NEURAL NETWORKS

1.0.0 Introduction

The recent rise of interest in neural networks has its roots in the recognition that the brain performs computations in a different manner than do conventional digital computers. Computers are extremely fast and precise at executing sequences of instructions that have been formulated for them. A human information process sing system is composed of neurons switching at speeds about a million times slower than computer gates. Yet, humans are more efficient than computers at computationally complex tasks such as speech understanding. Moreover, not only humans, but also even animals, can process visual information better than the fastest computers.

Artificial neural systems, or neural networks (NN), are physical cellular systems, which can acquire, store, and utilize experiential knowledge. The knowledge is in the form of stable states or mappings embedded in networks that can be recalled in response to the presentation cues. Neural network processing typically involves dealing with large-scale problems in terms of dimensionality, amount of data handled, and the volume of simulation or neural hardware processing. This large-scale approach is both essential and typical for real-life applications. By keeping view of all these, the research community has made an effort in designing and implementing the various neural network models for different applications. Now let us formally define the basic idea of neural network:

Definition: ***A neural network is a computing system made up of a number of simple, highly interconnected nodes or processing elements, which process information by its dynamic state response to external inputs***.

1.1.0 Humans and Computers

Human beings are more intelligent than computers. Computers could only do logical things well. But in case of solving cross word puzzles, vision problem, controlling an arm to pick it up or something similar, that requires exceptionally complex techniques. Like these problems, human beings do better than computers.

Computers are designed to carry out one instruction after another, extremely rapid, whereas our brains work with many slower units. Whereas a computer can typically carry out a few million operations every second, the units in the brain respond about ten per second. However, they work on many different things at once, which computer can't do.

The computer is a high-speed, serial machine and is used as such, compared to the slow, highly parallel nature of the brain. Counting is an essentially serial activity, as is

adding, with the thing done one after another, and so the computer can beat the brain any time. For vision, or speech recognition, the problem is a highly parallel one, with many different and conflicting inputs, triggering many different and conflicting ideas and memories, and it is only the combining of all these different factors that allow us to perform such feats, but then, our brains are able to operate in parallel easily and so we leave the computers far behind.

The conclusion that we can reach from all of this is that the problems that we are trying to solve are immensely parallel ones.

1.2.0 History of artificial neural networks

- The field of neural networks is not new. The first formal definition of a synthetic neuron model based on the highly simplified considerations of the biological model proposed by McCulloch and Pitts in 1943. The McCulloch-Pitts (MP) neuron model resembles what is known as a binary logic device.
- The next major development, after the MP neuron model was proposed, occurred in 1949, when D.O. Hebb proposed a learning mechanism for the brain that become the starting point for artificial neural networks (ANN) learning (training) algorithms. He postulated that as the brain learns, it changes its connectivity patterns.
- The idea of learning mechanism was first incorporated in ANN by E. Rosenblatt 1958.
- By introducing the least mean squares (LMS) learning algorithm, Widrow and Hoff developed in 1960 a model of a neuron that learned quickly and accurately. This model was called ADALINE for ADAptive LInear NEuron. The applications of ADALINE and its extension to MADALINE (for Many ADALINES) include pattern recognition, weather forecasting, and adaptive controls. The monograph on learning machines by Nils Nilsson (1965) summarized the developments of that time.
- In 1969, research in the field of ANN suffered a serious setback. Minsky and Papert published a book on perceptrons in which they proved that single layer neural networks have limitations in their abilities to process data, and are capable of any mapping that is linearly separable. They pointed out, carefully applying mathematical techniques, that are logical Exclusive-OR (XOR) function could not be realized by perceptrons.
- Further, Minsky and Papert argued that research into multi-layer neural networks would be unproductive. Due to this pessimistic view of Minsky and Papert, the field

of ANN entered into an almost total eclipse for nearly two decades. Fortunately, Minsky and Papert's judgment has been disapproved; multi-layer perceptron networks can solve all nonlinear separable problems.

