
UNIT 4 FUNDAMENTALS OF NEURAL NETWORKS

Structure	Page No
4.1 Introduction	5
Objectives	
4.2 Neural Networks	6
4.3 Models of a Neuron	10
4.4 Threshold Logic	16
4.5 Error-correction Learning	21
4.6 Summary	30
4.7 Solutions/Answers	30
4.8 References	31

4.1 INTRODUCTION

Research in the field of neural networks has been attracting increasing attention in recent years. Since 1943, when Warren McCulloch and Walter Pitts presented the first model of artificial neurons, new and more sophisticated proposals have been made from decade to decade. Mathematical analysis has solved some of the mysteries posed by the new models but has left many questions open for future investigations. Needless to say, the study of neurons, their interconnections, and their role as the brain's elementary building blocks is one of the most dynamic and important research fields in modern biology.

We shall begin this unit by defining the introduction to neural networks in Sec. 4.2 called neuron, which is the simplest kind of computing units used to build artificial neural networks.

Several associative neural memory models have been proposed over the last two decades e.g., Amari, 1972a; Anderson, 1972; Nakano, 1972; Kohonen, 1972 and 1974; Kohonen and Ruohonen, 1973; Hopfield, 1982; Kosko, 1987; Okajima et al., 1987; Kanerva, 1988; Chiueh and Goodman, 1988; Baird, 1990. These memory models can be classified in various ways depending on their architecture (static versus recurrent), their retrieval mode (synchronous versus asynchronous), the nature of the stored associations (autoassociative versus heteroassociative), the complexity and capability of the memory storage/recording algorithm, etc. A simple static synchronous associative memory is presented along with appropriate memory storage recipes. Then, this simple associative memory is extended into a recurrent auto associative memory by employing feedback. We shall describe these models in Sec. 4.3.

Computing elements neurons are a generalization of the common logic gates used in conventional computing and, since they operate by comparing their total input with a threshold, this field of research is known as threshold logic. We shall discuss threshold logic in Sec. 4.4.

Error correction rules were initially proposed as ad hoc rules for single unit training. These rules essentially drive the output error of a given unit to zero. In Sec. 4.5, we shall start with the classical perceptron learning rule and give a proof for its convergence.

Objectives

After studying the unit, you should be able to:

- define with artificial neural networks, relate between neurons and artificial computing units;
- describe the simplest kind of computing units used to build artificial neural networks;
- generalise the common logic gates used in conventional computing.

4.2 NEURAL NETWORKS

Works on artificial neural networks, commonly referred to as 'neural networks'.

Artificial neural networks are an attempt to do the modeling of the information processing capabilities of nervous systems. Thus, first of all, it is necessary to consider the essential properties of biological neural networks from the viewpoint of information processing. By this, we can design abstract models of artificial neural networks, which can then be simulated and analyzed.

Neural networks theory always emphasizes that the human brain computers in an entirely different way from the conventional digital computer. Although the models which have been proposed to explain the structure of the brain and the nervous systems of some animals are different in many respects, there is a general consensus that the essence of the operation of neural ensembles is "control through communication". Animal nervous systems are composed of thousands or millions of interconnected cells. Each one of them is a very complex arrangement which deals with incoming signals in many different ways. However, neurons are rather slow when compared to electronic logic gates. These can achieve switching times of a few nanoseconds, whereas neurons need several milliseconds to react to a stimulus. Nevertheless the brain is capable of solving problems which no digital computer can yet efficiently deal with. Massive and hierarchical networking of the brain seems to be the fundamental precondition for the emergence of consciousness and complex behavior. So far, however, biologists and neurologists have concentrated their research on uncovering the properties of individual neurons. Today, the mechanisms for the production and transport of signals from one neuron to the other are well-understood physiological phenomena, but how these individual systems cooperate to form complex and massively parallel systems capable of incredible information processing feats has not yet been completely elucidated. Mathematics, physics, and computer science can provide invaluable help in the study of these complex systems. It is not surprising that the study of the brain has become one of the most interdisciplinary areas of scientific research in recent years. However, we should be careful with the metaphors and paradigms commonly introduced when dealing with the nervous system. It seems to be a constant in the history of science that the brain has always been compared to the most complicated contemporary artifact produced by human industry.

In ancient times the brain was compared to a pneumatic machine, in the Renaissance to clockwork, and at the end of the last century to the telephone network. There are some today who consider computers the paradigm par excellence of a nervous system. It is rather paradoxical that when John von Neumann wrote his classical description of future universal computers, he tried to choose terms that would describe computers in terms of brains, not brains in terms of computers. The nervous system of an animal is an information processing totality. The sensory inputs, i.e., signals from the environment, are coded and processed to evoke the appropriate response. Biological neural networks are just one of many possible solutions to the problem of processing information. The main difference between neural networks and conventional computer systems is the massive parallelism and redundancy which they exploit in order to deal with the unreliability of the individual computing units. Moreover, biological neural networks are self-organizing systems and each individual neuron is also a delicate self-organizing structure capable of processing information in many different ways.

We can cite an example of human vision. Human vision is the function of visual system, which provides the information we need to interact with the environment. It is an information-processing task.

In this way, we can generalised neural network as a machine which is designed to model the way in which the brain performs a particular task or function of interest. A neural network has a complex interconnection of simple computing cells. These cells are known as processing units or 'neurons'. The benefits of neural networks are as follows:

- i) The most important property of a neuron is non-linearity. All neurons are interconnected in non linear form.
- ii) Neural networks offer input-output mapping, that is a unique input signal has a corresponding desired output.
- iii) Neural networks are adaptive to changes in the surrounding environment.
- iv) Neural networks are used to provide the information not only to select a particular pattern, but also to find the confidence in the decision made.
- v) A neural network can be designed to deal with a contextual information.
- vi) Neural network degrades gracefully under adverse operating conditions.
- vii) Neural network is well suited for implementations using VLSI (Very large scale integrated) technology.
- viii) Neural networks enjoy uniformity of analysis and design.
- ix) Neural networks are applicable for the interpretation of neural biological phenomenon as research tool.

Models of Computation

Artificial neural networks can be considered as just another approach to the problem of computation. The first formal definitions of computability were proposed in the 1930s and '40s and at least five different alternatives were studied at the time. The computer era was started, not with one single approach, but with a contest of alternative computing models. The five principal models of computation are described below.

1. The Mathematical Model

Mathematicians avoided dealing with the problem of a function's computability until the beginning of this century. This happened not just because existence theorems were considered sufficient to deal with functions, but mainly because nobody had come up with a satisfactory definition of computability, certainly a relative concept which depends on the specific tools that can be used. For example, the general solution for an algebraic equations of degree six cannot be formulated using only algebraic functions, yet this can be done if a more general class of functions is allowed as computational primitives. Another example, the squaring of the circle, is impossible using ruler and compass, but it has a trivial real solution. We can start to specify the available tools with the idea that some primitive functions and composition rules are "obviously" computable. All other functions which can be expressed in terms of these primitives and composition rules are then also computable. David Hilbert, the famous German mathematician, was the first to state the conjecture that a certain class of functions contains all intuitively computable functions. Hilbert was referring to the primitive recursive functions, the class of functions which can be constructed from the zero and successor function using composition, projection, and a deterministic number of iterations (primitive recursion). However, in 1928, Wilhelm Ackermann was able to find a computable function which is not primitive recursive. This led to the definition of the general recursive functions. In this formalism, a new composition rule has to be introduced, the so-called μ operator, which is equivalent to an indeterminate recursion

or a lookup in an infinite table. At the same time Alonzo Church and collaborators developed the lambda calculus, another alternative to the mathematical definition of the computability concept. In 1936, Church and Kleene were able to show that the general recursive functions can be expressed in the formalism of the lambda calculus. This led to the Church thesis that computable functions are the general recursive functions.

2. The Logic-Operational Model (Turing Machines)

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

3. The Computer Model

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

4. Cellular Automata

In machines like the Mark I and the ENIAC there was no clear separation between memory and processor, and both functional elements were intertwined. Some machines still worked with base 10 and not 2, some were sequential and others parallel. John von Neumann, who played a major role in defining the architecture of sequential machines, analyzed at that time a new computational model which he called cellular automata. Such automata operate in a “computing space” in which all data can be processed simultaneously. The main problem for cellular automata is communication and coordination between all the computing cells. This can be guaranteed through certain algorithms and conventions. It is not difficult to show that all computable functions, in the sense of Turing, can also be computed with cellular automata, even of the one-dimensional type, possessing only a few states. Turing himself considered this kind of computing model at one point in his career. Cellular automata as computing model resemble massively parallel multi-processor systems of the kind that has attracted considerable interest recently.

