptic weights, so far kept fixed, are updated in accordance with an error correction rule.

4.4.1. The Backpropagation Learning Algorithm

To derive the backpropagation learning algorithm, consider the following notation:

i, j	Indices referring to neurons in the network
t	t -th input vector (iteration)
N	Number of training patterns
M	Number of layers in the network
$y_j(t)$	Output signal of neuron j at iteration t
$e_j(t)$	Error signal of output unit j at iteration t
$w_{ji}(t)$	Synaptic weight connecting the output of unit j to the input of unit i at iteration t
$u_j(t)$	Net input of unit j at iteration t
$f_j(\cdot)$	Activation function of unit j
\mathbf{X}	Matrix of input (training) patterns
\mathbf{D}	Matrix of desired outputs
$x_i(t)$	i -th element of the input vector at iteration t

$d_j(t)$ j -th element of the output vector at iteration t
 α Learning rate

The description to be presented here follows that of Hagan et al. (1996) and de Castro (1998).

For multi-layer networks, the output of a given layer is the input to the next layer,

$$\mathbf{y}^{m+1} = \mathbf{f}^{m+1} (\mathbf{W}^{m+1} \mathbf{y}^m + \mathbf{b}^{m+1}), m = 0, 1, \dots, M-1, \quad (29)$$

where M is the number of layers in the network, and the superindex m refers to the layer taken into account (e.g., $m = 0$: input layer, $m = 1$: first hidden layer, ..., $m = M-1$: output layer). The nodes in the input layer ($m = 0$) receive the input patterns

$$\mathbf{y}^0 = \mathbf{x}, \quad (30)$$

that represents the initial condition for Equation (29). The outputs in the output layer are the network outputs:

$$\mathbf{y} = \mathbf{y}^M. \quad (31)$$

Equation (13) demonstrates that the network output can be a function of only the input vector \mathbf{x} and the weight matrices \mathbf{W}^m . If we explicitly consider the bias terms \mathbf{b}^m , Equation (13) becomes Equation (32) for the network of Figure 27:

$$\mathbf{y} = \mathbf{y}^3 = \mathbf{f}^3 (\mathbf{W}^3 \mathbf{f}^2 (\mathbf{W}^2 \mathbf{f}^1 (\mathbf{W}^1 \mathbf{x} + \mathbf{b}^1) + \mathbf{b}^2) + \mathbf{b}^3). \quad (32)$$

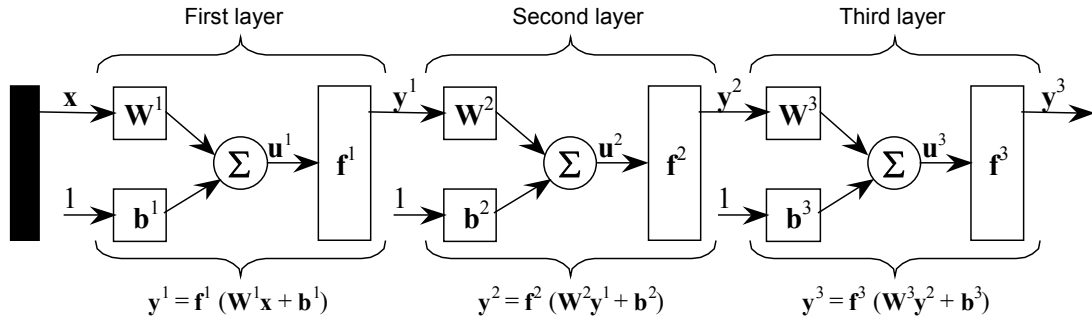


Figure 27: An MLP network with three layers. (Matrix notation.)

The backpropagation algorithm to MLP networks is a generalization of the LMS algorithm that uses as a performance index the *mean squared error* (MSE). A set of input patterns and desired outputs is provided

$$\{(\mathbf{x}_1, \mathbf{d}_1), (\mathbf{x}_2, \mathbf{d}_2), \dots, (\mathbf{x}_N, \mathbf{d}_N)\},$$

where \mathbf{x}_i is the i -th input to the network and \mathbf{d}_i is the corresponding desired output, $i = 1, \dots, N$. From Equation (32) we can observe that, if ω is the vector of network parameters (weights and biases), then the network output can be given as a function of ω and x

$$\mathbf{y}^M = f(\omega, x).$$

After each training pattern is presented to the network, the network output is compared with the desired output. The algorithm must adjust the vector of parameters so as to minimize the mathematical expectation of the mean squared error:

$$\mathfrak{J}(\omega) = E(e(\omega)^2) = E((d - y(\omega))^2). \quad (33)$$

If the network has multiple outputs, Equation (33) becomes

$$\mathfrak{J}(\omega) = E(\mathbf{e}(\omega)^T \mathbf{e}(\omega)) = E((\mathbf{d} - \mathbf{y}(\omega))^T (\mathbf{d} - \mathbf{y}(\omega))).$$

Similarly to the LMS algorithm, the mean squared error can be approximated by

$$\hat{\mathfrak{J}}(\omega) = \mathbf{e}(t)^T \mathbf{e}(t) = (\mathbf{d}(t) - \mathbf{y}(t))^T (\mathbf{d}(t) - \mathbf{y}(t)),$$

where the expectation of the mean squared error was substituted by the error at iteration t . In order not to overload the notation, assume $\hat{\mathfrak{J}}(\omega) = \mathfrak{J}(\omega)$.

The updating rule known as steepest descent to minimize the squared error is given by

$$w_{ij}^m(t+1) = w_{ij}^m(t) - \alpha \frac{\partial \mathfrak{J}(t)}{\partial w_{ij}^m}, \quad (34)$$

$$b_i^m(t+1) = b_i^m(t) - \alpha \frac{\partial \mathfrak{J}(t)}{\partial b_i^m}, \quad (35)$$

where α is the learning rate.

The most elaborate part of the determination of the partial derivatives that will produce the components of the gradient vector. To determine these derivatives it will be necessary to use the chain rule.

The Chain Rule

For a multi-layer network, the error is not a direct function of the weights in the hidden layers, reason why the calculus of these derivatives is not straightforward.

As the error is an indirect function of the weights in the hidden layers, the chain rule must be used to determine the derivatives. The chain rule will be used to determine the derivatives in Equations (34) and (35).

$$\frac{\partial \mathfrak{J}}{\partial w_{ij}^m} = \frac{\partial \mathfrak{J}}{\partial u_i^m} \times \frac{\partial u_i^m}{\partial w_{ij}^m}, \quad (36)$$

$$\frac{\partial \mathfrak{J}}{\partial b_i^m} = \frac{\partial \mathfrak{J}}{\partial u_i^m} \times \frac{\partial u_i^m}{\partial b_i^m}. \quad (37)$$

The second term of both equations above can be easily determined for the net input of layer m as an explicit function of the weights and biases in this layer:

$$u_i^m = \sum_{j=1}^{S^{m-1}} w_{ij}^m y_j^{m-1} + b_i^m, \quad (38)$$

where S^m is the number of neurons in layer m .

Therefore,

$$\frac{\partial u_i^m}{\partial w_{ij}^m} = y_j^{m-1}, \frac{\partial u_i^m}{\partial b_i^m} = 1. \quad (39)$$

Now define the *sensitivity* (δ) of \mathfrak{J} to changes in the i -th element of the net input in layer m as

$$\delta_i^m \equiv \frac{\partial \mathfrak{S}}{\partial u_i^m}, \quad (40)$$

Equations (36) and (37) can now be simplified by

$$\frac{\partial \mathfrak{S}}{\partial w_{ij}^m} = \delta_i^m y_j^{m-1}, \quad (41)$$

$$\frac{\partial \mathfrak{S}}{\partial b_i^m} = \delta_i^m. \quad (42)$$

Thus Equations (34) and (35) become

$$w_{ij}^m(t+1) = w_{ij}^m(t) - \alpha \delta_i^m y_j^{m-1}, \quad (43)$$

$$b_i^m(t+1) = b_i^m(t) - \alpha \delta_i^m. \quad (44)$$

In matrix notation these equations become

$$\mathbf{W}^m(t+1) = \mathbf{W}^m(t) - \alpha \boldsymbol{\delta}^m (\mathbf{y}^{m-1})^T, \quad (45)$$

$$\mathbf{b}^m(t+1) = \mathbf{b}^m(t) - \alpha \boldsymbol{\delta}^m, \quad (46)$$

where

$$\boldsymbol{\delta}^m \equiv \frac{\partial \mathfrak{S}}{\partial \mathbf{u}^m} = \begin{bmatrix} \frac{\partial \mathfrak{S}}{\partial u_1^m} \\ \frac{\partial \mathfrak{S}}{\partial u_2^m} \\ \vdots \\ \frac{\partial \mathfrak{S}}{\partial u_{S^m}^m} \end{bmatrix}. \quad (47)$$

Backpropagating the Sensitivities

It is now necessary to calculate the sensitivity $\boldsymbol{\delta}^m$ that requires another application of the chain rule. This process gives rise to the term *backpropagation* because it describes a recurrence relationship in which the sensitivity of layer m is determined from the sensitivity of layer $m+1$.

To derive the recurrence relationship for the sensitivities, we will use the following Jacobian matrix

$$\frac{\partial \mathbf{u}^{m+1}}{\partial \mathbf{u}^m} = \begin{bmatrix} \frac{\partial u_1^{m+1}}{\partial u_1^m} & \frac{\partial u_1^{m+1}}{\partial u_2^m} & \dots & \frac{\partial u_1^{m+1}}{\partial u_{S^m}^m} \\ \frac{\partial u_2^{m+1}}{\partial u_1^m} & \frac{\partial u_2^{m+1}}{\partial u_2^m} & \dots & \frac{\partial u_2^{m+1}}{\partial u_{S^m}^m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial u_{S^{m+1}}^{m+1}}{\partial u_1^m} & \frac{\partial u_{S^{m+1}}^{m+1}}{\partial u_2^m} & \dots & \frac{\partial u_{S^{m+1}}^{m+1}}{\partial u_{S^m}^m} \end{bmatrix}. \quad (48)$$

Next, we want to find an expression for this matrix. Consider the element i,j of this matrix

$$\frac{\partial u_i^{m+1}}{\partial u_j^m} = w_{ij}^{m+1} \frac{\partial y_j^m}{\partial u_j^m} = w_{ij}^{m+1} \frac{\partial f^m(u_j^m)}{\partial u_j^m} = w_{ij}^{m+1} \dot{f}^m(u_j^m), \quad (49)$$

where

$$\dot{f}^m(u_j^m) = \frac{\partial f^m(u_j^m)}{\partial u_j^m}. \quad (50)$$

Therefore, the Jacobian matrix can be written as

$$\frac{\partial \mathbf{u}^{m+1}}{\partial \mathbf{u}^m} = \mathbf{W}^{m+1} \dot{\mathbf{F}}^m(\mathbf{u}^m), \quad (51)$$

where

$$\dot{\mathbf{F}}^m(\mathbf{u}^m) = \begin{bmatrix} \dot{f}^m(u_1^m) & 0 & \cdots & 0 \\ 0 & \dot{f}^m(u_2^m) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \dot{f}^m(u_{s^m}^m) \end{bmatrix}. \quad (52)$$

It is now possible to write the recurrence relationship for the sensitivity using the chain rule in matrix form

$$\begin{aligned} \boldsymbol{\delta}^m &= \frac{\partial \mathfrak{S}}{\partial \mathbf{u}^m} = \left(\frac{\partial \mathbf{u}^{m+1}}{\partial \mathbf{u}^m} \right)^T \frac{\partial \mathfrak{S}}{\partial \mathbf{u}^{m+1}} = \dot{\mathbf{F}}^m(\mathbf{u}^m) (\mathbf{W}^{m+1})^T \frac{\partial \mathfrak{S}}{\partial \mathbf{u}^{m+1}} \\ &= \dot{\mathbf{F}}^m(\mathbf{u}^m) (\mathbf{W}^{m+1})^T \boldsymbol{\delta}^{m+1}. \end{aligned} \quad (53)$$