- Nevertheless, a few dedicated researchers such as Kohonen, Grossberg, Anderson and Hopfield continued their efforts.
- The study of learning in networks of threshold elements and of the mathematical theory of neural networks was pursued by Sun - Ichi – Amari (1972, 1977). Also Kunihiko Fukushima developed a class of neural network architectures known as neocognitrons in 1980.
- There have been many impressive demonstrations of ANN capabilities: a network has been trained to convert text to phonetic representations, which were then converted to speech by other means (Sejnowsky and Rosenberg 1987); other network can recognize handwritten characters (Burr 1987); and a neural network based image-compression system has been devised (Cottrell, Munro, and Zipser 1987). These all use the backpropagation network, perhaps the most successful of the current algorithms. Backpropagation, invented independently in three separate research efforts (Werbos 1974, Parker 1982, and Rumelhart, Hinton and Williams 1986) provides a systematic means for training multi-layer networks, thereby overcoming limitations presented by Minsky.

1.3.0 Characteristics of ANN

Artificial neural networks are biologically inspired; that is, they are composed of elements that perform in a manner that is analogous to the most elementary functions of the biological neuron. The important characteristics of artificial neural networks are learning from experience, generalize from previous examples to new ones, and abstract essential characteristics from inputs containing irrelevant data.

1.3.1 Learning

The NNs learn by examples. Thus, NN architectures can be 'trained' with known examples of a problem before they are tested for their 'inference' capability on unknown instances of the problem. They can, therefore, identify new objects previously untrained. ANN can modify their behavior in response to their environment. Shown a set of inputs (perhaps with desired outputs), they self-adjust to produce consistent responses. A wide variety of training algorithms has been discussed in later units.

1.3.2 Parallel operation

The NNs can process information in parallel, at high speed, and in a distributed manner.

1.3.3 Mapping

The NNs exhibit mapping capabilities, that is, they can map input patterns to their associated output patterns.

1.3.4 Generalization

The NNs possess the capability to generalize. Thus, they can predict new outcomes from past trends. Once trained, a network's response can be to a degree, insensitive to minor variations in its input. This ability to see through noise and distortion to the pattern that lies within is vital to pattern recognition in a real-world environment. It is important to note that the ANN generalizes automatically as a result of its structure, not by using human intelligence embedded in the form of adhoc computer programs.

1.3.5 Robust

The NNs are robust systems and are fault tolerant. They can, therefore, recall full patterns from incomplete, partial or noisy patterns

1.3.6 Abstraction

Some ANN's are capable of abstracting the essence of a set of inputs. i.e. they can extract features of the given set of data, for example, convolution neural networks are used to extract different features from images like edges, dark spots, shapes ..etc. Such networks are trained for feature patterns based on which they can classify or cluster the given input set.

1.3.7 Applicability

ANN's are not a panacea. They are clearly unsuited to such tasks as calculating the payroll. They are preferred for a large class of pattern-recognition tasks that conventional computers do poorly, if at all.

1.4.0 Applications

Neural networks are preferred when the task is related to large-amount data processing. The following are the potential applications of neural networks:

- Classification

- Prediction
- Data Association
- Data Conceptualization
- Data Filtering
- Optimization

In addition to the above fields, neural networks can apply to the fields of Medicine, Commercial and Engineering, etc.

2.1.0 The Biological Prototype

ANN's are biologically inspired; that is viewing at the organization of the brain considering network configurations and algorithms. The human nerve system, built of cells called neurons is of staggering complexity. It contains approximately ten thousand million (10^{11}) basic neurons. Each of these neurons is connected to about ten thousand (10^4) others. The connection of each neuron with other neurons forms a densely network called a neural network. These massive interconnections provide an exceptionally large computing power and memory. The neuron accepts many inputs, which are all added up in some fashion. If enough active inputs are received at once, then the neuron will be activated at once, then the neuron will be activated and "fire"; if not, then the neuron will remain in its inactive, quit state. The schematic diagram of biological neuron is shown in Fig.2.1. From a systems theory, the neuron considered to be as a multiple-input-single-output (MISO) system as shown in Fig.2.2.