5. The Biological Model (Neural Networks)

The explanation of important aspects of the physiology of neurons set the stage for the formulation of artificial neural network models which do not operate sequentially, as Turing machines do. Neural networks have a hierarchical multilayered structure which sets them apart from cellular automata, so that information is transmitted not only to the immediate neighbors but also to more distant units. In artificial neural networks one can connect each unit to any other. In contrast to conventional computers, no program is handed over to the hardware—such a program has to be created, that is, the free parameters of the network have to be found adaptively. Although neural networks and cellular automata are potentially more efficient than conventional computers in certain application areas, at the time of their conception they were not yet ready to take center stage. The necessary theory for harnessing the dynamics of complex parallel systems is still being developed right before our eyes. In the meantime, conventional computer technology has made great strides. There is no better illustration for the simultaneous and related emergence of these various computability models than the life and work of John von Neumann himself. He participated in the definition and development of at least three of these models: in the architecture of sequential computers, the theory of cellular automata and the first neural network models. He also collaborated with Church and Turing in Princeton.

Artificial neural networks have, as initial motivation, the structure of biological systems, and constitute an alternative computability paradigm.

Elements of a Computing Model

What are the elementary components of any conceivable computing model? In the theory of general recursive functions, for example, it is possible to reduce any computable function to some composition rules and a small set of primitive functions. For a universal computer, we ask about the existence of a minimal and sufficient instruction set. For an arbitrary computing model, metaphoric expression has been proposed and is given in Fig. 1.

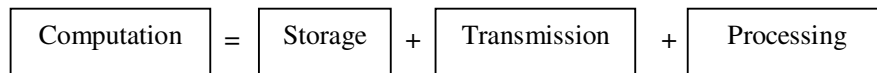


Fig. 1: Metaphoric expression for computing

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

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

In biological neural networks information is stored at the contact points between different neurons, the so-called **synapses**. Other forms of storage are also known, because neurons are themselves complex systems of self-organizing signaling.

Nervous systems possess global architectures of variable complexity, but all are composed of similar building blocks, the neural cells or **neurons**. They can perform

different functions, which in turn leads to a very variable morphology. If we analyze the human cortex under a microscope, we can find several different types of neurons. Although the neurons have very different forms, it is possible to recognize a hierarchical structure of different layers. Each one has specific functional characteristics. Neurons receive signals and produce a response. The general structure of a generic neuron has branches to the left which are the transmission channels for incoming information and are called **dendrites**. Dendrites receive the signals at the contact regions with other cells. Some animals have neurons with a very different morphology. In insects, for example, the dendrites go directly into the axon and the cell body is located far from them. Organelles in the body of the cell produce all necessary chemicals for the continuous working of the neuron. The mitochondria is a part used to supply energy of the cell, since they produce chemicals which are consumed by other cell structures. The output signals are transmitted by the axon, of which each cell has at most one. Some cells do not have an axon, because their task is only to set some cells in contact with others. These four elements, dendrites, synapses, cell body, and axon are the minimal structure we will adopt from the biological model. Artificial neurons for computing will have input channels, a cell body and an output channel. Synapses will be simulated by contact points between the cell body and input or output connections; a weight will be associated with these points.

Now try the following exercises.

E1) Write all the elements of a neurons.

E2) Describe all the models of computation.

In the following section, we shall discuss the model of neuron.

4.3 MODELS OF A NEURON

An artificial neuron takes input to produce output with the help of the transfer function or activation function. Therefore, different types of transfer functions are in use, such as threshold function, sigmoid function and others. These functions are discussed below.

- i) **Threshold function:** Threshold functions is also known as Heaviside function or Hard-limit transfer function. In this function, the output is 0 if the input is less than 0 otherwise the output is 1. Generally, it is used in a perceptron neuron. This is defined as

$$\phi(u) = \begin{cases} 0, & \text{if } u < 0 \\ 1, & \text{otherwise} \end{cases}$$

This function can be described diagrammatically as shown in Fig 2. The output of a neuron with threshold function can be expressed as

$$y_k = \begin{cases} 0 & \text{if } u_k < 0 \\ 1 & \text{otherwise} \end{cases}$$

where $u_k = \sum_{i=1}^n w_{k_i} x_i + b_k$, which is the induced local field.

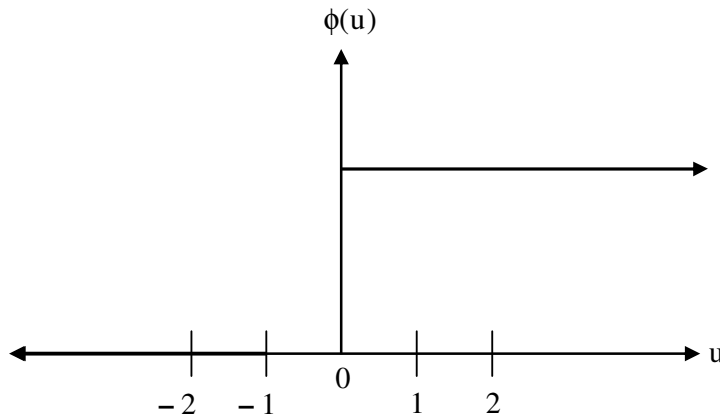


Fig. 2: Threshold function

- ii) **Linear Transfer function:** In this function, the output is same as the input within the range from -1 to 1 . This is expressed as

$$\phi(u) = u, \text{ where } -1 \leq u \leq 1$$

This is generally used in linear filter. The graph of this function is shown in Fig 3.

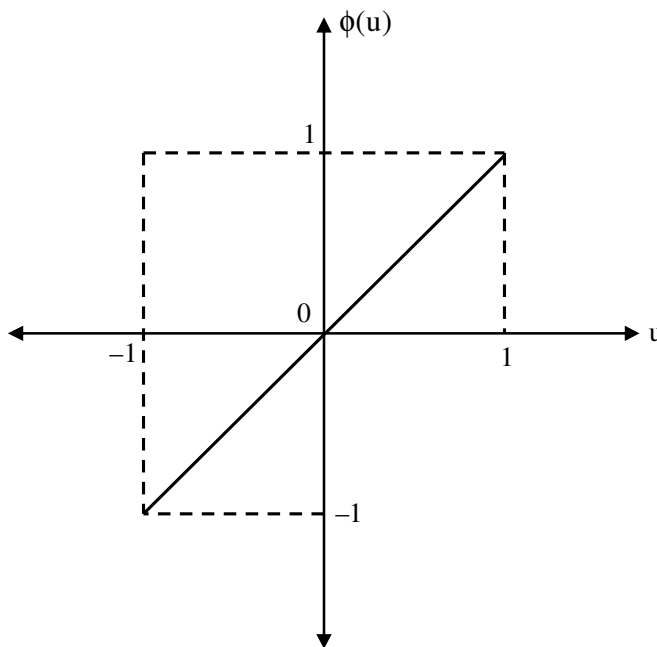


Fig. 3: Linear transfer function

- iii) **Sigmoid Function:** The range of the output of log-sigmoid is from 0 to 1 . It is given by

$$\phi(u) = \frac{1}{1 + e^{-au}}$$

where a is the slope parameter of the log-sigmoid function depends on the value of a . This function is used in a Back-propagation neural network. The graph of a sigmoid function is always S-shaped. It can be seen that for the infinite value of a , it becomes threshold function, which assumes the values 0 and 1 . A sigmoid function varies from 0 to 1 . Another example of sigmoid function is tan-sigmoid transfer function, which is defined as

$$\phi(u) = \tanh(u) = \frac{e^{au} - e^{-au}}{e^{au} + e^{-au}}$$

The output varies from -1 to 1. The graph of log-sigmoid and tan-sigmoid functions are shown in Fig. 4 and Fig. 5, respectively.