Note that the sensitivities are (back)propagated from the last to the first layer

$$\boldsymbol{\delta}^M \rightarrow \boldsymbol{\delta}^{M-1} \rightarrow \dots \rightarrow \boldsymbol{\delta}^2 \rightarrow \boldsymbol{\delta}^1 \quad (54)$$

There is a last step to be executed to complete the backpropagation algorithm. We need the starting point, $\boldsymbol{\delta}^M$, for the recurrence relation of Equation (53).

$$\delta_i^M = \frac{\partial \mathfrak{S}}{\partial u_i^M} = \frac{\partial (\mathbf{d} - \mathbf{y})^T (\mathbf{d} - \mathbf{y})}{\partial u_i^M} = \frac{\partial \sum_{j=1}^{s^M} (d_j - y_j)^2}{\partial u_i^M} = -2(d_i - y_i) \frac{\partial y_i}{\partial u_i^M}. \quad (55)$$

Now, since

$$\frac{\partial y_i}{\partial u_i^M} = \frac{\partial y_i^M}{\partial u_i^M} = \frac{\partial f^M(u_j^M)}{\partial u_i^M} = \dot{f}^M(u_j^M). \quad (56)$$

It is possible to write

$$\delta_i^M = -2(d_i - y_i) \dot{f}^M(u_j^M). \quad (57)$$

Or equivalently in matrix notation

$$\boldsymbol{\delta}^M = -2\dot{\mathbf{F}}^M(\mathbf{u}^M)(\mathbf{d} - \mathbf{y}). \quad (58)$$

Figure 28 provides an adaptation, using matrix notation, of the result presented by Narendra and Parthasarathy (1990) and describes a flow graph for the backpropagation algorithm.

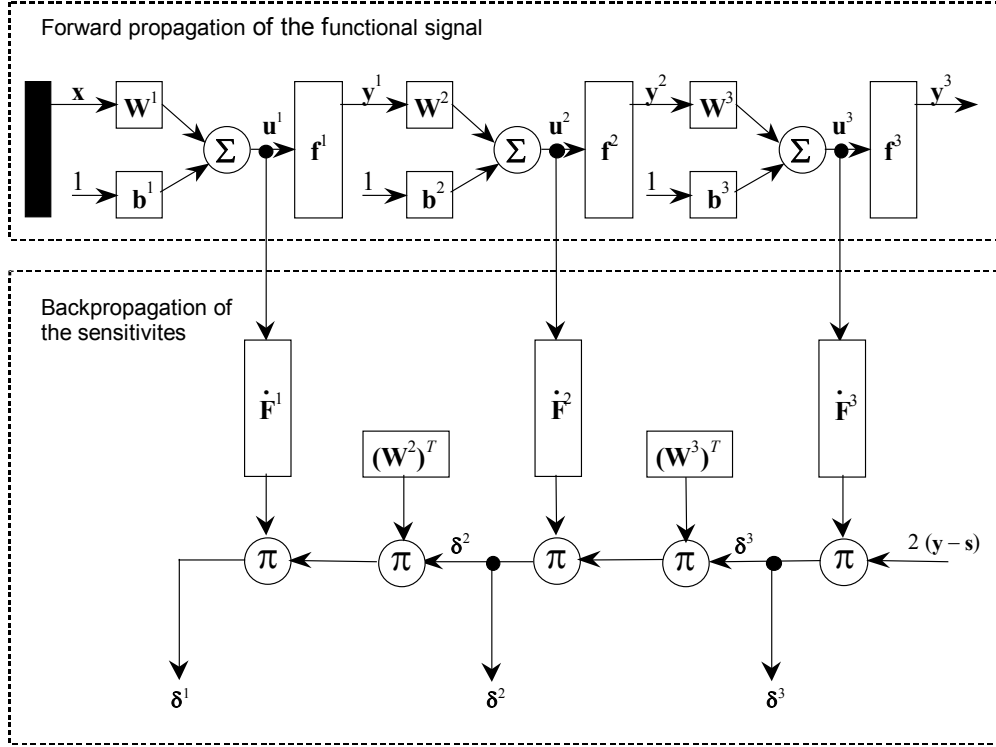


Figure 28: Architectural graph of an MLP network with three layers representing the forward propagation of the functional signals and the backward propagation of the sensitivities.

Algo 4 presents the pseudocode for the backpropagation algorithm. The value of α should be small, $0 < \alpha < 1$.

```

procedure [W] = backprop(max_it,min_err, $\alpha$ ,X,D)
  for m from 1 to M do,
    initialize  $\mathbf{W}^m$            //small random values
    initialize  $\mathbf{b}^m$            //small random values
  end for
  t  $\leftarrow$  1
  while t < max_it & MSE > min_err do,
    for i from 1 to N do,      //for each training pattern
      //forward propagation of the functional signal
       $\mathbf{y}^0 \leftarrow \mathbf{x}_i$                                      //Equation (30)
       $\mathbf{y}_i^{m+1} \leftarrow \mathbf{f}^{m+1}(\mathbf{W}^{m+1}\mathbf{y}_i^m + \mathbf{b}^{m+1})$ ,  $m = 0, 1, \dots, M-1$ , //Equation (29)
      //backpropagation of sensitivities
       $\delta_i^M \leftarrow -2\dot{\mathbf{F}}^M(\mathbf{u}_i^M)(\mathbf{d}_i - \mathbf{y}_i)$            //Equation (58)
       $\delta_i^m \leftarrow \dot{\mathbf{F}}^m(\mathbf{u}_i^m)(\mathbf{W}^{m+1})^T \delta_i^{m+1}$ ,  $m = M-1, \dots, 2, 1$ , //Equation (53)
      //update weights and biases
    end for
    t  $\leftarrow$  t + 1
  end while

```

```

 $\mathbf{W}^m \leftarrow \mathbf{W}^m - \alpha \delta_i^m (\mathbf{y}_i^{m-1})^T, m = 1, \dots, M,$  //Equation (45)
 $\mathbf{b}^m \leftarrow \mathbf{b}^m - \alpha \delta_i^m, m = 1, \dots, M,$  //Equation (46)
//calculate the error for pattern  $i$ 
 $E_i \leftarrow \mathbf{e}_i^T \mathbf{e}_i = (\mathbf{d}_i - \mathbf{y}_i)^T (\mathbf{d}_i - \mathbf{y}_i)$ 
end for
MSE  $\leftarrow 1/N \cdot \text{sum}(E_i)$  //MSE
t  $\leftarrow t + 1$ 
end while
end procedure

```

Algo 4: Learning algorithm for the MLP network trained via the backpropagation algorithm.

4.4.2. Universal Function Approximation

An MLP network can be seen as a generic tool to perform a *non-linear input-output mapping*. More specifically, let m be the number of inputs to the network and o the number of outputs. The input-output relationship of the network defines a mapping from an input m -dimensional Euclidean space into an output o -dimensional Euclidean space, which is infinitely continuously differentiable.

Cybenko (1989) was the first researcher to rigorously demonstrate that an MLP neural network with a single hidden layer is sufficient to uniformly approximate any continuous function that fits a unit hypercube.

The *universal function approximation theorem* is as follows:

Theorem: Let $f(\cdot)$ be a non-constant continuous, limited, and monotonically increasing function. Let \mathbf{I}_m be a unit hypercube of dimension m , $(0,1)^m$. The space of continuous functions in \mathbf{I}_m is denoted by $\mathbf{C}(\mathbf{I}_m)$. Thus, given any function $g \in \mathbf{C}(\mathbf{I}_m)$ and $\varepsilon > 0$, there is an integer M and sets of real-valued constants α_i and w_{ij} , where $i = 1, \dots, o$ and $j = 1, \dots, m$, it is possible to define

$$F(x_1, x_2, \dots, x_m) = \sum_{i=1}^o \alpha_i f\left(\sum_{j=1}^m w_{ij} x_j - w_{i0}\right). \quad (59)$$

as an approximation to the function $g(\cdot)$ such that

$$|F(x_1, x_2, \dots, x_m) - g(x_1, x_2, \dots, x_m)| < \varepsilon$$

For all $\{x_1, \dots, x_m\} \in \mathbf{I}_m$.

Proof: see Cybenko (1989).

This theorem is directly applicable to the MLP networks. First, note that the sigmoidal function used in the MLP networks is continuous, non-constant, limited, and monotonically increasing; satisfying the constraints imposed to $f(\cdot)$. Then, note that Equation (59) represents the outputs of an MLP network as illustrated in Figure 29.

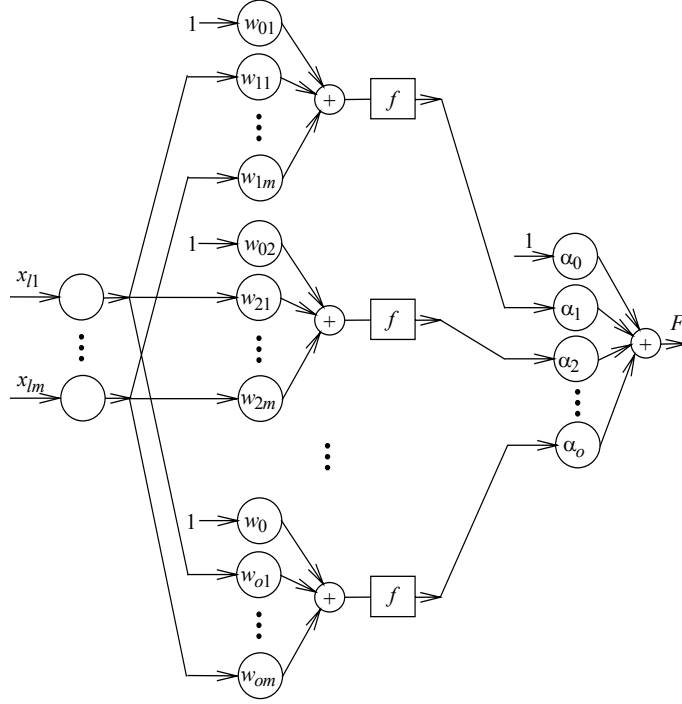


Figure 29: An MLP network as a universal function approximator. The network parameters compose Equation (59). (Courtesy of © Fernando Von Zuben)

In summary, the theorem states that an MLP network with a single hidden layer is capable of uniform approximation, given an appropriate training data set to represent the function. However, the theorem does not say anything regarding the number of units in the hidden layer necessary to perform an approximation with the precision ε .

4.4.3. Network Structure, Generalization, Epoch, and Convergence

There are a few aspects of the MLP network training that deserve some comments.

- *Network architecture:* the number of network inputs is usually given by the training data, while the number of outputs and hidden units are design issues. For instance, if a data set to be classified has a single class, one or two output units can be used. In the case of a single sigmoidal output in the range $(-1,1)$, a network output less than '0' can be considered as not belonging to the class, and an output greater than or equal to zero can be considered as belonging to the class. If two output units are used, then one of them corresponds to a membership of the class, while the other corresponds to not belonging to the class. Finally, the number of hidden units is chosen so that the network can appropriately classify or approximate the training set. By increasing the number of hidden units, one increases the network mapping capability.
- *Generalization:* we have already discussed that too much training may result in overfitting, while too little training may result in underfitting. Overfitting is only possible if there are enough hidden units to promote an increasingly better repre-

sensation of the data set. On the other side, if the network is not big enough to perform an adequate input-output mapping, underfitting may occur.