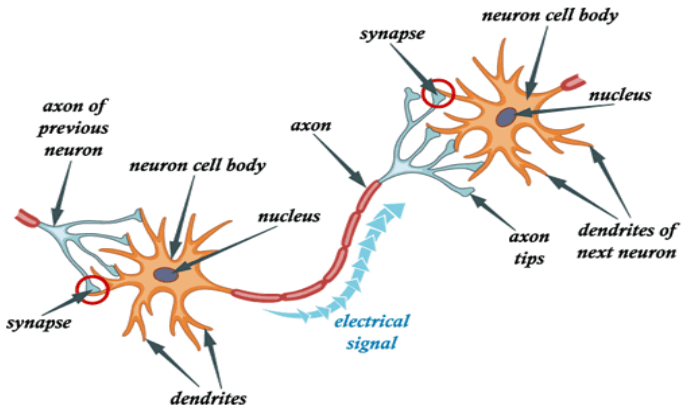


Fig. 2.1. A Schematic view of the biological neuron

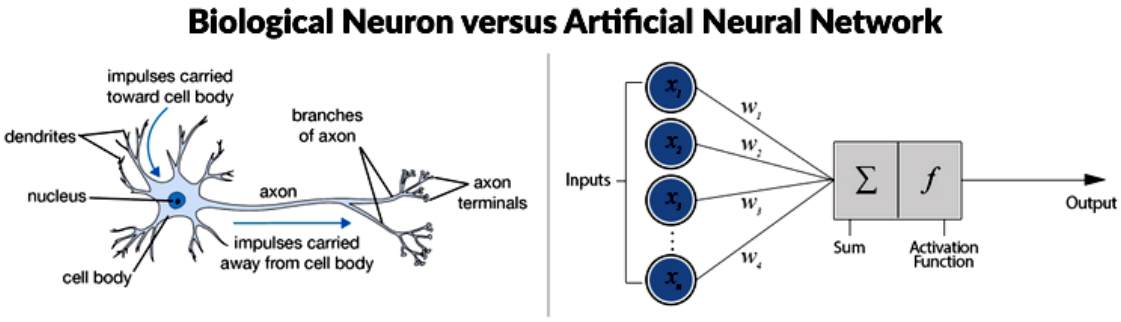


Fig.2.2 Model representation of a biological neuron with multiple inputs

Human	Artificial
Neuron	Processing Element
Dendrites	Combining Function
Cell Body	Transfer Function
Axons	Element Output
Synapses	Weights

The soma is the body of the neuron. Attached to the soma there are long irregularly shaped filaments, called dendrites. These nerve processes are often less than a micron in diameter, and have complex branching shapes. The dendrites act as the connections through which all the inputs to the neuron arrive. These cells are able to perform more complex functions than simple addition on the inputs they receive, but considering simple summation is a reasonable approximation.

Another type of nerve process attached to the soma is called an axon. This is electrically active, unlike the dendrite, and serves as the output channel of the neuron. Axons always appear on output cell, but are often absent from interconnections, which have both inputs and outputs on dendrites. The axon is a non-linear threshold device, producing a voltage pulse, called an action potential, that last about 1 millisecond (10^{-3} sec) when the resting potential within the soma rises above a certain critical threshold.

The axon terminates in a specialized contact called a synapse that couples the axon with the dendrite of another cell. There is no direct linkage across the junction; rather, it is temporally chemical one. The synapse releases chemicals called neurotransmitters when its potential is raised sufficiently by the action potential. It may take the arrival of more than one action potential before the synapse is triggered. The neurotransmitters that are released by the synapse diffuse across the gap, any chemically activate gates on the dendrites, which, when open, allow charged ions to flow. It is this flow of ions that alters the dendrite potential, and provides a voltage pulse on the dendrite, which is then conducted along into the

next neuron body. Each dendrite may have many synapses acting on it, allowing massive interconnectivity to be achieved.