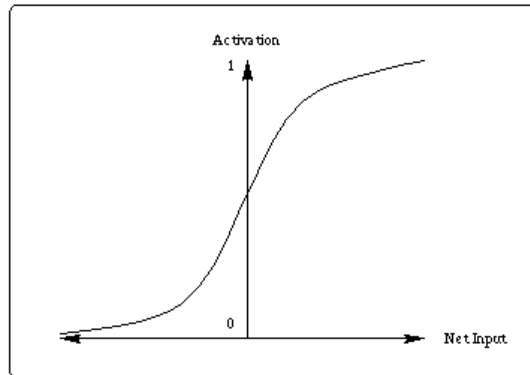


Fig. 4: Log-sigmoid function

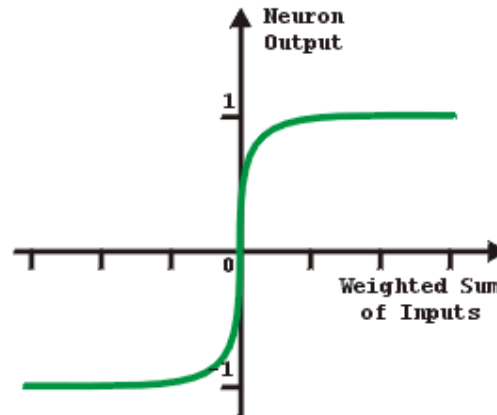


Fig. 5: Tan-sigmoid function

E3) What is the role of an activation function in neural networks?

So far, we introduced the neural networks. Now let us discuss the networks with primitive functions.

Networks of primitive functions

The structure of an abstract neuron is shown in Fig. 6 with n inputs. Each input channel i can transmit a real value x_i . The primitive function f computed in the body of the abstract neuron can be selected arbitrarily. Usually the input channels have an associated weight, which means that the incoming information x_i is multiplied by the corresponding weight w_i . The transmitted information is integrated at the neuron (usually just by adding the different signals) and the primitive function is then evaluated.

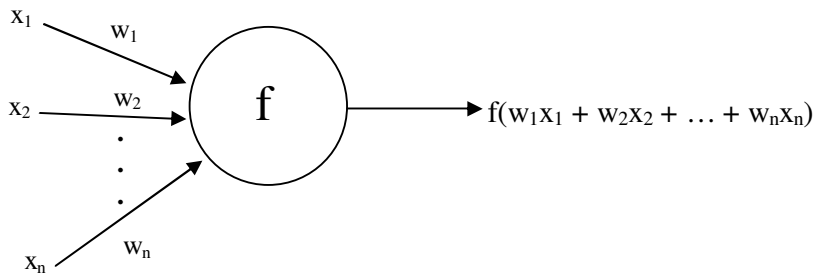


Fig. 6: An abstract neuron

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

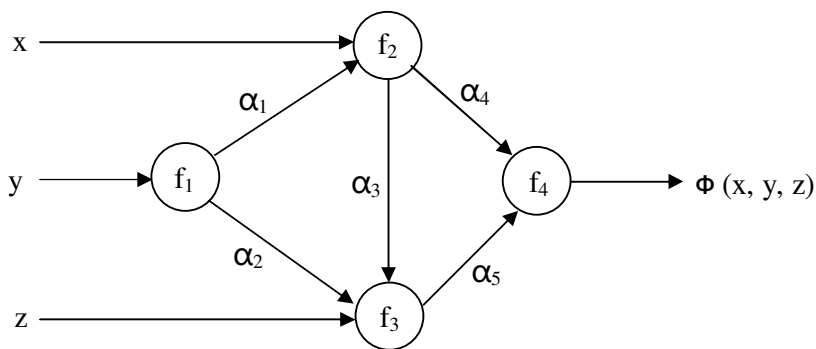


Fig. 7: Functional model of an artificial neural network

The digraph of a typical artificial neural networks have the structure shown in Fig. 7. The network can be thought of as a function Φ which is evaluated at the point (x, y, z) . The nodes implement the primitive functions f_1, f_2, f_3, f_4 which are combined to produce Φ and are written at vertices. Vertices are connected by the edges. The function Φ implemented by a neural network will be called the **network function** by mentioning weights at different edges. Different selections of the weights $\alpha_1, \alpha_2, \dots, \alpha_5$ produce different network functions.

Therefore, tree elements are particularly important in any model of artificial neural networks:

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

The neural architecture may be classified in the three fundamentally different classes of networks, which are shown in Fig. 8.

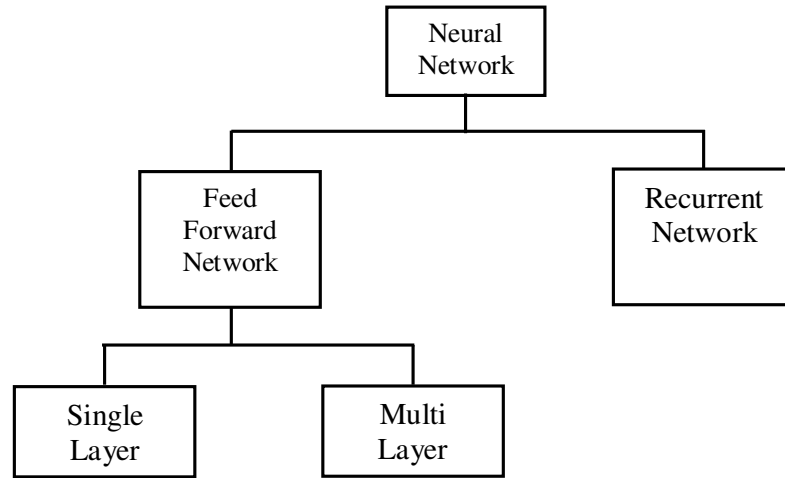


Fig. 8: Types of Neural Networks

For this, consider a input layer made up of input neurons x_1, x_2, \dots, x_n which receives the input signals and a output layer made up of output neurons y_1, y_2, \dots, y_n , receives output signals. The edges, which carry the synaptic links as weights connect every i^{th} input neurons to the j^{th} output neurons are labelled by weight w_{ij} , not vice-versa. If in a network the input layer is connected with a output layer in a acyclic way, then such a network is known as feed forward network. In feed forward network, only the output layer performs computation, therefore it is named as single layer feed forward network. In case of multilayer feed forward network, one or more hidden layers are connected to process. These hidden layers perform the useful intermediary computations, before directing the input to the output layer. These networks are shown in Fig. 9.

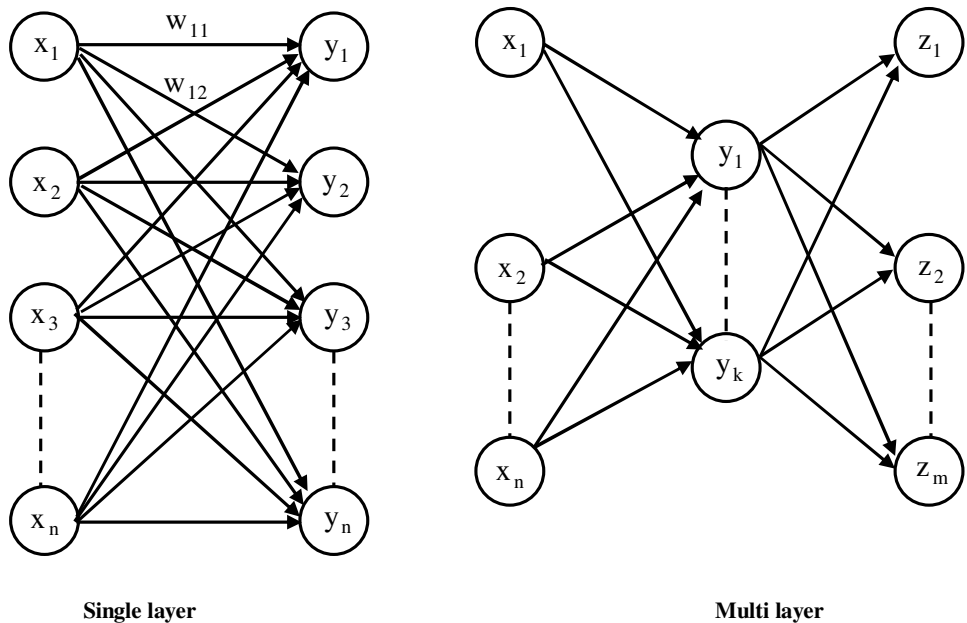


Fig. 9: Single layer and multi layer perceptrons

Recurrent networks are slightly different from the feed forward networks, in the sense that they work in cyclic way. In recurrent networks, the output layer is feedback into itself as input. These are shown in Fig. 10.