- *Convergence*: the MLP network is usually trained until a minimum value for the error is achieved; for instance, stop training if the $MSE < 0.001$. Another choice for the *stopping criterion* is to stop training if the estimated gradient vector at iteration t has a small norm; for instance, stop training if $\|\partial\mathfrak{J}/\partial w\| < 0.001$. A small value for the norm of the gradient value indicates that the network is close to a local minimum.
- *Epoch*: after all the training patterns have been presented to the network and the weights are adjusted, an *epoch* is said to have occurred. This terminology is used in all artificial neural network learning algorithms, including the perceptron and self-organizing maps.
- *Data normalization*: when the activation functions of the neurons have a well-defined range, such as all sigmoidal functions, it is important to normalize the data such that they belong to a range equal to or smaller than the range of the activation functions. This helps in avoiding the saturation of the neurons in the network.
- *Initialization of the weight vectors and biases*: most network models, mainly multi-layer feedforward networks, are sensitive to the initial values chosen for the network weights. This sensitivity can be expressed in several ways, such as guarantee of convergence, convergence speed, and convergence to local optima solutions.

4.4.4. Biological Plausibility of Backpropagation

The backpropagation of error signals is probably the most problematic feature in biological terms. Despite the apparent simplicity and elegance of the backpropagation algorithm, it seems quite implausible that something like the equations described above are computed in the brain (Anderson, 1995; O'Reilly & Munakata, 2000; Trappenberg, 2002).

In order to accomplish it, some form of information exchange between post-synaptic and pre-synaptic neurons should be possible. For instance, there would be the requirement, in biological terms, that the sensitivity values were propagated backwards from the dendrite of the post-synaptic neuron, across the synapse, into the axon terminal of the pre-synaptic neuron, down the axon of this neuron, and then integrated and multiplied by both the strength of that synapse and some kind of derivative, and then propagated back out its dendrites, and so on. If a neuron were to gather the backpropagated errors from all the other nodes to which it projects, some synchronization issues would arise, and it would also affect the true parallel processing in the brain.

However, it is possible to rewrite the backpropagation equations so that the error backpropagation between neurons takes place using standard activation signals. This approach has the advantage that it ties into the psychological interpretation of the teaching signal (desired output) as an actual state of experience that reflects something like an outcome or corrected response. The result is a very powerful learning algorithm that need not ignore issues of biological plausibility (Hinton & McClelland, 1987; O'Reilly, 1996).

4.4.5. Examples of Application

Two simple applications for the MLP network will be presented. The first problem – universal function approximation – aims at demonstrating that the MLP network with non-linear hidden units is capable of universal approximation, and how this is accomplished. The second example – the XOR problem – is the classic problem requiring hidden units. Also, many other more difficult problems involve an XOR as a subproblem.

Universal Function Approximation

To illustrate the universal function approximation of the MLP neural networks, consider the following example: approximate one period of the function $\sin(x)\cos(2x)$ using an additive composition of basic functions ($f(\cdot)$ is the hyperbolic tangent function) under the form of Equation (59), as illustrated in Figure 30.

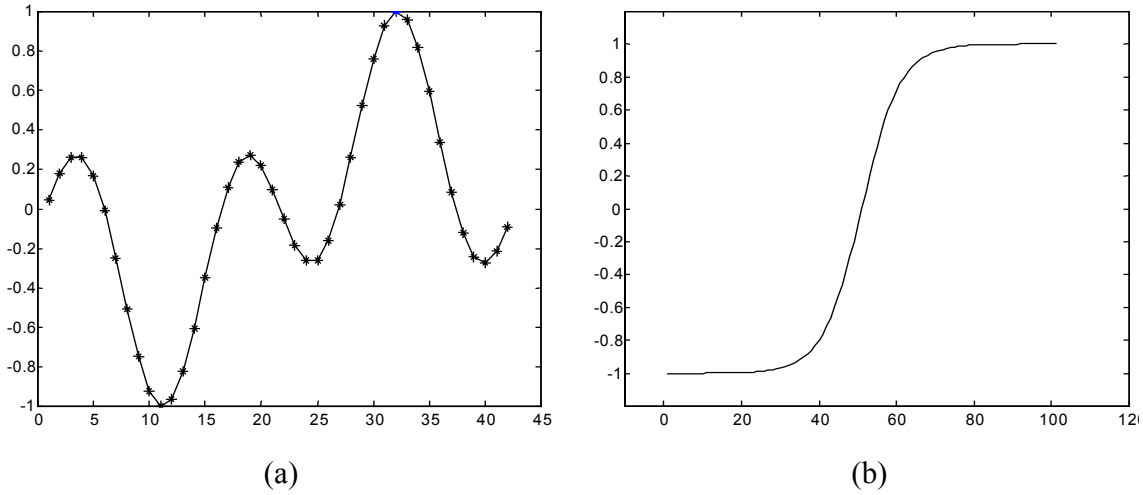


Figure 30: Universal approximation. (a) Function to be approximated (information available: 42 training data equally spaced in the domain of the function). (b) Basic function to be used in the approximation: hyperbolic tangent, satisfying the constraints of the universal function approximation theorem.

The MLP network depicted in Figure 31 is used to approximate this function. The architecture is: $m = 1$ input, $S^1 = 5$ hidden units in a single hidden layer, and $o = 1$ output unit. The training algorithm is the backpropagation introduced above. By looking at this picture, and assuming the network has a linear output, y is expressed as

$$y = w_{10} + w_{11}z_1 + w_{12}z_2 + w_{13}z_3 + w_{14}z_4 + w_{15}z_5, \quad (60)$$

where

$$z_l = f\left(\sum_{j=1}^m v_{lj}x_j + v_{l0}\right), \quad l = 1, \dots, 5, \quad (61)$$

where $f(\cdot)$ is the hyperbolic tangent, m is the number of network inputs, v_{lj} is the weight of the synapse connecting input j to the hidden unit l . (Compare Equations (60) and (61) with Equation (59).)

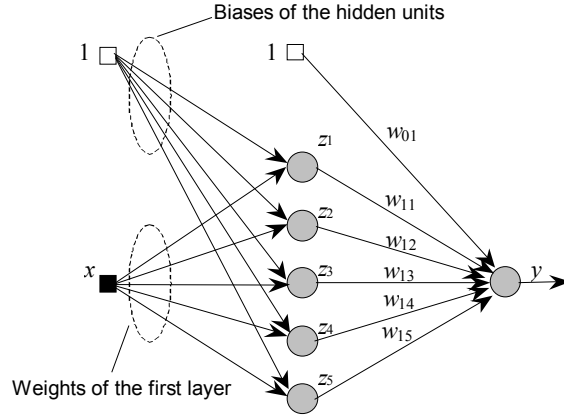


Figure 31: MLP network trained via the error backpropagation algorithm used to approximate the function $\sin(x).\cos(2x)$. The activation functions are hyperbolic tangents.

After training the network so as to define appropriate weight sets, let us analyze the components of Equation (60) that determine the network output. Figure 32 depicts the activation functions (basic functions) of the hidden units after network training. Note that these curves are hyperbolic tangents. Figure 33 presents the linear combination of the outputs of the hidden units, where the coefficients of the combination are the weights of the second layer in the network. Finally, the resultant approximation is presented in Figure 34.

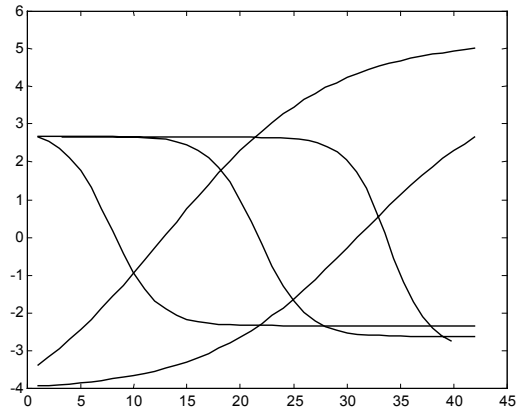


Figure 32: Activation functions of the hidden units, after the network training, multiplied by the corresponding weights of the output layer. Note that all functions are hyperbolic tangents (basic functions).

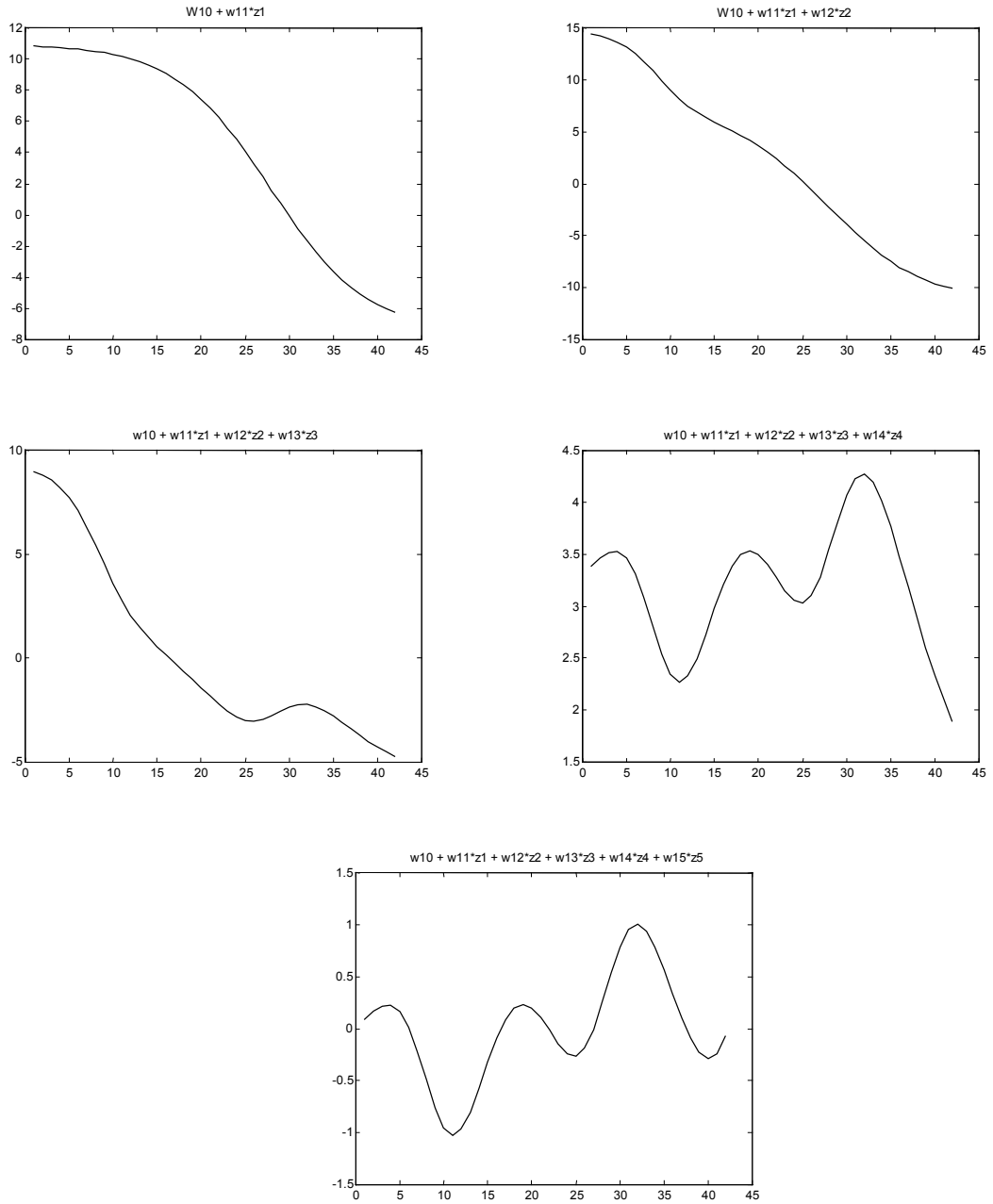


Figure 33: Linear combination of the outputs of the hidden units. The coefficients of the linear combination are the weights of the second layer. This figure presents the sum of the curves that compose Figure 32, corresponding to the additive composition (linear combination) of the basic functions of the network's neurons.