2.2.0 **Artificial Neuron**

The artificial neuron is developed to mimic the first-order characteristics of the biological neuron. In similar to the biological neuron, the artificial neuron receives many inputs representing the output of other neurons. Each input is multiplied by a corresponding weight, analogous to the synaptic strength. All of these weighted inputs are then summed and passed through an activation function to determine the neuron input. This artificial neuron model is shown in Fig.2.3.

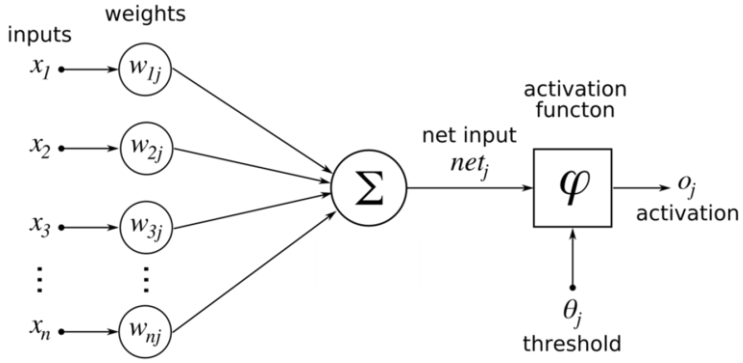


Fig. 2.3 An artificial neuron

The mathematical model of the artificial neuron may written as

$$u(t) = w_1x_1 + w_2x_2 + w_3x_3 + \ldots\ldots\ldots + w_nx_n$$

$$= \sum_{i=1}^{n} w_i x_i + \theta = \sum_{i=0}^{n} w_i x_i \qquad (2.1)$$

Assuming $w_0 = \theta$ and $x_0 = 1$

$$y(t) = f[u(t)] \qquad (2.2)$$

where f[.] is a nonlinear function called as the activation function, the input-output function or the transfer function. In equation (2.1) and Fig.2.3, $[x_0, x_1, \ldots, x_n]$ represent the inputs, $[w_0, w_1, \ldots . , w_n]$ represents the corresponding synaptic weights. In vector form, we can represent the neural inputs and the synaptic weights as

$X = [x_0, x_1, \ldots, x_n]^T$, and $W = [w_0, w_1, \ldots . , w_n]$

Equations (2.1) and (2.2) can be represented in vector form as:

$$U = WX \qquad (2.3)$$

$$Y = f[U] \qquad (2.4)$$

The activation function f[.] is chosen as a nonlinear function to emulate the nonlinear behavior of conduction current mechanism in biological neuron. The behavior of the artificial neuron depends both on the weights and the activation function. Sigmoidal functions are the

commonly used activation functions in multi-layer static neural networks. Other types of activation functions are discussed in later units.

2.0.0 McCulloch-Pitts Model

The McCulloch-Pitts model of the neuron is shown in Fig. 2.4(a). The inputs x_i, for i= 1,2, . . .,n are 0 or 1, depending on the absence or presence of the input impulse at instant k. The neuron's output signal is denoted as **Y.** The firing rule for this model is defined as follows

$$\mathbf{Y}^{k+1} = \begin{cases} 1 & if \quad \sum_{i=1}^{n} w_i x_i^k \geq T \\ 0 & if \quad \sum_{i=1}^{n} w_i x_i^k < T \end{cases} \qquad 1$$

where super script k = 0,1,2,. . . . , denotes the discrete – time instant, and w_i is the multiplicative weight connecting the i'th input with the neuron's membrane. Note that w_i = +1 for excitatory synapses, w_i = -1 for inhibitory synapses for this model, and T is the neuron's threshold value, which needs to be exceeded by the weighted sum of the signals for the neuron to fire.

This model can perform the basic operations NOT, OR and AND, provided its weights and thresholds are approximately selected. Any multivariable combinational function can be implemented using either the NOT and OR, or alternatively the NOT and AND, Boolean operations. Examples of three-input NOR and NAND gates using the McCulloch-Pitts neuron model are shown in (Fig.2.4 (b) and Fig 2.4(c)).