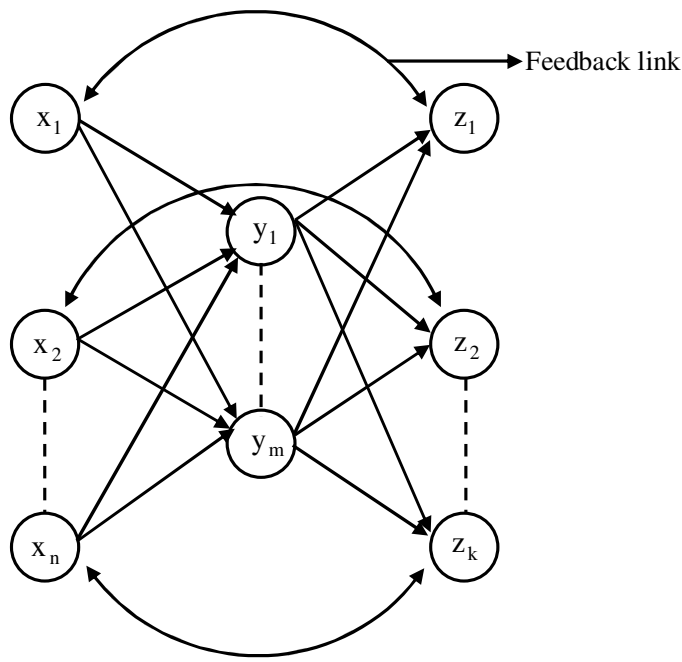


Fig. 10: Recurrent Networks

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

There are three aspects to the construction of a neural network:

1. **Structure** – the architecture and topology of the neural network.
2. **Encoding** – the method of changing weights/modifying weights.
3. **Recall** – the method and capacity to retrieve information.

Let's cover the first one – structure. This relates to how many layers the network should contain, and what their functions are, such as for input, for output, or for feature extraction. Structure also encompasses how interconnections are made between neurons in the network, and what their functions are.

The second aspect is encoding. Encoding refers to the paradigm used for the determination of and changing of weights on the connections between neurons. In the case of the multilayer feed-forward neural network, we initially can define weights by randomization. Subsequently, in the process of training, we can apply the backpropagation algorithm, which is a mean of updating weights starting from the output backwards. When the training is completed in the multilayer feed-forward neural network, the encoding ends since weights do not change after training is completed.

Finally, recall is also an important aspect of a neural network. Recall refers to getting an expected output for a given input. If the same input as before is presented to the network, the same corresponding output as before should result. The type of recall can characterize the network as being autoassociative or heteroassociative.

Autoassociation is the phenomenon of associating an input vector with itself as the output, whereas heteroassociation is that of recalling a related vector given an input vector.

The three aspects to the construction of a neural network mentioned above essentially distinguish between different neural networks and are part of their design process.

Now, try the following exercises.

-
- E4) Express the network function ϕ given in Fig. 7, in terms of the primitive functions f_1, f_2, f_3 and f_4 and of the weights $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ and α_5 .
 - E5) Differentiate between the biological neural networks and the artificial neural networks.
-

So far, we have introduced the neural networks. In the following section, we shall discuss the threshold logic.

4.4 THRESHOLD LOGIC

The computing elements are a generalization of the common logic gates used in conventional computing and, since they operate by comparing their total input with a threshold, which is shown in Fig. 11.

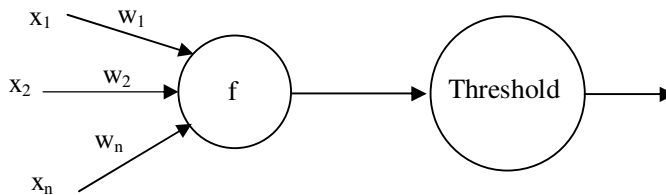


Fig. 11: Threshold logic

The first model we consider was proposed in 1943, by Warren McCulloch and Walter Pitts. McCulloch-Pitts networks are simpler, because they use solely binary signals, i.e., ones or zeros. The nodes produce only binary results and the edges transmit exclusively ones or zeros. The networks are composed of directed unweighted edges of excitatory or of inhibitory type. Each McCulloch-Pitts unit is assigned a certain threshold value θ .

Therefore, the McCulloch-Pitts model seems very limited. Since only binary information can be produced and transmitted. But it already contains all necessary features to implement the more complex models. In Fig. 12, following Minsky it will be represented as a circle with a black half. Incoming edges arrive at the white half, outgoing edges leave from the black half. Outgoing edges can fan out any number of times.

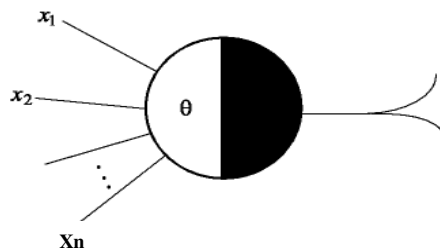


Fig. 12: Diagram of a McCulloch-Pitts unit

The rule for evaluating the input to a McCulloch-Pitts unit is given in the following steps:

Step 1: Assume that a McCulloch-Pitts unit gets an input x_1, x_2, \dots, x_n through n excitatory edges and an input y_1, y_2, \dots, y_m through m inhibitory edges.

Step 2: If $m \geq 1$ and at least one of the signals y_1, y_2, \dots, y_m is 1, the unit is inhibited and the result of the computation is 0. Otherwise the total excitation $x = x_1 + x_2 + \dots + x_n$ is computed and compared with the threshold θ of the unit (if $n=0$ then $x=0$). If $x \geq \theta$ the unit fires a 1, if $x < \theta$ the result of the computation is 0.

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

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

Now, let us define the conjunction, disjunction and negation in case of these units. Simple logical functions can be implemented directly with a single McCulloch-Pitts unit. The output value 1 represents the logical value true and 0, the logical value false. A single unit can compute the disjunction or the conjunction of n arguments. The logic elements can reduce the complexity of the circuit used to implement a given logical function.

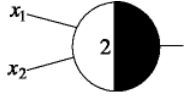
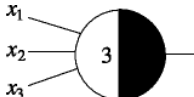
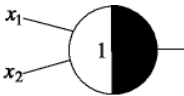
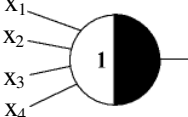
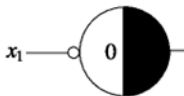
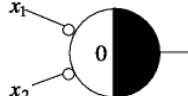
Gates	Implementation	Generalized
Disjunction (AND)		
Conjunction (OR)		
Negation (NOT)		

Fig. 13: Logical gates

Try the following exercise.

- E6) Find polynomial expressions corresponding to the OR, AND and NOT for x_1 and x_2 .

Geometric Interpretation

It is not easy to visualize the kind of functions that can be computed with McCulloch-Pitts cells by using a diagram. For this, the eight vertices of a three-dimensional unit cube can be considered. Each of the three logical variables x_1 , x_2 and x_3 can assume one of two possible binary values. There are eight possible combinations of x_1 , x_2 and x_3 which can be represented by the vertices of the cube. A logical function is just an assignment of a 0 or a 1 to each of the vertices. Fig. 13 shows one of these assignments. In the case of n variables, the cube consists of 2^n vertices and admits 2^{2^n} different binary assignments.

McCulloch-Pitts units divide the input space into two half-spaces. For a given input (x_1, x_2, x_3) and a threshold θ the condition $x_1 + x_2 + x_3 \geq \theta$ is tested, which is true for all points to one side of the plane with the equation $x_1 + x_2 + x_3 = \theta$ and false for all points to the other side (without including the plane itself in this case). Fig. 14 shows function values of a logical function of x_1 , x_2 and x_3 .

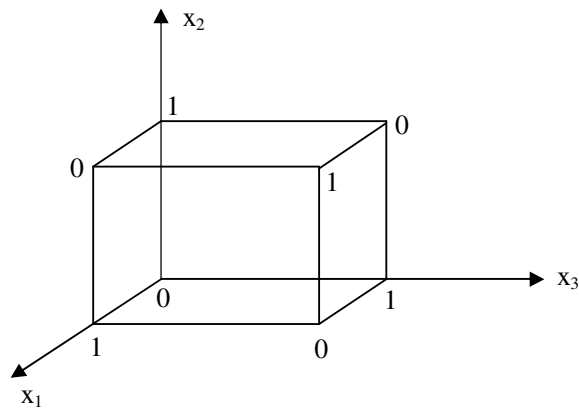


Fig. 14: Function values of a logical function of x_1 , x_2 and x_3

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

Example 1: Consider the vector $(1, 0, 1)$. It is the only one which fulfills the condition $x_1 \wedge \neg x_2 \wedge x_3$. This condition can be tested by a single computing unit.