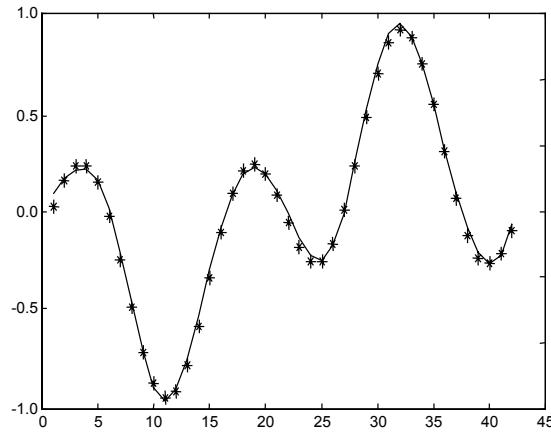


Figure 34: Approximation obtained by the MLP network for the function $\sin(x).\cos(2x)$. The * correspond to the training samples, and the line (—) is the network approximation.

The XOR Problem

This is the classic problem requiring hidden units in an MLP network in order to appropriately classify the data set. It is known that an MLP network with only two hidden units is capable of solving it. The application of the backpropagation algorithm to solve this problem is straightforward, and left as an exercise to the reader.

4.5. Self-Organizing Maps

The *self-organizing map* (SOM) discussed in this section is an unsupervised (self-organized or competitive – Section 3.3.2) network introduced by T. Kohonen (1982). The network architecture is a single layer feedforward network with the post-synaptic neurons placed at the nodes of a *lattice* or *grid* that is usually uni- or bi-dimensional, as illustrated Figure 35. This network is trained via a competitive learning scheme, thus there are lateral connections between the output units which are not shown in the picture.

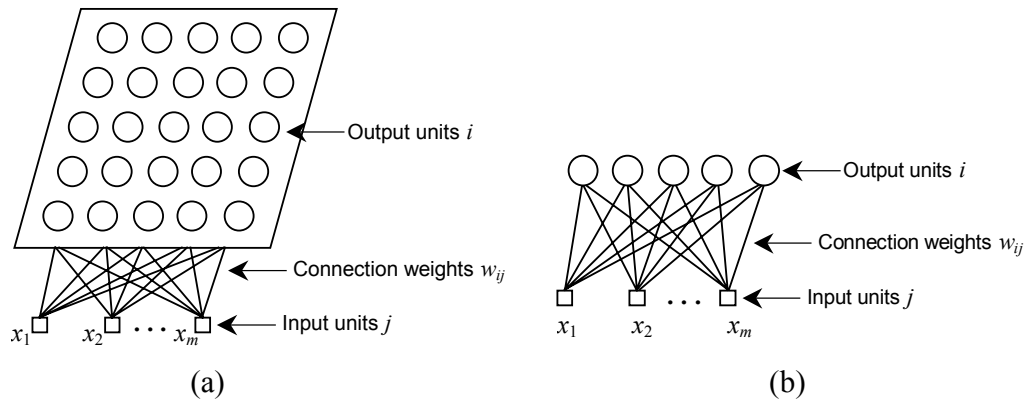


Figure 35: Typical architectures (single layer feedforward) of a self-organized map. (a) Bi-dimensional output grid. (b) Uni-dimensional output grid.

The output units become selectively tuned to various input signal patterns or classes of patterns through an unsupervised learning process. In the standard version to be presented here, only one neuron or local group of neurons at a time gives the active response to the current input. The locations of the responses tend to become ordered as if some meaningful coordinate system for different input features were being created over the network (Kohonen, 1990). A self-organizing map is thus characterized by the formation of a topographic map of the input patterns in which the spatial location or coordinates of a neuron in the network are indicative of intrinsic statistical features contained in the input patterns.

4.5.1. Self-Organizing Map Learning Algorithm

The main objective of the self-organizing map is to transform an input pattern of dimension m into a uni- or bi-dimensional discrete map, and to perform this transformation adaptively in a topologically ordered form. Each unit in the grid presented in Figure 35 is fully connected to all the pre-synaptic units in the input layer. The uni-dimensional grid of Figure 35(b) is a particular case of the bi-dimensional grid of Figure 35(a).

The self-organizing learning algorithm proceeds by first initializing the synaptic strengths of the neurons in the network, what usually follows the same rule of the other networks discussed. That is, by assigning them small values picked from a random or uniform distribution. This stage is important so that no a priori order is imposed into the map. After initialization, there are three essential processes involved in the formation of the self-organized map (Haykin, 1999): *competition*, *cooperation*, and *adaptation*.

Let $\mathbf{x} = [x_1, x_2, \dots, x_m]^T \in \mathfrak{R}^m$ be an input vector (training pattern) that is assumed to be presented in parallel to all the neurons i in the network. The whole data set \mathbf{X} is composed on N patterns, $\mathbf{X} \in \mathfrak{R}^{m \times N}$. The weight vector of unit i is $\mathbf{w}_i = [w_{i1}, w_{i2}, \dots, w_{im}]^T \in \mathfrak{R}^m$, and thus the weight matrix of the network is $\mathbf{W} \in \mathfrak{R}^{o \times m}$, where o is the number of output (post-synaptic) units in the network.

Competition: For each input pattern the o neurons in the network will compete to become activated. One simple way of performing the competition among the output units is by applying the competitive learning scheme discussed in Section 3.3.2. That is, for a neuron i to be the winner, the distance between its corresponding weight vector \mathbf{w}_i and a certain input pattern \mathbf{x} must be the smallest measure (among all the network output units), given a certain metric $\|\cdot\|$ usually taken to be the Euclidean distance. Let $i(\mathbf{x})$ indicate the winner neuron for input pattern \mathbf{x} . It can thus be determined by

$$i(\mathbf{x}) = \arg \min_j \|\mathbf{x} - \mathbf{w}_j\|, \quad j = 1, \dots, o \quad (62)$$

Neuron $i(\mathbf{x})$ that satisfies Equation (62) is known as the best-matching, winner, or winning unit for the input pattern \mathbf{x} .

Cooperation: The winning neuron locates the center of a topological neighborhood of cooperating neurons. There are neurobiological evidences for the lateral interaction among a set of excited neurons. In particular, a neuron that is firing tends to excite the neurons in its immediate neighborhood more than those far away from it. One form of simulating this is by defining topological neighborhoods around a winning neuron, in which a similar kind of adaptation to that of the winner will be imposed.

In biophysically inspired neural networks, there are a number of ways of defining the topological neighborhood by using various kinds of lateral feedback connections and other lateral interactions. In the present discussion, lateral interaction is enforced directly by defining a neighborhood set Nu around a winning unit u . At each iteration, all the units within Nu are updated, whereas units outside Nu are left intact. The width or radius of Nu is usually time variant; Nu is initialized very wide in the beginning of the learning process and shrinks monotonically with time. Figure 36 illustrates some typical topological neighborhoods for bi-dimensional and uni-dimensional output grids. In Figure 36(a), two different forms of implementing a squared neighborhood for a bi-dimensional grid are depicted. Note that other neighborhoods are still possible, such as an hexagonal (or circular) neighborhood.

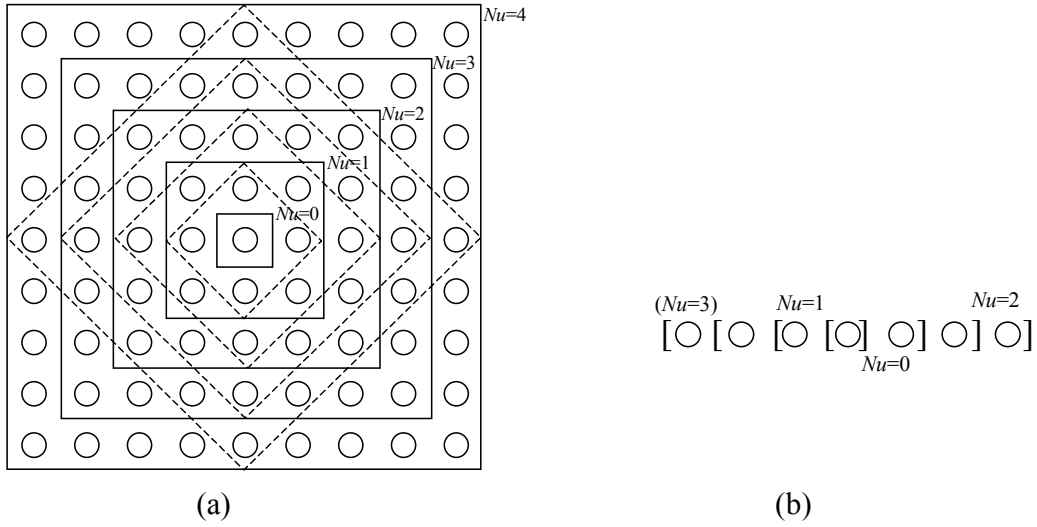


Figure 36: Examples of topological neighborhoods for bi-dimensional (a) and uni-dimensional grids (b).

Adaptation: The updating process for the weight vectors is similar to the one presented in Equation (17) for competitive learning schemes, but takes into account the topological neighborhood of the winning unit.

$$\mathbf{w}_i(t+1) = \begin{cases} \mathbf{w}_i(t) + \alpha(t)[\mathbf{x}(t) - \mathbf{w}_i(t)] & \text{if } i \in Nu \\ \mathbf{w}_i(t) & \text{if } i \notin Nu \end{cases}, \quad (63)$$

where the learning rate $\alpha(t)$ decreases with time, $0 < \alpha(t) < 1$.

It is sometimes useful to keep the initial values for the parameters α and Nu fixed for a number of iterations, usually 1000, before starting to cool (reduce) them. Following this approach may result in an initial period corresponding to a self-organizing or *ordering phase* of the map, followed by a *convergence phase* in which, as the values of α and Nu are monotonically reduced, the network will later suffer very little adaptation.

```

procedure [W] = som(max_it,  $\alpha$ , X)
  initialize W //small random values ( $w_i \neq w_j$  for  $i \neq j$ )
  t  $\leftarrow$  1
  while t < max_it do,
    vet_permut  $\leftarrow$  randperm(N) //vector with permutations of N
    for j from 1 to N do, //for all input patterns
      //select the index  $i$  of a pattern  $\mathbf{x}_i$  to be presented to the SOM
       $i \leftarrow$  vet_permut(j) //present patterns randomly
      //perform the competition and determine the winner unit  $i(\mathbf{x})$ 
       $i(\mathbf{x}_i) \leftarrow \arg \min_j \|\mathbf{x}_i - \mathbf{w}_j(t)\|, j = 1, \dots, O$ 
      //cooperation and adaptation
       $\mathbf{w}_i(t+1) = \begin{cases} \mathbf{w}_i(t) + \alpha(t)[\mathbf{x}_i - \mathbf{w}_i(t)] & \text{if } i \in Nu \\ \mathbf{w}_i(t) & \text{if } i \notin Nu \end{cases}$ 
      //update the learning rate and the neighborhood radius
       $\alpha \leftarrow \alpha/t$  //until  $\alpha$  reaches a minimal value
       $Nu \leftarrow Nu - 1$  //until  $Nu = 1$ 
    end for
    t  $\leftarrow$  t + 1
  end while
end procedure

```

Algo 5: Learning algorithm for the standard Kohonen self-organized map (SOM).

4.5.2. Biological Basis and Inspiration of the Self-Organizing Map

It has been long since a topographical organization of the brain, in particular of the cerebral cortex, has been suggested. These could be deduced from functional deficits and behavioral impairments produced by various kinds of lesions, hemorrhages, tumors, and malformations. Different regions in the brain thereby seemed to be dedicated to specific tasks (see Figure 5).

Nowadays, more advanced techniques, such as *positron emission topography* (PET), *magneto encephalography* (MEG), and *electroencephalogram* (EEG), can be used to measure nervous activity, thus allowing the study of the behavior of living brains. After a large number of experimentation and observation, a fairly detailed organization view of the brain has been proposed. Especially in higher animals, such as the human beings, the various cortices in the cell mass seem to contain many kinds of topographic maps, such that a particular location of the neural response in the map often directly corresponds to a specific modality and quality of sensory signal.