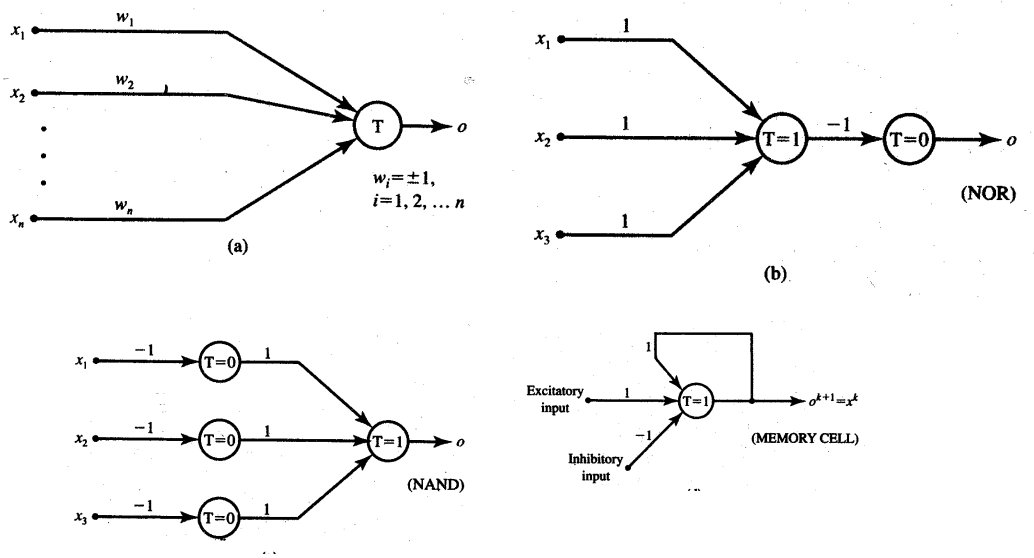


Fig.2.4 McCulloch-Pitts

2.1.0 **Keyword definitions**

Action potential: The pulse of electrical potential generated across the membrane of a neuron (or an axon) following the application of a stimulus greater than threshold value.

Axon: The output fiber of a neuron, which carries the information in the form of action potentials to other neurons in the network.

Dendrite: The input line of the neuron that carries a temporal summation of action potentials to the soma.

Excitatory neuron: A neuron that transmits an action potential that has excitatory (positive) influence on the recipient nerve cells.

Inhibitory neuron: A neuron that transmits an action potential that has inhibitory (negative) influence on the recipient nerve cells.

Lateral inhibition: The local spatial interaction where the neural activity generated by one neuron is suppressed by the activity of its neighbors.

Latency: The time between the application of the stimulus and the peak of the resulting action potential output.

Refractory period: The minimum time required for the axon to generate two consecutive action potentials.

Neural state: A neuron is active if it's firing a sequence of action potentials.

Neuron: The basic nerve cell for processing biological information.

Soma: The body of a neuron, which provides aggregation, thresholding and nonlinear activation to dendrite inputs.

Synapse: The junction point between the axon (of a pre-synaptic neuron) and the dendrite (of a post-synaptic neuron). This acts as a memory (storage) to the past-accumulated experience (knowledge).

Activation Functions

3.1.0 **Operations of Artificial Neuron**

The schematic diagram of artificial neuron is shown in Fig.3.1. The artificial neuron mainly performs two operations, one is the summing of weighted net input and the second is passing the net input through an activation function. The activation function also called nonlinear function and some time transfer function of artificial neuron.

The net input of j^{th} neuron may be written as

$NET_j = w_1x_1 + w_2x_2 + w_3x_3 + \ldots + w_nx_n + \theta_j$ (3.1)

where θ_j is the threshold of j^{th} neuron,

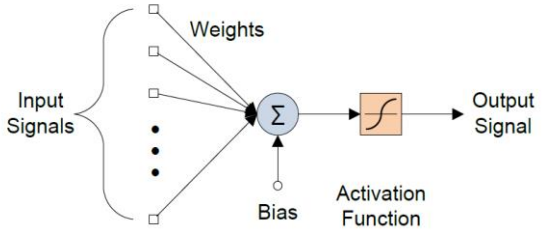


Fig. 3.1 Artificial neuron.