Assume that a function F of three arguments has been defined according to the following table: To compute this function it is only necessary to decode all those vectors for which the function's value is 1. Fig. 15 shows a network capable of computing the function F .

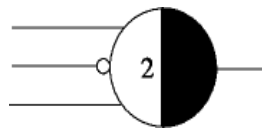


Fig. 15: Decoder for the vector (1, 0, 1)

Input vectors	F
(0, 0, 1)	1
(0, 1, 0)	1
All others	0

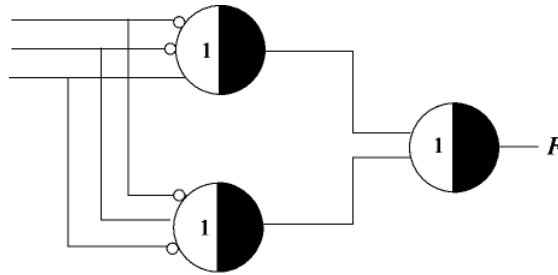


Fig. 16: Synthesis of the function F

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

Example 2: Assume that three weighted edges converge on the unit shown in Fig. 17(a). The unit computes Equivalent networks $0.3x_1 + 0.2x_2 + 0.4x_3 \geq 0.7$ or $2x_1 + 2x_2 + 4x_3 \geq 7$.

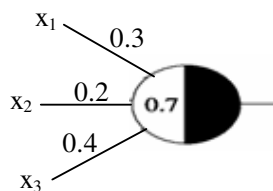


Fig. 17 (a): Weighted units

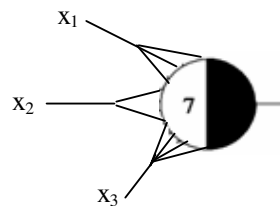


Fig. 17 (b): Equivalent computing unit

The Fig. 17 shows that positive rational weights can be simulated by simply fanning-out the edges of the network required number of times. This means that we can either use weighted edges or go for a more complex topology of the network, with many redundant edges.

The same can be done in the case of irrational weights if the number of input vectors is finite.

Applications of Threshold Logic

Threshold units can be used in any application in which we want to reduce the

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

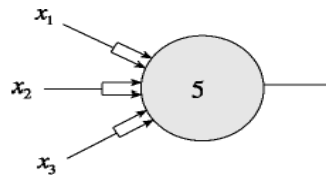


Fig. 18: Fault-tolerant gate

Threshold logic also offers a simpler way to achieve fault-tolerance. Fig. 18 shows an example of a unit that can be used to compute the conjunction of three inputs with inherent fault tolerance. Assume that three inputs x_1, x_2, x_3 can be transmitted, each with probability p of error. The probability of a false result when x_1, x_2 and x_3 are equal, and we are computing the conjunction of the three inputs, is $3p$, since we assume that all three values are transmitted independently of each other. But assume that we transmit each value using two independent lines. The gate of Fig. 18 has a threshold of 5, that is, it will produce the correct result even in the case where an input value is transmitted with an error.

The probability that exactly two ones arrive as zeros is p^2 and, since there are 15 combinations of two out of six lines, the probability of getting the wrong answer is $15p^2$ in this case. If p is small enough then $15p^2 < 3p$ and the performance of the gate is improved for this combination of input values. Other combinations can be analyzed in a similar way. If threshold units are more reliable than the communication channels redundancy can be exploited to increase the reliability of any computing system

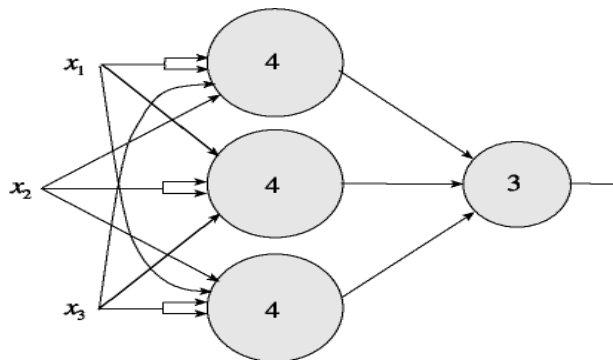


Fig. 19: A fault-tolerant AND built of noisy components

When the computing units are unreliable, fault tolerance is achieved using redundant

networks. Fig. 19 is an example of a network built using four units. Assume that the first three units connected directly to the three bits of input x_1, x_2, x_3 all fire with probability 1 when the total excitation is greater than or equal to the threshold θ but also with probability p when it is $\theta - 1$. The duplicated connections add redundancy to the transmitted bit, but in such a way that all three units fire with probability one when the three bits are 1. Each unit also fires with probability p if two out of three inputs are 1. However each unit reacts to a different combination. The last unit, finally, is also noisy and fires any time the three units in the first level fire and also with probability p when two of them fire. Since, in the first level, at most one unit fires when just two inputs are set to 1, the third unit will only fire when all three inputs are 1. This makes the logical circuit, the AND function of three inputs, built out of unreliable components error-proof.

Now, in the following section, we shall discuss error correction learning.

4.5 ERROR-CORRECTION LEARNING

Neural networks also learn in the same way as we learn from our surroundings. The learning in neural networks may be through a teacher or without a teacher. The learning with a teacher is referred as supervised learning while the learning without a teacher may be categorized as unsupervised learning and reinforced learning. All these learning are performed on neural networks as performed on human learning. In case of supervised learning, the teacher has the knowledge of environment. Here the teacher and the knowledge are considered a set of input-output examples. However, the environment is unknown to the neural networks. The block diagram of this process is given in Fig. 20.

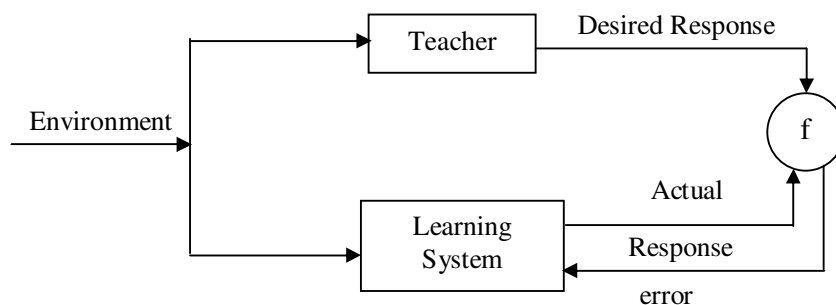


Fig. 20: Block diagram of learning process

If the teacher is not present, the system learns of its own by discovering and adapting to structural features in the input pattern, then the learning is unsupervised learning. While in turn, if the available teacher does not present the expected answer but only helps to identify whether the computed output is correct or incorrect, then the learning is reinforced learning. In this process of learning a reward is given for a correct answer and a penalty for wrong answer.

Consider that a teacher and a neural network are to state a training vector taken from the same environment. In this the teacher has the desired ability about the training vector to provide to the neural network. The response to be performed by the neural network is the optimum action, which is adjusted in between the training vector and the error signal. Therefore,

$$\text{The error signal} = \text{Desired response} - \text{the actual response}$$

The adjustment is made iteratively. By this, the knowledge of the teacher about environment is transferred to the neural networks with the help of training. This knowledge is stored in the form of fixed synaptic weights, in the neural networks,

which are represented as long-term memory. In this way, the neural networks are able to perform completely with the environment by itself. This type of supervised learning forms the basis of error-correction learning. In the block diagram of supervised learning, it is clear that the environment, which is not known, is outside the closed loop feedback system. The performance measures of such systems are in terms of mean square error, or the sum of squared errors over the training sample. These are defined as a function of synaptic weights of the system.

In this function, the free parameters are considered as coordinates, and this function is a multi-dimensional error surface or simply error surface. The true error surface is distributed uniformly over all the possible sets of input-output examples. Each point on the surface is a representative of the given operation under the supervision of a teacher. The point has to move down iteratively to reach towards a minimum point of the error surface. This minimum point is a local minimum. The supervised learning system performs this with the help of the information about the gradient of the error surface. The gradient at any point is a vector quantity. The example of such error surface is a "random walk".

Now the following tasks can be performed with the help of such learning processes.

- i) The most important task of a learning process is pattern recognition. Senses receive and then recognize the source of the data by making patterns like faces, voice on the telephone, smelling, etc.
- ii) Another important task is pattern association. In this associative memory plays an important role, that learn by associations.
- iii) Another important learning task is the approximation of functions.
- iv) Control is the learning task for neural networks. Control is of a plant, which may be a process of any system or critical part of the system. This part or process or plant is maintained in a controlled condition.
- v) The task of beamforming is done by the learning processes in neural networks. The spatial properties of a target signal and the background noise are distinguished through beamforming.