The development of the self-organizing maps as neural models is motivated by this distinctive feature of local brain organization into topologically ordered maps. The computational map constitutes a basic building block in the information-processing infrastructure of the nervous system. A computational map is defined by an array of neurons representing slightly different tuned processors or filters, which operate on the sensory information-bearing signals in parallel (Haykin, 1999). The use of these computational maps offers some interesting properties:

- At each stage of representation, each incoming piece of information is kept in its proper context;
- Neurons dealing with closely related pieces of information are close together so they can interact via short synaptic connections.

The self-organizing map, or perhaps more accurately named *artificial topographic map*, was designed to be a system capable of learning through self-organization in a neurobi-

ologically inspired manner. Under this perspective, the most relevant aspect of the nervous system to be taken into account is its capability of topographical map formation. As stated by T. Kohonen: “The spatial location or coordinates of a cell in the network then correspond to a particular domain of input signal patterns.” (Kohonen, 1990; p. 1464)

4.5.3. Examples of Applications

To illustrate the behavior and applicability of the SOM, it will first be applied to a structured random distribution of data. This will demonstrate the self-organizing properties of the network. The second problem – animals data set – demonstrates the SOM capability of identifying classes of data, and allowing for the visualization of useful relationships in a high-dimensional data set.

Structured Random Distribution

To illustrate the behavior of the SOM, consider an input data set composed of 500 bi-dimensional samples uniformly distributed over the interval $[-1,1]$, as depicted in Figure 37(a). The input patterns were normalized to the unit interval so that their distribution lie on the surface of a unit circle. A bi-dimensional grid with 6 rows and 6 columns (6×6) is used. Figure 37(b) to (d) shows three stages of the adaptive process as the network learns to represent the input distribution.

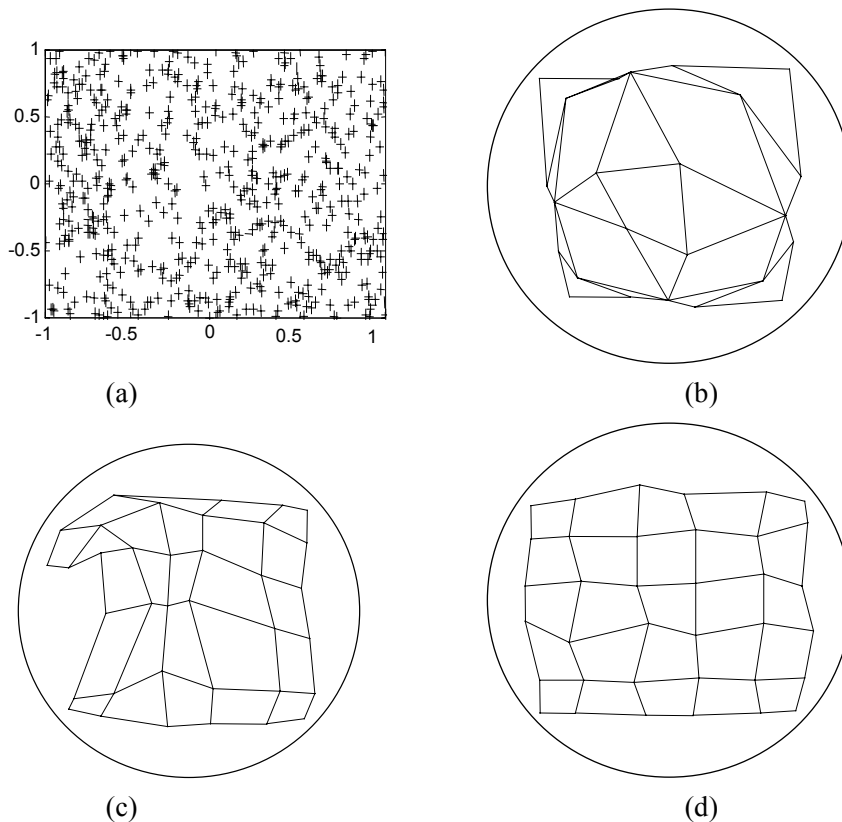


Figure 37: Application of the SOM network with a bi-dimensional grid to a randomly distributed input data set. (a) Input data set for a bi-dimensional SOM. (b) Condition of the grid after 1000 iterations. (c) Condition of the grid after 5000 iterations. (d) Grid after 10000 iterations.

Semantic Maps

This section illustrates the SOM behavior by applying it to the Animals data set. This high-dimensional data set was originally proposed by Ritter and Kohonen (1989) to verify the SOM capability of creating a topographic map based on a symbolic data set. The data set is composed of 16 input vectors each representing an animal with the binary feature attributes as shown in Table 1. A value of ‘1’ in this table corresponds to the presence of an attribute, whilst a value of ‘0’ corresponds to the lack of this attribute. The authors suggested that the interestingness of this data set lies in the fact that the relationship between the different symbols may not be directly detectable from their encodings, thus not presuming any metric relations even when the symbols represent similar items.

Table 1: Animals data set: animals’ names and their attributes.

		Dove	Hen	Duck	Goose	Owl	Hawk	Eagle	Fox	Dog	Wolf	Cat	Tiger	Lion	Horse	Zebra	Cow
Is	Small	1	1	1	1	1	1	0	0	0	0	1	0	0	0	0	0
	Medium	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0
	Big	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
Has	Two legs	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
	Four legs	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
	Hair	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
	Hooves	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
	Mane	0	0	0	0	0	0	0	0	0	1	0	0	1	1	1	0
	Feathers	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
Likes to	Hunt	0	0	0	0	1	1	1	1	0	1	1	1	1	0	0	0
	Run	0	0	0	0	0	0	0	0	1	1	0	1	1	1	1	0
	Fly	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0
	Swim	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0

Figure 38 illustrates the performance of a SOM with a uni-dimensional output grid applied to the Animals data set. In Figure 38(a) an output grid with 14 units was used and in Figure 38(b) a grid with 8 units was used. Note the interesting capability of the SOM to cluster data which are semantically similar without using any labeled information about the input patterns. By observing Table 1 it is possible to note that the horse and zebra, as well as the owl and the hawk, have the same set of attributes, thus justifying them being classified as belonging to the same cluster. Note also that animals with similar features are mapped onto neighboring neurons.

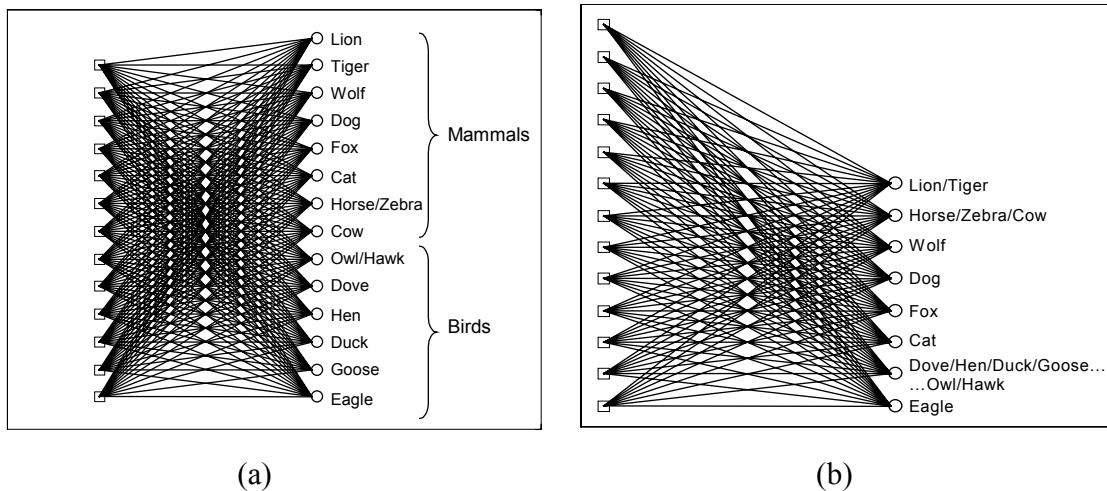


Figure 38: Uni-dimensional feature map containing labeled units with strongest responses to their respective inputs. (a) Output grid with 14 units. (b) Output grid with 8 units.

4.6. Discrete Hopfield Network

J. Hopfield (1982, 1984) realized that in physical systems composed of a large number of simple elements, interactions among large numbers of elementary components yield complex collective phenomena, such as the stable magnetic orientations and domains in a magnetic system or the vortex patterns in fluid flow. Particularly, Hopfield noticed that there are classes of physical systems whose spontaneous behavior can be used as a form of general and error-correcting *content-addressable memory* (CAM). The primary function of a content-addressable memory, also called an *associative memory*, is to retrieve a pattern stored in the memory as a response to the presentation of an incomplete or noisy version of that pattern, as illustrated in Figure 39. In a CAM, the memory is reached not by knowing its address, but rather by supplying some subpart of the memory.

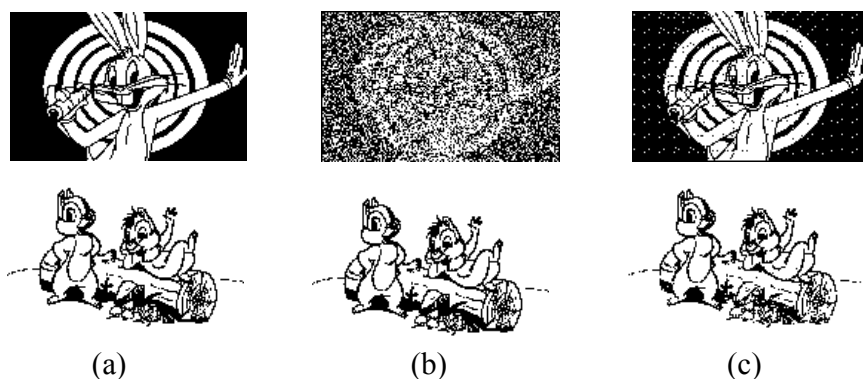


Figure 39: Content-addressable memory (CAM) or associative memory. Given a memorized pattern (a), the role of the CAM is to restore (c) the stored pattern given an incomplete or noisy version of it (b).

By assuming that the time evolution of a physical system can be described by a set of general coordinates, then a point in a space, called state space, represents the instantaneous condition of the system. Therefore, the equations of motion of the system describe a flow in state space. Assuming also systems whose flow is performed towards locally stable points from anywhere within given neighborhoods of these stable points, these systems become particularly useful as content-addressable memories.

The original neuron model used by Hopfield (1982) was that of McCulloch and Pitts (1943), whose output could be either '0' or '1', and the inputs were external and from other neurons as well. Thus, his analysis involved networks with full back-coupling, that is, recurrent networks (such as the one illustrated in Figure 13). Other important features of the Hopfield network were the storage (instead of iterative learning) of the memories of the system, and the use of an asynchronous procedure of memory retrieval, as will be further discussed.

4.6.1. Elementary Theory of Non-Linear Dynamical Systems

Dynamical systems embody all systems whose behavior varies with time, more formally, whose *state* varies with time. A suitable model to study dynamical neural networks is the *state space model*, which takes into account state variables whose values, at any instant of time, are supposed to contain sufficient information to predict the future behavior of the system.

Dynamical systems are composed of two main parts: *state* and *dynamics*. The *state* describes the current condition of the system under the form of a vector parametrized in relation to time. The set of all possible states is termed the *state space* of the system. The *dynamics* describes how the state of the system varies with time, and the sequence of states of a dynamical system is termed its *trajectory* in the state space.