Error correction rules were initially proposed as ad hoc rules for single unit training. These rules essentially drive the output error of a given unit to zero. We start with the classical perceptron learning rule and give a proof for its convergence.

Throughout this section, an attempt is made to point out criterion functions that are minimized by using each rule. We will also cast these learning rules as relaxation rules, thus unifying them with the other gradient-based search rules.

i) Perceptron Learning Rule

Consider the following version of a linear threshold gate shown in Fig. 21. We will refer to it as the perceptron. The perceptron maps an input vector $x = x_1 x_2 \dots x_{n+1}^T$ to a bipolar binary output y , and thus it may be viewed as a simple two-class classifier. The input signal x_{n+1} is usually set to 1 and plays the role of a bias to the perceptron. We will denote by w the vector $[w = w_1 w_2 \dots w_{n+1}]^T$ consisting of the free parameters (weights) of the perceptron. The input/output relation for the perceptron is given by $y = \text{sgn}(x^T w)$, where sgn is the "sign" function which returns +1 or -1 depending on whether the sign of its scalar argument is positive or negative, respectively.

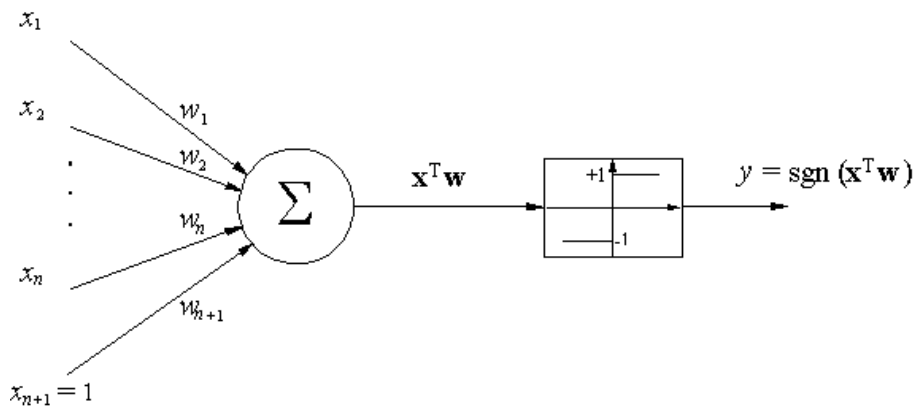


Fig. 21: The perceptron computational unit.

Assume we are training the above perceptron to load (learn) the training pairs: $\{x_1, d_1\}, \{x_2, d_2\}, \dots, \{x_m, d_m\}$ where $x^k \in \mathbb{R}^{n+1}$ is the k th input vector and $d^k \in \{-1, +1\}$, $k = 1, 2, \dots, m$, is the desired target for the k th input vector (usually, the order of these training pairs is random). The entire collection of these pairs is called the training set. The goal, then, is to design a perceptron such that for each input vector x_k of the training set, the perceptron output y_k matches the desired target d_k ; that is, we require $y_k = \text{sgn}(w^T x_k) = d_k$ for each $k = 1, 2, \dots, m$. In this case, we say that the perceptron correctly classifies the training set. Of course, "designing" an appropriate perceptron to correctly classify the training set amounts to determining a weight vector w^* such that the following relations are satisfied:

$$\begin{cases} (x^k)^T w^* > 0 & \text{if } d^k = +1 \\ (x^k)^T w^* < 0 & \text{if } d^k = -1 \end{cases} \quad (1)$$

Recall that the set of all x which satisfy $x^T w^* = 0$ defines a hyperplane in \mathbb{R}^n . Thus, in the context of the above discussion, finding a solution vector w^* to Eqn. (32) is equivalent to finding a separating hyperplane which correctly classifies all vectors x_k , $k = 1, 2, \dots, m$. In other words, we desire a hyperplane $x^T w^* = 0$ which partitions the input space into two distinct regions, one containing all points x_k with $d_k = +1$ and the other region containing all points x_k with $d_k = -1$.

One possible incremental method for arriving at a solution w^* is to invoke the perceptron learning rule (Rosenblatt, 1962):

$$\begin{cases} w^1 & \text{is set arbitrarily} \\ w^{k+1} = w^k + \rho(d^k - y^k)x^k & k = 1, 2, \dots \end{cases} \quad (2)$$

where ρ is a positive constant, called the learning rate. The incremental learning process given in Eqn. (2) proceeds as follows. First, an initial weight vector w_1 is selected (usually at random) to begin the process. Then the m pairs $\{x_k, d_k\}$ of the training set are used to successively update the weight vector until (hopefully) a solution w^* is found which correctly classifies the training set. This process of sequentially presenting the training patterns is usually referred to as "cycling" through the training set, and a complete presentation of the m training pairs is referred to as a cycle (or pass) through the training set. In general, more than one cycle through the training set is required to determine an appropriate solution vector. Hence, in Eqn. (2), k in w_k refers to the iteration number. On the other hand, k in x_k (and d_k) is the label of the training pair presented at the k th iteration. To be more precise, if the

number of training pairs, m , is finite, then the superscripts in x_k and d_k should be replaced by $(k \sim 1) \bmod m + 1$. Here, $a \bmod b$ returns the remainder of the division of a by b (e.g., $5 \bmod 8 = 5$, and $19 \bmod 8 = 3$). This observation is valid for all incremental learning rules presented here.

Notice that for $\eta = 0.5$, the perceptron learning rule can be written as:

$$\begin{cases} w^1 \text{ is set arbitrarily} \\ w^{k+1} = w^k + z^k & \text{if } (z^k)^T w^k \leq 0 \\ w^{k+1} = w^k & \text{otherwise} \end{cases} \quad (3)$$

where

$$z^k = \begin{cases} +x^k & \text{if } d^k = +1 \\ -x^k & \text{if } d^k = -1 \end{cases} \quad (4)$$

That is, a correction is done if and only if a misclassification, indicated by

$$(z^k)^T w^k \leq 0 \quad (5)$$

occurs. The addition of vector z_k to w_k in Eqn. (3) moves the weight vector directly toward and perhaps across the hyperplane $(z^k)^T w^k = 0$. The new inner product $(z^k)^T w^{k+1}$ is larger than $(z^k)^T w^k$ by the amount of $\|z^k\|^2$ and the correction $w_k = w_k + \tilde{1} w_k$ is clearly moving w_k in a good direction, the direction of increasing $(z^k)^T w^k$, as can be seen from Fig. 22. Thus, the perceptron learning rule attempts to find a solution w^* for the following system of inequalities

$$(z^k)^T w > 0 \text{ for } k = 1, 2, \dots, m \quad (6)$$

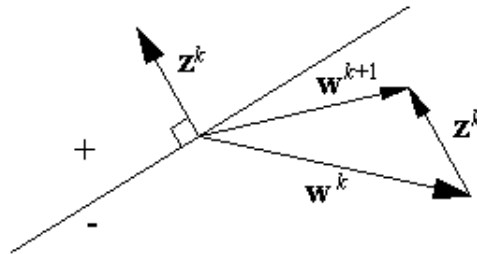


Fig. 22: Geometric representation of the perceptron learning rule with $\eta = 0.5$.