Assume a non-linear dynamical system whose state variables $s_1(t), s_2(t), \dots, s_m(t)$, where t is the continuous time (independent variable) and m is the order of the system, can be concatenated into a vector $\mathbf{s}(t) = [s_1(t), s_2(t), \dots, s_m(t)]^T$ named the state vector of the system. Assume also non-linear dynamical systems whose dynamics can be expressed as a first-order differential equation:

$$\frac{d}{dt}\mathbf{s}(t) = \mathbf{F}(\mathbf{s}(t)) \quad (64)$$

Note that the function $\mathbf{F}(\cdot)$ in Equation (64) is similar to function $\dot{\mathbf{F}}(\cdot)$ in Equation (52) in the sense that both are diagonal; that means:

$$\mathbf{F}(\mathbf{s}) = \begin{bmatrix} f_1(s_1) & 0 & \cdots & 0 \\ 0 & f_2(s_2) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & f_m(s_m) \end{bmatrix}.$$

The system described in Equation (64) is said to be *autonomous* because there is no explicit dependence of $\mathbf{F}(\cdot)$ with relation to time. The state space proposed in Equation (64) can be viewed as describing the motion of a point in an m -dimensional state space. This is important because it provides a visual and conceptual tool to analyze the dynamics of non-

linear systems. At a particular instant of time, the state of the system is represented by a single point in the m -dimensional state space.

A non-linear dynamical system can present a variety of behaviors. Consider an autonomous dynamical system described by the state space Equation (64). A given constant vector \mathbf{s}^* is said to be an *equilibrium* or *stationary state* of the system if $\mathbf{F}(\mathbf{s}^*) = \mathbf{0}$, where $\mathbf{0}$ is the null vector. Therefore, \mathbf{s}^* is a solution to Equation (64).

An *attractor* is a region of the state space to where the trajectories converge given some initial condition. The region of the state space from which the system evolves towards the attractor is termed its *basin of attraction*. There are different types of attractors, for instance, *point attractors* and *limit cycles*. A point attractor is a single equilibrium point to which the trajectory of the system converges, and the limit cycle is a periodic orbit to which the trajectory converges. Therefore, a system initialized in the basin of attraction of a given attractor is going to converge to such attractor. The notions of attractors and basins of attraction are illustrated in Figure 40.

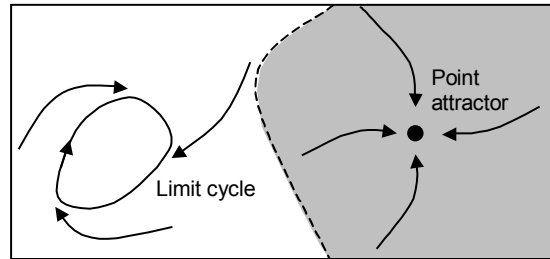


Figure 40: Attractors and basins of attraction. There is an imaginary boundary (dashed line) that separates the basin of attraction of each of the attractors. The left-hand side attractor is a limit cycle, while the right-hand one is a point attractor.

4.6.2. Recurrent Neural Networks as Non-Linear Dynamical Systems

The Hopfield neural networks were proposed by J. Hopfield in the early 1980s (Hopfield, 1982, 1984) as neural models derived from statistical physics. They incorporate a fundamental principle from physics; the storage of information in a dynamically stable configuration. Each pattern is stored in a valley of an energy surface. The non-linear dynamics of the network is such that the energy is minimized and valleys represent points of stable equilibrium, each one with its own basin of attraction. Therefore, a network memory corresponds to a point of stable equilibrium, thus allowing the system to behave as an associative memory.

In this case, convergent flow to stable states becomes the essential feature of the CAM operation. There is a simple mathematical condition that guarantees that the state space flow algorithm converges to stable states. Any symmetric matrix \mathbf{W} with zero-diagonal elements (i.e., $w_{ij} = w_{ji}$, $w_{ii} = 0$) will produce such a flow. The proof of this property follows from the construction of an appropriate energy function that is always decreased by any state change produced by the algorithm.

Assume a single-layer neural network composed of N neurons and a corresponding set of unit delays, forming a multiple-loop feedback system, as illustrated in Figure 13 and reproduced in Figure 41 for convenience.

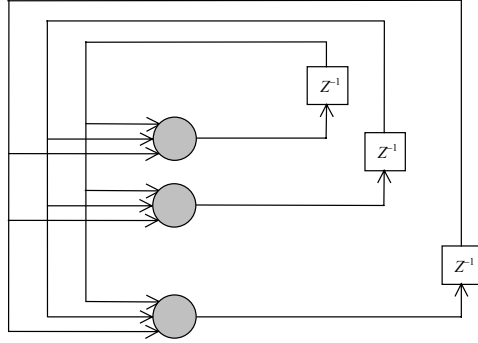


Figure 41: Recurrent neural network with no hidden layer.

Assume that the neurons in this network have symmetric coupling described by $w_{ij} = w_{ji}$ ($i, j = 1, \dots, N$), where w_{ij} is the synaptic strength connecting the output of unit i to the input of unit j . Assume also that no self-feedback is allowed, i.e., $w_{ii} = 0$ ($i = 1, \dots, N$). Let $u_j(t)$ be the net input to neuron j and $y_j(t)$ its corresponding output $y_j(t) = f(u_j(t))$, where $f(\cdot)$ is the activation function of unit j .

Let now the vector $\mathbf{u}(t)$ be the state of the network at time t . Then, from the basic integrate and fire model of a neuron given by Equation (1) it is possible to define an additive model of a neuron as given by Equation (65).

$$\tau_m \frac{du(t)}{dt} = u_{res} - u(t) + R_m i(t), \quad (1)$$

where τ_m is the membrane time constant of the neuron determined by the average conductance of the channels (among other things); u_{res} is the resting potential of the neuron; $i(t)$ is the input current given by the sum of the synaptic currents generated by firings of pre-synaptic neurons; and R_m is the resistance of the neuron to the flow of current (ions).

$$C_j \frac{du_j}{dt} = I_j - \frac{u_j}{R_j} + \sum_{\substack{i=1 \\ i \neq j}}^N w_{ji} f_i(u_i), \quad j = 1, \dots, N. \quad (65)$$

where R_j and C_j are the leakage resistance and capacitance of the neuron respectively, I_j corresponds to an externally applied bias, u_j is the net input to unit j , and $f_i(\cdot)$ is the activation function of unit i .

To the dynamics provided by Equation (65) it is possible to derive an energy function as given by Equation (66) (see Hopfield, 1984).

$$E = -\frac{1}{2} \sum_{i=1}^N \sum_{\substack{j=1 \\ j \neq i}}^N w_{ji} y_i y_j + \sum_{j=1}^N \frac{1}{R_j} \int_0^{y_j} f_j^{-1}(z) dz + \sum_{j=1}^N b_j y_j \quad (66)$$

This energy function is a monotonically decreasing function of the state of the network. Therefore, the stationary states of this network are such that they correspond to equilibrium points (Hopfield, 1984). These equilibrium points are termed attractors because there is a neighborhood (basin of attraction) over which these points exert a dominant influence.

4.6.3. Discrete Hopfield Network

The discrete Hopfield networks constitute a particular case of recurrent networks for which the state space is discrete and the neuron models are the ones proposed by McCulloch and Pitts (Section 3.1.2). This allows it to be seen as a content-addressable memory or non-linear associative memory. In this case, the equilibrium points of the network are known to correspond to the patterns to be memorized. However, the network weights that produce the desired equilibrium points are not known, and thus must be determined according to some rule.

Suppose the goal is to memorize a set \mathbf{X} of m -dimensional vectors (bipolar strings built from the set $\{-1, 1\}$), denoted by $\{\mathbf{x}_i \mid i = 1, \dots, N\}$ in a network with m neurons. These vectors are called the *fundamental memories* of the system and represent the patterns to be memorized. According to the generalized Hebb's rule, the weight matrix $\mathbf{W} \in \mathbb{R}^{m \times m}$ of the network is determined by, in vector notation:

$$\mathbf{W} = \sum_{i=1}^N \mathbf{x}_i (\mathbf{x}_i)^T \quad (67)$$

As discussed previously, to guarantee the convergence of the algorithm to stable states, it is necessary to set $w_{ii} = 0$. Therefore, Equation (67) becomes

$$\mathbf{W} = \sum_{i=1}^N \mathbf{x}_i (\mathbf{x}_i)^T - N\mathbf{I} \quad (68)$$

where \mathbf{I} is the identity matrix.

Equation (68) satisfies all the conditions required for the convergence of this discrete network to stable states: it is symmetric and with zero-valued diagonal. A monotonically decreasing energy function can be defined (see Hopfield 1982).

After storing the memories in the network (*storage phase*), its retrieval capability can now be assessed by presenting new or noisy patterns to the network (*retrieval phase*). Assume an m -dimensional vector $\mathbf{z} \in \mathbb{R}^{m \times 1}$ is taken and presented to the network as its initial state. The vector \mathbf{z} , called *probe*, can be an incomplete or noisy version of one of the fundamental memories of the network.

The memory retrieval process obeys a dynamical rule termed *asynchronous updating*. Randomly select a neuron j of the network

$$s_j(0) = z_j, \quad j = 1, \dots, m,$$

where $s_j(0)$ is the state of neuron j at iteration $t = 0$, and z_j is the j -th element of vector \mathbf{z} . At each iteration, update one randomly selected element of the state vector \mathbf{s} according to the rule:

$$s_j(t+1) = \text{sign} \left[\sum_{i=1}^m w_{ji} s_i(t) \right], \quad j = 1, \dots, m, \quad (69)$$

where the function $\text{sign}[u_j]$ returns 1 if $u_j > 0$, -1 if $u_j < 0$, and remains unchanged if $u_j = 0$.

Repeat the iteration until the state vector \mathbf{s} remains unchanged. The fixed point, stable state, computed $\mathbf{s}_{\text{fixed}}$ corresponds to the output vector returned by the network: $\mathbf{y} = \mathbf{s}_{\text{fixed}}$.

Let $\mathbf{X} \in \mathcal{R}^{m \times N}$ be the matrix of N input patterns of dimension m each (the patterns are placed columnwise in matrix \mathbf{X}), and $\mathbf{Z} \in \mathcal{R}^{m \times n}$ be the matrix of n probe patterns of dimension m each. Algo 6 presents the pseudocode for the discrete Hopfield network, where $\mathbf{Y} \in \mathcal{R}^{m \times N}$ is the matrix of restored patterns.

```

procedure [ $\mathbf{W}, \mathbf{Y}$ ] = hopfield( $\mathbf{X}, \mathbf{Z}$ )
    //storage phase
     $\mathbf{W} \leftarrow \mathbf{X} \mathbf{X}^T$ 
    //retrieval phase
    for  $i$  from 1 to  $n$  do,
        while  $\mathbf{y}_i$  is changing do,
             $\mathbf{y}_i \leftarrow \text{sign}(\mathbf{W} \cdot \mathbf{z}_i)$ 
        end while
    end for
end procedure

```

Algo 6: Discrete Hopfield network.

4.6.4. Spurious Attractors

If the Hopfield network stores N fundamental memories using the generalized Hebb's rule, the stable states present in the energy surface are not restricted to the fundamental memories stored. Any stable state that is not associated with the fundamental memories are named *spurious attractors*. Spurious attractors are due to several factors. The energy function is symmetrical; any linear combination of an odd number of stable states is also a stable state; and for a large number of fundamental memories the energy function generates stable states that are not correlated with any of the fundamental memories.

4.6.5. Example of Application

To illustrate the error-correcting capability of the Hopfield network, consider the same character recognition problem discussed in Section 4.2.4 and illustrated in Figure 42. In the present case, the patterns are assumed to be bipolar, that is, composed of a bit -1 (dark gray) or $+1$ (light gray). As each pattern has a dimension $m = 120$, the network used in the experiment has $m = 120$ neurons, and therefore $m^2 - m = 12,280$ connections.



Figure 42: Fundamental memories for the discrete Hopfield network.

To test the storage capability of the network, the same patterns (fundamental memories) can be presented to the network. To demonstrate its error-correcting capability, introduce random noise in the patterns by independently reversing each pixel of the pattern from +1 to -1, or vice-versa, with a given probability. The corrupted patterns are used as probes to the network.