In an analysis of any learning algorithm, there are two main issues to consider:

1. The existence of solutions and
2. Convergence of the algorithm to the desired solutions (if they exist).

In the case of the perceptron, it is clear that a solution vector (i.e., a vector w^* which correctly classifies the training set) exists if and only if the given training set is linearly separable. Assuming, then, that the training set is linearly separable, we may proceed to show that the perceptron learning rule converges to a solution (Novikoff, 1962; Ridgway, 1962; Nilsson, 1965) as follows. Let w^* be any solution vector, so that

$$(z^k)^T w^* > 0 \text{ for } k = 1, 2, \dots, m \quad (7)$$

Then, from Eqn. (3), if the k th pattern is misclassified we may write

$$w^k + \tilde{I}w^* = w^k - w^* + z^k \quad (8)$$

Where k is a positive scale factor, and hence

$$\|w^{k+1} - \alpha w^*\|^2 = \|w^k - \alpha w^*\|^2 + 2(z^k)^T (w^k - \alpha w^*) + \|z^k\|^2 \quad (9)$$

Since z^k is misclassified, we have $(z^k)^T w^k \leq 0$, and thus

$$\|w^{k+1} - \alpha w^*\|^2 \leq \|w^k - \alpha w^*\|^2 - 2\alpha(z^k)^T w^* + \|z^k\|^2 \quad (10)$$

Now, let $\beta^2 = \max_i \|z^i\|^2$ and $\gamma = \min_i (z^i)^T w^*$ (is positive since $(z^i)^T w^* > 0$) and substitute into Eqn. (10) to get

$$\|w^{k+1} - \alpha w^*\|^2 \leq \|w^k - \alpha w^*\|^2 - 2\alpha\gamma + \beta^2 \quad (11)$$

If we choose sufficiently large, in particular $\alpha = \frac{\beta^2}{\gamma}$, we obtain

$$\|w^{k+1} - \alpha w^*\|^2 \leq \|w^k - \alpha w^*\|^2 - \beta^2 \quad (12)$$

Thus, the square distance between w^k and w^* is reduced by at least β^2 at each correction, and after k corrections we may write Eqn. (12) as

$$0 \leq \|w^{k+1} - \alpha w^*\|^2 \leq \|w^1 - \alpha w^*\|^2 - k\beta^2 \quad (13)$$

It follows that the sequence of corrections must terminate after no more than k_0 corrections, where

$$k_0 = \frac{\|w^1 - \alpha w^*\|^2}{\beta^2} \quad (14)$$

Therefore, if a solution exists, it is achieved in a finite number of iterations. When corrections cease, the resulting weight vector must classify all the samples correctly since a correction occurs whenever a sample is misclassified and since each sample appears infinitely often in the sequence. In general, a linearly separable problem admits an infinite number of solutions. The perceptron learning rule in Eqn. (2) converges to one of these solutions. This solution, though, is sensitive to the value of the learning rate, used and to the order of presentation of the training pairs. This sensitivity is responsible for the varying quality of the perceptron generated separating surface observed in simulations.

The bound on the number of corrections, k_0 , given by Eqn. (14) depends on the choice of the initial weight vector w_1 . If $w_1 = 0$, we get

$$k_0 = \frac{\alpha^2 \|w^*\|^2}{\beta^2} = \frac{\beta^2 \|w^*\|^2}{\gamma^2}$$

or

$$k_0 = \frac{\alpha^2 \|W^*\|^2}{\beta^2} = \frac{\beta^2 \|w^*\|^2}{\gamma^2} \quad (15)$$

Here, k_0 is a function of the initially unknown solution weight vector w^* .

Therefore, Eqn. (15) is of no help for predicting the maximum number of corrections. However, the denominator of Eqn. (15) implies that the difficulty of the problem is essentially determined by the samples most nearly orthogonal to the solution vector.

ii) **Generalizations of the Perceptron Learning Rule**

The perceptron learning rule may be generalized to include a variable increment ρ^k and a fixed, positive margin b . This generalized learning rule updates the weight vector whenever $(z^k)^T w^k$ fails to exceed the margin b . Here, the algorithm for weight vector update is given by

$$\begin{cases} w^1 & \text{arbitrary} \\ w^{k+1} = w^k + \rho^k z^k & \text{if } (z^k)^T w^k \leq b \\ w^{k+1} = w^k & \text{otherwise} \end{cases} \quad (16)$$

The margin b is useful because it gives a dead-zone robustness to the decision boundary. That is, the perceptron's decision hyperplane is constrained to lie in a region between the two classes such that sufficient clearance is realized between this hyperplane and the extreme points (boundary patterns) of the training set. This makes the perceptron robust with respect to noisy inputs. It can be shown (Duda and Hart, 1973) that if the training set is linearly separable and if the following three conditions are satisfied:

$$1. \quad \rho^k \geq 0 \quad (17)$$

$$2. \quad \lim_{m \rightarrow \infty} \sum_{k=1}^m \rho^k = \infty \quad (18)$$

$$3. \quad \lim_{m \rightarrow \infty} \frac{\sum_{k=1}^m (\rho^k)^2}{\left(\sum_{k=1}^m \rho^k \right)^2} = 0 \quad (19)$$

(e.g., $\rho^k = \frac{\rho}{k}$ or even $\rho^k = k$) then w_k converges to a solution w^* which satisfies $(z^i)^T w^* > b$ for $i = 1, 2, \dots, m$. Furthermore, when ρ^k is fixed at a positive constant, this learning rule converges in finite time.

Another variant of the perceptron learning rule is given by the "batch" update procedure

$$\begin{cases} w^1 & \text{arbitrary} \\ w^{k+1} = w^k + \rho \sum_{z \in Z(w^k)} z \end{cases} \quad (20)$$

where $Z(w^k)$ is the set of patterns z misclassified by w^k . Here, the weight vector change $\Delta w = w^{k+1} - w^k$ is along the direction of the resultant vector of all misclassified patterns. In general, this update procedure converges faster than the perceptron rule, but it requires more storage.

In the nonlinearly separable case, the above algorithms do not converge. Few theoretical results are available on the behavior of these algorithms for nonlinearly separable problems [see Minsky and Papert (1969) and Block and Levin (1970) for some preliminary results]. For example, it is known that the length of w in the perceptron rule is bounded, i.e., tends to fluctuate near some limiting value $\|w^*\|$ (Efron, 1964). This information may be used to terminate the search for w^* . Another approach is to average the weight vectors near the fluctuation point w^* . Butz (1967) proposed the use of a reinforcement factor, $0 < \alpha < 1$, in the perceptron learning rule. This reinforcement places w in a region that tends to minimize the probability of error for nonlinearly separable cases. Butz's rule is as follows

$$\begin{cases} w^1 & \text{arbitrary} \\ w^{k+1} = w^k + \rho z^k & \text{if } (z^k)^T w \leq 0 \\ w^{k+1} = w^k + \rho \gamma z^k & \text{if } (z^k)^T w > 0 \end{cases} \quad (21)$$

iii) The Perceptron Criterion Function

It is interesting to see how the above error correction rules can be derived by a gradient descent on an appropriate criterion (objective) function. For the perceptron, we may define the following criterion function (Duda and Hart, 1973):

$$J(w) = - \sum_{z \in Z(w)} z^T w \quad (22)$$

where $Z(w)$ is the set of samples misclassified by w (i.e., $z^T w < 0$). Note that if $Z(w)$ is empty then $J(w) = 0$, otherwise $J(w) > 0$. Geometrically, $J(w)$ is proportional to the sum of the distances from the misclassified samples to the decision boundary. The smaller J is, the better the weight vector w will be.

Given this objective function $J(w)$, we can incrementally improve the search point w^k at each iteration by sliding downhill on the surface defined by $J(w)$ in w space. Specifically, we may use J to perform a discrete gradient descent search, which updates w^k so that a step is taken downhill in the "steepest" direction along the search surface $J(w)$ at w^k . This can be achieved by making w^k proportional to the gradient of J at the present location w^k , formally we may write

$$w^{k+1} = w^k - \rho \nabla J(w) \Big|_{w=w^k} = w^k - \rho \left[\frac{\partial J}{\partial w_1} \quad \frac{\partial J}{\partial w_2} \quad \dots \quad \frac{\partial J}{\partial w_{n+1}} \right]^T \Big|_{w=w^k} \quad (23)$$

Here, the initial search point, w^1 , and the learning rate (step size) are to be specified by the user. We will refer to Eqn. (23) as the steepest gradient descent search rule or, simply, gradient descent. Next, substituting the gradient

$$\nabla J(w^k) = - \sum_{z \in Z(w^k)} z \quad (24)$$

in Eqn. (20), leads to the weight update rule

$$w^{k+1} = w^k + \rho \sum_{z \in Z(w^k)} z \quad (25)$$

The learning rule given in Eqn. (25) is identical to the multiple-sample (batch) perceptron rule of Eqn. (20). The original perceptron learning rule of Eqn. (3) can be thought of as an "incremental" gradient descent search rule for minimizing the perceptron criterion function in Eqn. (22). Following a similar procedure as in Eqns. (23) through (24), it can be shown that

$$J(w) = - \sum_{z^T w \leq b} (z^T w - b) \quad (26)$$

is the appropriate criterion function for the modified perceptron rule.