5. From Natural to Artificial Neural Networks

The study of any mathematical model involves the mapping of the mathematical domain into some observable part of the natural world. The nervous system studied here includes neurons, synaptic strengths, neural firings, networks of neurons and several other features of the brain viewed at a specific level of description. In Section 4, where the typical neural networks were described, the models introduced were invariably placed in the biological context. Even in the case of the Hopfield network, where this description was not so explicit, it was discussed that a continuous model of such network uses a simple integrate-and-fire neuron, and thus the network can also be viewed as searching for minima in an energy surface. Table 2 summarizes the interpretation of biological neural networks as artificial neural networks.

Table 2: Interpretation of the biological terminology into the computational domain.

Biology (Nervous System)	Artificial Neural Networks
Neuron	Unit, node, or (artificial) neuron
Synaptic contact	Connection
Synaptic efficacy (such as the number of vesicles released with each action potential, the amount of neurotransmitter within each vesicle, and the total number of channel receptors exposed to neurotransmitters)	Weight, connection strength, or synaptic strength
Neural integration over the input signals	Net input to the neuron
Triggering of an action potential (spike)	Output of the neuron given by an activation function applied to its net input
Limited range firing rate	Limited output of the processing strength by an activation function
Threshold of activation	Bias
Laminae (layers) of neurons	Layers of neurons
Network of neurons	Network of neurons
Excitation/inhibition	Positive/negative weight

6. Scope of Neurocomputing

Artificial neural networks can, in principle, compute any computable function, i.e., they can do everything a standard digital computer can do. Practical applications of neural networks most often employ supervised learning, where the training data includes both the input and the desired outputs or results. After successful training, the input data can be presented to the network that is then used to compute an output value that approximates the desired output. However, for many applications it may be hard to select appropriate input data and to code the data, if necessary, as numeric values.

Artificial neural networks are particularly useful for clustering, classification and function approximation (mapping) problems to which hard and fast rules cannot be easily applied. As discussed in Section 4.4, multi-layer perceptrons are universal function approximators, thus any finite-dimensional vector function on a compact set can be approximated to an arbitrary precision by these networks.

Self-organizing feature maps, like the one described in this report, find applications in diverse fields, such as unsupervised categorization of data (clustering), pattern recognition, remote sensing, robotics, processing of semantic information, and knowledge discovery in databases (data mining). Recurrent networks have been broadly applied to time series prediction, pattern recognition, system identification and control, and natural language processing.

Vasts lists of practical and commercial applications, including links to other sources, of neural networks can be found at the following sites:

- ftp://ftp.sas.com/pub/neural/FAQ7.html#A_applications
- <http://www.emsl.pnl.gov:2080/proj/neuron/neural/products/>

7. Summary

The nervous system is one of the major control systems of living organisms. It is responsible, among other things, for receiving stimuli from the environment, processing them, and then promoting an output response. It has been very influential in the design of computational tools for problem solving that use neuron-like elements as the main processing units.

In a simplified form, the nervous system is composed of nerve cells, mainly neurons, structured in networks of neurons. Through synaptic modulation the nervous system is capable of generating internal representations of environmental stimuli, to the point that these stimuli become unnecessary for the emergence of ideas and creativity. The internal models themselves are capable of self-stimulation.

Artificial neural networks were designed as highly abstract models based upon simple mathematical models of biological neurons. The assembly of artificial neurons in network structures allow the creation of powerful neurocomputational devices, with applications to several domains, from pattern recognition to control.

This report introduced the field of neural networks by presenting a framework to design ANN (neuron models, network structures and learning algorithms), and then proceeded with a review of some of the most well-known types of neural networks. Examples of applications of all the reviewed networks were also provided.

8. A Quick Guide to the Literature

- Anderson, J. A. (1995), *An Introduction to Neural Networks*, The MIT Press.
Anderson presents a very good attempt at linking artificial neural networks with neuroscience. His book is very readable and discusses the biological plausibility and motivation of several neural network types.
- Churchland, P. S. and Sejnowski, T. J. (1994), *The Computational Brain*, MIT Press.
- O'Reilly, R. C. and Munakata, Y. (2000), *Computational Explorations in Cognitive Neuroscience: Understanding the Mind by Simulating the Brain*, The MIT Press.
- Dayan, P. and Abbot, L. F. (2001), *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*, The MIT Press.
- Trappenberg, T. (2002), *Fundamentals of Computational Neuroscience*, Oxford University Press.

The field of computational neuroscience is an emerging field of research that focuses on the understanding of how the brain works by using biologically-based computational models made up of networks of neuron-like units. The four books above are some of the most recent works that describe computational models that are more biologically plausible than the artificial neural networks described here. However, all of them bring (extensive) discussions of the plausibility of ANN as biological models. The importance of ANN for neuroscience becomes evident.

- Fausett, L. (1994), *Fundamentals of Neural Networks: Architectures, Algorithms, and Applications*, Prentice Hall.
Fausett's book is a very comprehensible textbook introducing the most basic neural network algorithms.

- Hagan, M. T., Demuth, H. B. and Beale, M. H. (1996), *Neural Network Design*, PWS Publishing Company.
This book is particularly interesting for readers familiar with linear algebra and programming languages like Matlab® and Mathematica®. In particular, it was written by some of the contributors of the Matlab® neural network toolbox.

- Haykin, S. (1999), *Neural Networks: A Comprehensive Foundation*, 2nd Ed., Prentice Hall.
Haykin's book is a must-read textbook about neural networks. It is written primarily for graduate students and provides detailed mathematical description of the hows, whys and whens the models described work.

- Hebb, D. O. (1949), *The Organization of Behavior: A Neuropsychological Theory*, New York: Wiley.

Hebb's describes one of the first learning rules for neural networks. The basic premise of this work is that behavior can be explained by the action of neurons.

- Hopfield, J. J. (1982), "Neural Networks and Physical Systems with Emergent Collective Computational Capabilities", *Proc. of the Nat. Acad. of Sci. USA*, **79**, pp. 2554-2558.
- Hopfield, J. J. (1984), "Neurons with Graded Response Have Collective Computational Properties like those of Two-State Neurons", *Proc. of the Nat. Acad. of Sci. USA*, **81**, pp. 3088-3092.

These two pioneer papers describe how a discrete-time and continuous-time network, respectively, behave as content-addressable memories. Hopfield demonstrates the similarity of these models with some physical systems and discusses their biological plausibility under the perspective of neuroscience.

- Kohonen, T (1990), "The Self-Organizing Map", *Proceedings of the IEEE*, **78**(9), pp. 1464-1480.

- Kohonen, T. (1982), “Self-Organized Formation of Topologically Correct Feature Maps”, *Biological Cybernetics*, **43**, pp. 59-69.
- Kohonen, T. (2000), *Self-Organizing Maps*, 3rd Ed., Springer-Verlag.
Kohonen’s works introduce competitive networks with particular emphasis on the self-organizing maps, and provide extensive analyses of linear associative networks.
- McCulloch, W. and Pitts, W. H. (1943), “A Logical Calculus of the Ideas Immanent in Nervous Activity”, *Bulletin of Mathematical Biophysics*, **5**, pp. 115-133.
This paper introduces the first mathematical model of a neuron in which a weighted sum of the input signals is compared to a threshold in order to determine the firing of the neuron.
- Minsky, M. L. and Papert, S. A. (1969), *Perceptrons*, MIT Press.
A book that was responsible for a massive abandonment of research and funding in artificial neural networks. It rigorously demonstrated the limitations of single layer perceptrons and speculated few hope for the proposal of learning algorithms for multi-layer networks.
- Rumelhart, D. E., McClelland, J. L. and the PDP Research Group (1986), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*, The MIT Press.
- McClelland, J. L., Rumelhart, D. E. and the PDP Research Group (1986), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 2: Psychological and Biological Models*, The MIT Press.
These two volumes have for a long time been the most cited works in the field of artificial neural networks. They played a major role in the resurgence of interest in the field of ANN. Among the many aspects discussed, the backpropagation learning algorithm is fully described (vol. 1) and the biological plausibility of neural networks (vol. 2) was also presented.
- Artificial Neural Networks FAQ (Frequently Asked Questions): On line website [November 27th 2002] <ftp://ftp.sas.com/pub/neural/FAQ.html>
The FAQ website for neural networks is worth looking at, mainly the part describing books about artificial neural networks. A variety of texts is presented along with some comments about their level and contents. It presents links to free and commercial software, hardware, data sets, everything one may need to understand, play with, and apply ANNs.
- Journals: IEEE Transactions on Neural Networks, Neural Networks, International Journal of Neurocomputing, International Journal of Neural Systems, Neural Computation, Neural Computing Surveys, Neural Processing Letters, Neural Network News, Network: Computation in Neural Systems, Connection Science: Journal of Neural Computing, International Journal of Neural Networks, Neural Networks Today.
- Conferences: IEEE International Joint Conference on Neural Networks, International Conference on Artificial Neural Networks, International Conference on Neural Information Processing, European Symposium on Neural Networks, International Conference on Cognitive and Neural Systems.
These are the most well-known international periodicals and conferences of the field. Further and more detailed information can be found at the ANN FAQ web site given above.

9. Exercises

9.1. Questions

1. It has been discussed, in Section 2, that a neuron can fire by receiving input signals from other neurons. In most cases we assume that these signals come from the environment through sensory cells in the skin, eyes, and so forth. But we also know that the brain is capable of presenting thought processes from stimuli generated within itself. Discuss how these “self-stimuli” can be generated?
2. Artificial neural networks have been extensively used in pattern recognition tasks, such as character and face recognition. Independently of the potentiality of the network implemented, would you expect an ANN to outperform human beings, for instance in noisy character recognition? Justify your answer.

9.2. Computational Exercises

1. Section 3.1.2 introduced the McCulloch and Pitts neuron and presented one form of implementing it so as to simulate the logical AND and OR functions. Assuming there is a NOT neuron as illustrated in Figure 43, connect some AND, OR and NOT neurons such that the resulting artificial neural network performs the XOR function with inputs a and b . Draw the resultant network and describe the required threshold for each neuron in the network.



Figure 43: A NOT neuron and the truth-table for the XOR function.

2. In Section 4.1, the extended Hebbian updating rule for the weight vector was presented. Write a pseudocode for the Hebb network assuming the following:

Input patterns: $\mathbf{X} \in \mathbb{R}^{m \times N}$;

Desired outputs: $\mathbf{d} \in \mathbb{R}^{1 \times N}$;

Network outputs: $\mathbf{y} \in \mathbb{R}^{1 \times N}$;

Bias: $b \in \mathbb{R}$.

The procedure should return as outputs the network output y for each training pattern, the weight vector \mathbf{w} and the bias term b .

3. Apply the extended Hebb network procedure, written in the previous exercise to realize the Boolean functions AND and OR. What happens when the second, third, and fourth patterns are presented to the network? Explain why the learning rule is not capable of generating a correct decision line?

Change the binary vector of desired outputs by a bipolar vector, that is, every value ‘0’ in vector \mathbf{d} should correspond to a value ‘-1’, as illustrated in Figure 44, and re-apply

the Hebb network to this problem. What happens? Is the network capable of solving the problem? Justify your answer.

Inputs		Desired output	Inputs		Desired output
x_1	x_2	$x_1 \text{ AND } x_2$	x_1	x_2	$x_1 \text{ OR } x_2$
0	0	-1	0	0	-1
0	1	-1	0	1	+1
1	0	-1	1	0	+1
1	1	+1	1	1	+1

Figure 44: Boolean functions AND and OR with bipolar outputs.

4. Apply the simple perceptron algorithm (Algo 1) to the AND and OR functions with binary inputs and desired outputs. Initialize the weight vector and bias all with zero, and with small random numbers as well. Compare the results of the initialization methods and draw the decision lines.
5. Apply the simple perceptron algorithm (Algo 1) to the XOR function. Is it capable of finding a decision line to correctly classify the inputs into '1' or '0'? Why using bipolar inputs and/or desired outputs does not solve the problem?
6. Implement the perceptron algorithm for multiple output neurons (Algo 2) and apply it to classify the characters presented in Figure 24.

Insert some random noise in the training data and test the network sensitivity in relation to the noise level. That is, test how the network performance degrades in relation to the level of noise introduced in the test set.

7. Determine the gradient of an arbitrary bias for an ADALINE network with linear output nodes. That is,

$$\text{Determine } \frac{\partial \mathfrak{J}}{\partial b_i}, \text{ where } \mathfrak{J} = \sum_{i=1}^o e_i^2 = \sum_{i=1}^o (d_i - y_i)^2, \text{ and } y_i = \sum_j w_{ij} \cdot x_j + b_i, j = 1, \dots, m.$$

8. Write a pseudocode for the ADALINE network.
9. Algo 3 presents the algorithm to run the single-layer perceptron network. Write a pseudocode for an algorithm to run the multi-layer perceptron (MLP) network. Assume generic activation functions and a number of hidden layers to be entered by the user. (The trained weight matrices are entered as inputs to the algorithm.)
10. Apply an MLP network with two sigmoid hidden units and a single output unit trained with the backpropagation learning algorithm to solve the XOR problem.
11. Apply an MLP network with threshold (McCulloch and Pitts) units, two hidden and a single output unit, trained with the backpropagation learning algorithm to solve the XOR problem.

Determine the weights of the network and draw the decision boundaries constructed for each hidden unit, and the decision boundary constructed by the complete network.

12. It is known that the network presented in Figure 45 is capable of solving the XOR problem with a single hidden unit.

Can you apply the backpropagation algorithm to train this network? If yes, how? If not, why?

Find an appropriate set of weights so that the network with sigmoid hidden units and linear (or sigmoid) output unit solves the XOR problem.

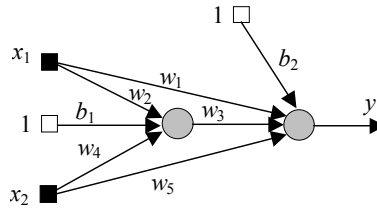


Figure 45: Network with connections from the input units directly to the output unit.

13. Train an MLP network to approximate one period of the function $\sin(x)\cos(2x)$ with only 36 training samples. Test the network sensitivity with relation to:
 - a) the number of hidden units; and
 - b) the learning rate α .
14. Insert some noise into the data set of the previous exercise and re-train the MLP network with this new data set. Study the generalization capability of this network (Section 3.3.1).
15. Apply the MLP network trained with the backpropagation learning algorithm to solve the character recognition task of Section 4.2.4.
Determine a suitable network architecture and then test the resultant network sensitivity to noise in the test data.
16. The *encoding/decoding* (ENC/DEC) problem is a sort of *parity* problem, that is, the output is required to be one if the input pattern contains an odd number of ones, and zero otherwise. Assuming a data set with 10 samples ($N = 10$) of dimension 10 each ($m = 10$), and a network with 10 outputs ($o = 10$), the input vector is composed of N bits, from which only one has a value of '1', and the others have values '0'. The desired output vector is identical to the input vector so that the network has to make an auto-association between the input and output values. The goal is to learn a mapping from the m inputs to the h hidden units (*encoding*) and then from the h hidden units to the o outputs (*decoding*). Usually, in a network m - h - o (m inputs, h hidden units, and o outputs), $h < N$, where N is the number of input patterns. If $h \leq \log_2 N$ it is said to be a tight encoder.
Train an MLP network with $h = 10$ to solve an ENC/DEC problem of dimension 10.
Is it possible to train an MLP network so that it becomes a tight encoder?
17. Apply the self-organized map to the Animals data set presented in Section 4.5.3. Consider a bi-dimensional output grid with 10 rows by 10 columns of neurons (10×10).
18. Implement the discrete Hopfield network and apply it to the character recognition problem, as described in Section 4.6.5. Test the network sensitivity to noise, that is, introduce noise in the patterns with increasing probability (5%, 10%, 15%, 20%, 25%,...) and study the error-correcting capability of the network.
19. A more efficient rule, than the generalized Hebb's rule, to define the weight matrix of a discrete Hopfield network is the *projection rule* or *pseudo-inverse rule*. Although

this rule does not present the biological motivation of the Hebb's rule, it explores some algebraic features of the equilibrium points.

A vector \mathbf{x}_i is going to be an equilibrium point if $\mathbf{W}\mathbf{x}_i = \mathbf{x}_i$, $i = 1, \dots, n$. Let \mathbf{X} be the matrix of all input patterns, then $\mathbf{W}\mathbf{X} = \mathbf{X}$. A solution to this equation can be given by:

$$\mathbf{W} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T.$$

The necessary condition for $(\mathbf{X}^T\mathbf{X})^{-1}$ to exist is that the vectors \mathbf{x}_i , $i = 1, \dots, n$ be linearly independent, and thus, matrix \mathbf{X} has full rank.

Test if the input patterns used in the previous exercise are linearly independent (see Appendix B) and use the projection rule to store the patterns in the discrete Hopfield network. Test the error-correcting capability of the network trained with this rule and compare with the results obtained in the previous exercise.

9.3. Thought Exercises

1. Discuss the similarities and differences between the perceptron and the ADALINE updating rules. Why is the ADALINE more powerful than the perceptron algorithm?
2. Provide a graph description of a neural network, more specifically, propose a signal-flow graph of the artificial neuron depicted in Figure 9.

9.4. Projects and Challenges

1. In Section 3.1.3 a basic integrate-and-fire neuron model was presented according to Equations (1) to (3). Equation (1) corresponds to an inhomogeneous linear differential equation that can be solved numerically, and also analytically in some cases. Assuming that the sum of the input currents to the neuron $i(t)$ is constant, that is, $i(t) = i$, determine the analytical solution to Equation (1).

Assume now that the activation threshold is set to $\theta = 10$. Plot a graph with the variation of the solution to Equation (1), $u(t)$, as a function of time t for $Ri = 8$.

Finally, plot a graph with the variation of the solution to Equation (1), $u(t)$, as a function of time t for $Ri = 10$. If the neuron fires a spike, reset its membrane potential to a fixed value $u_{\text{res}} = 1$.

Can you observe spikes in your graphs? If not, explain why.

2. In Section 4.5.1 the SOM learning algorithm was presented. A competitive learning rule based upon the minimization of the Euclidean distance between the input pattern and the neurons' weights was presented. Another criterion to determine the winning neuron, i.e., the best matching neuron, is based on the maximization of the inner product $\mathbf{w}_j^T \mathbf{x}$.

Show that minimizing the Euclidean distance between a given input pattern and the neurons' weights is mathematically equivalent to maximizing the inner product $\mathbf{w}_j^T \mathbf{x}$.

Implement this new competitive rule and compare qualitatively the results with those obtained with the Euclidean distance.

Use the same initialization values for the network weights and the same parameters for the algorithm and compare quantitatively and qualitatively both competitive strategies.

3. Explain why a multi-layer feedforward neural network with sigmoidal output units cannot learn the simple function $y = 1/x$, with $x \in (0,1)$. How can you modify this network so that it can learn this function.

References

- [1] Anderson, J. A. (1995), *An Introduction to Neural Networks*, The MIT Press.
- [2] Cybenko G. (1989), "Approximation by Superpositions of a Sigmoidal Function", *Mathematics of Control Signals and Systems*, **2**, pp. 303-314.
- [3] Dayan, P. and Abbot, L. F. (2001), *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*, The MIT Press.
- [4] de Castro, L. N. (1998), *Analysis and Synthesis of Learning Strategies for Artificial Neural Networks* (in Portuguese), M.Sc. Dissertation, School of Computer and Electrical Engineering, State University of Campinas, Brazil, 246 p.
- [5] Fausett, L. (1994), *Fundamentals of Neural Networks: Architectures, Algorithms, and Applications*, Prentice Hall.
- [6] Hagan, M. T., Demuth, H. B. and Beale, M. H. (1996), *Neural Network Design*, PWS Publishing Company.
- [7] Haykin, S. (1999), *Neural Networks: A Comprehensive Foundation*, 2nd Ed., Prentice Hall.
- [8] Hebb, D. O. (1949), *The Organization of Behavior: A Neuropsychological Theory*, New York: Wiley.
- [9] Hinton, G. E. and McClelland, J. L. (1987), "Learning Representation by Recirculation", In D. Z. Anderson (ed.), *Neural information Processing Systems*, New York: American Institute of Physics, pp. 358-366.
- [10] Hopfield, J. J. (1982), "Neural Networks and Physical Systems with Emergent Collective Computational Capabilities", *Proc. of the Nat. Acad. of Sci. USA*, **79**, pp. 2554-2558.
- [11] Hopfield, J. J. (1984), "Neurons with Graded Response Have Collective Computational Properties like those of Two-State Neurons", *Proc. of the Nat. Acad. of Sci. USA*, **81**, pp. 3088-3092.
- [12] Kelso, S. R., Ganong, A. H. and Brown, T. H. (1986), "Hebbian Synapses in Hippocampus", *Proc. of the Nat. Ac. of Sciences U.S.A*, **83**, pp. 5326-5330.
- [13] Kohonen, T. (1990), "The Self-Organizing Map", *Proceedings of the IEEE*, **78**(9), pp. 1464-1480.
- [14] Kohonen, T. (1982), "Self-Organized Formation of Topologically Correct Feature Maps", *Biological Cybernetics*, **43**, pp. 59-69.
- [15] Kohonen, T. (2000), *Self-Organizing Maps*, 3rd Ed., Springer-Verlag.
- [16] McClelland, J. L., Rumelhart, D. E. and the PDP Research Group (1986), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 2: Psychological and Biological Models*, The MIT Press.
- [17] McCulloch, W. and Pitts, W. H. (1943), "A Logical Calculus of the Ideas Immanent in Nervous Activity", *Bulletin of Mathematical Biophysics*, **5**, pp. 115-133.
- [18] Minsky, M. L. and Papert, S. A. (1969), *Perceptrons*, MIT Press.
- [19] Narendra, K. S. and Pathasarathy, K., (1990), "Identification and Control of Dynamical Systems Using Neural Networks", *IEEE Trans. on Neural Networks*, **1**(1), pp. 4-27.

- [20] Nigrin, A. (1993), *Neural Networks for Pattern Recognition*, Cambridge, MA: The MIT Press.
- [21] O'Reilly, R. C. (1996), "Biologically Plausible Error-Driven Learning Using Local Activation Differences: The Generalized Recirculation Algorithm", *Neural Computation*, **8**(5), pp. 985-938.
- [22] O'Reilly, R. C. and Munakata, Y. (2000), *Computational Explorations in Cognitive Neuroscience: Understanding the Mind by Simulating the Brain*, The MIT Press.
- [23] Ritter, H. and Kohonen, T. (1989), "Self-Organizing Semantic Maps", *Biological Cybernetics*, **61**, pp. 241-254.
- [24] Rosenblatt, F. (1958), "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain", *Psychological Review*, **65**, pp. 386-408.
- [25] Rosenblatt, F. (1962), *Principles of Neurodynamics*, Spartan Books.
- [26] Rumelhart, D. E., McClelland, J. L. and the PDP Research Group (1986), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*, The MIT Press.
- [27] Sutton, R. S. and Barto, A. G. (1998), *Reinforcement Learning: An Introduction*, A Bradford Book.
- [28] Trappenberg, T. (2002), *Fundamentals of Computational Neuroscience*, Oxford University Press.
- [29] von Neumann, J. (1982), "First Draft of a Report on the EDVAC", In B. Randall (ed.), *The Origins of Digital Computers: Selected Papers*, 3rd Ed., Springer, Berlin.
- [30] Widrow, B. and Hoff, M. E. Jr. (1960), "Adaptive Switching Circuits", *WESCON Convention Record*, **4**, pp. 96-104.
- [31] Wiener, N. (1948), *Cybernetics*, New York: Wiley.