Here, it may be noted that the gradient of J in Eqn. (24) is not mathematically precise. Due to the piecewise linear nature of J , sudden changes in the gradient of J occur every time the perceptron output y goes through a transition at $(z^k)^T w = 0$. Therefore, the gradient of J is not defined at "transition" points w satisfying $(z^k)^T w = 0$, $k = 1, 2, \dots, m$. However, because of the discrete nature of Eqn. (23), the

likelihood of w^k overlapping with one of these transition points is negligible, and thus we may still express J as in Eqn. (24).

iv) Mays Learning Rule

The criterion functions in Eqns. (22) and (26) are by no means the only functions we can construct that are minimized when w is a solution vector. For example, an alternative function is the quadratic function

$$J(w) = \frac{1}{2} \sum_{z^T w \leq b} (z^T w - b)^2 \tag{27}$$

where b is a positive constant margin. Like the previous criterion functions, the function $J(w)$ in Eqn. (27) focuses attention on the misclassified samples. Its major difference is that its gradient is continuous, whereas the gradient of the perceptron criterion function, with or without the use of margin, is not. Unfortunately, the present function can be dominated by the input vectors with the largest magnitudes. We may eliminate this undesirable effect by dividing by $\|z\|^2$:

$$J(w) = \frac{1}{2} \sum_{z^T w \leq b} \frac{(z^T w - b)^2}{\|z\|^2} \tag{28}$$

The gradient of $J(w)$ in Eqn (28) is given by

$$\nabla J(w) = \sum_{z^T w \leq b} \frac{z^T w - b}{\|z\|^2} z \tag{29}$$

which, upon substituting in Eqn. (23), leads to the following learning rule

$$\begin{cases} w^1 & \text{arbitrary} \\ w^{k+1} & = w^k + \rho \sum_{z^T w \leq b} \frac{b - z^T w^k}{\|z\|^2} z \end{cases} \tag{30}$$

If we consider the incremental update version of Eqn. (30), we arrive at Mays' rule (Mays, 1964):

$$\begin{cases} w^1 & \text{arbitrary} \\ w^{k+1} & = w^k + \rho \frac{b - (z^k)^T w^k}{\|z^k\|^2} z^k & \text{if } (z^k)^T w^k \leq b \\ w^{k+1} & = w^k & \text{otherwise} \end{cases} \tag{31}$$

If the training set is linearly separable, Mays' rule converges in a finite number of iterations, for $0 < \rho < 2$ (Duda and Hart, 1973). In the case of a nonlinearly separable training set, the training procedure in Eqn. (31) will never converge. To fix this problem a decreasing learning rate such as $\rho = \frac{\rho_0}{k}$ may be used to force convergence to some approximate separating surface (Duda and Singleton, 1964).

v) Widrow-Hoff (-LMS) Learning Rule

Another example of an error correcting rule with a quadratic criterion function is the Widrow-Hoff rule (Widrow and Hoff, 1960). This rule was originally used to train the linear unit, also known as the adaptive linear combiner element (ADALINE), shown in Fig. 23. In this case, the output of the linear unit in response to the input x_k is simply $y_k = (x_k)^T w$. The Widrow-Hoff rule was originally proposed as an ad hoc rule, which embodies the so-called minimal disturbance principle. Later, it was

discovered (Widrow and Stearns, 1985) that this rule converges in the mean square to the solution w^* which corresponds to the least-mean-square (LMS) output error, if all input patterns are of the same length (i.e., $\|x_k\|$ is the same for all k). Therefore, this rule is sometimes referred to as the -LMS rule.

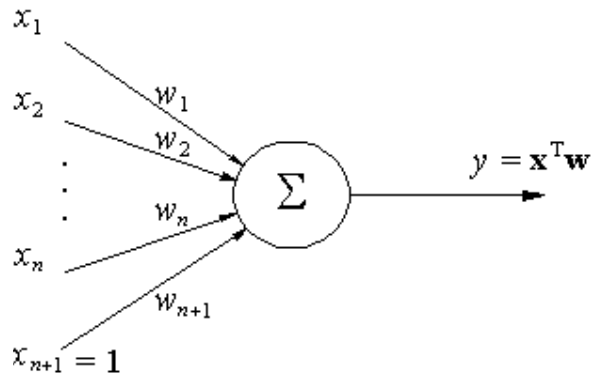


Fig. 23: Adaptive linear combiner element (ADALINE).

The -LMS rule is given by

$$\begin{cases} w^1 = 0 \text{ or arbitrary} \\ w^{k+1} = w^k + \alpha(d^k - y^k) \frac{x^k}{\|x^k\|^2} \end{cases} \quad (32)$$

where d^k is the desired response, and $\alpha > 0$. Eqn. (32) is similar to the perceptron rule if we set, in Eqn. (2), as

$$\rho = \rho^k \frac{\alpha}{\|x^k\|^2} \quad (33)$$

Though, the error in Eqn. (32) is measured at the linear output; not after the nonlinearity as in the perceptron. The constant ρ controls the stability and speed of convergence (Widrow and Stearns, 1985; Widrow and Lehr, 1990). If the input vectors are independent over time, stability is insured for most practical purposes.

As for May's rule, this rule is self-normalizing in the sense that the choice does not depend on the magnitude of the input vectors. Since the -LMS rule selects w^k to be collinear with x^k , the desired error correction is achieved with a weight change of the smallest possible magnitude. Thus, when adapting to learn a new training sample, the responses to previous training samples are minimally disturbed, on the average. This is the basic idea behind the minimal disturbance principle on which the -LMS is founded. Alternatively, one can show that the -LMS learning rule is a gradient descent minimizer of an appropriate quadratic criterion function.

Try the following exercises.

E7) Define error-correction learning.

E8) Differentiate supervised and unsupervised learning by citing examples of each.

4.6 SUMMARY

In this unit, we have covered the following points.

1. Mathematical analysis has solved some of the mysteries posed by the new models but has left many questions open for future investigations. Needless to say, the study of neurons, their interconnections, and their role as the brain's elementary building blocks is one of the most dynamic and important research fields in modern biology.
2. The simplest kind of computing units are used to build artificial neural networks. These computing elements are a generalization of the common logic gates used in conventional computing and, since they operate by comparing their total input with a threshold, this field of research is known as threshold logic.
3. A simple static synchronous associative memory is presented along with appropriate memory storage recipes. Then, this simple associative memory is extended into a recurrent auto associative memory by employing feedback.
4. Error correction rules were initially proposed as ad hoc rules for single unit training. These rules essentially drive the output error of a given unit to zero. We start with the classical perceptron learning rule and give a proof for its convergence.

4.7 SOLUTIONS/ANSWERS

- E1) The elements of a neuron are dendrites, synapses, cell body and axis.
- E2) The five models of computations are given below
- i) The mathematical model
 - ii) The logic operational model
 - iii) The computer model
 - iv) Cellular automate
 - v) The biological model.
- E3) The activation function (also known as the transfer function) performs the task of axon and synapse. The output of a neuron largely depends on its activation function. Different types of activation function are in use, such as hard-limit, linear log sigmoid, tan sigmoid and others.
- E4) $\phi = \alpha_4 f_2 + \alpha_5 f_3$
 $f_2 = \alpha_1 f_1$
 $f_3 = f_1 \alpha_2 + f_2 \alpha_3$
- E5) A biological neuron model is a mathematical description of the properties of nerve cells, or neurons, that is designed to accurately describe and predict biological processes. This is in contrast to the artificial neuron, which arise for computational effectiveness, although these goals sometimes overlap.
- E6) $x_1 \wedge x_2$
 $x_1 \vee x_2$
 $\neg x_1$ or $\neg x_2$

E8) Supervised networks are a little more straightforward to conceptualize than unsupervised networks. We can apply the inputs to the supervised network along with an expected response, much like the Pavlovian conditioned stimulus and response regimen. We can mold the network with stimulus-response pairs. A stock market forecaster may present economic data (the stimulus) along with metrics of stock market performance (the response) to the neural network to the presenter and attempt to predict the future once training is complete.

We provide unsupervised networks with only stimulus. For example, we want an unsupervised network to correctly classify parts from a conveyor belt into part numbers, providing an image of each part to do the classification (the stimulus). The unsupervised network in this case would act like a look-up memory that is indexed by its contents, or a Content-Addressable-Memory (CAM).

4.8 REFERENCES

1. Simon S Haykin and Simon Haykin, (1998), *Neural Networks: A Comprehensive Foundation*, Pearson Education.
2. Raul Rojas, (1996), *Neural Networks: A Systematic Introduction*.
3. S. Rajasekaran and G.A. Vijayalakshmi Pai, (2010), *Neural Networks: Fuzzy Logic, and Genetic Algorithms*.
4. D.K. Pratihar, (2008), *Soft Computing*.
5. Valluru B. Rao and Hayagriva V. Rao, *C++ Neural Networks & Fuzzy Logic*.