$X = [x_1\ x_2 \ldots x_n]$ in the input vector and $W = [w_1\ w_2 \ldots w_n]$ is the synaptic weight vector. The NET_j signal is processed by an activation function F to produce the neuron's output signal.

$$OUT_j = F(NET_j) \qquad (3.2)$$

What functional form for F(.) should be selected? Can it be – square root, log, e^x , x^3 and so on. Mathematicians and computer scientists, however, have found that, the sigmoid (S-shaped) function is more useful. In addition to this sigmoid function, there are number of other function are using in artificial neural networks. They are discussed in the next section.

3.2.0 **Types of activation functions**

The behavior of the artificial neuron depends both on the synaptic weights and the activation function. Sigmoid functions are the commonly used activation functions in multi-layered feed forward neural networks. Neurons with sigmoid functions bear a greater resemblance to the biological neurons than with other activation functions. The other feature of sigmoid function is that it is differentiable, and gives a continuous values output. Some of the popular activation functions are described below along with their other characteristics.

1. **Sigmoid function (Unipolar sigmoid)** . The characteristics of this function is shown in Fig.3.2 and its mathematical description is

$$y(x) = f(x) = \frac{1}{1+e^{-x}} \qquad (3.1)$$

and its range of signal is $0<y<1$.

The derivative of the above function is written as

$$y'(x) = f'(x) = f(x)\,(1-f(x)) \qquad (3.2)$$

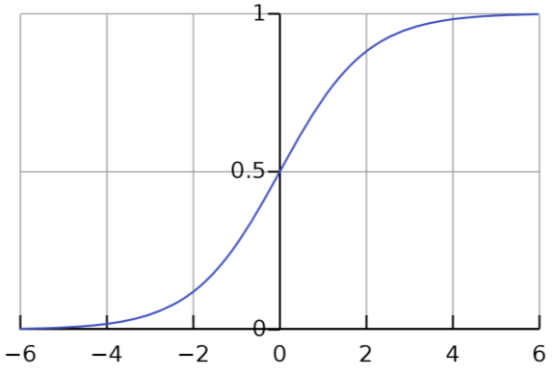


Fig.3.2 A sigmoid (S-shaped) function

Moreover, sigmoid functions are continuous and monatonic, and remain finite even as x approaches to ±∞. Because they are monatonic, they also provide for more efficient network training.

2. **Hyperbolic tangent (bipolar sigmoid) function**. The characteristics of this function is shown in Fig.3.3 and its mathematical description is

$$y(x)=f(x)=\tanh(x)=\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \qquad (3.3)$$

range of signal is -1<y<1 and its derivative can be obtained as

$$y' = f'(x) = 1-[f(x)]^2 \qquad (3.4)$$

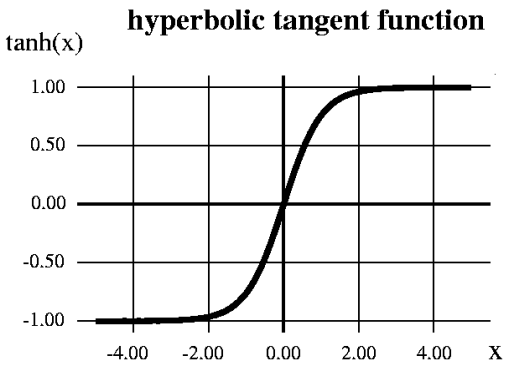


Fig.3.3 A hyperbolic tangent (bipolar sigmoid) function

3. **Radial basis function**: The Gaussian function is the most commonly used " radially symmetric" function, the characteristics of this function is shown in Fig.3.4 and its mathematical description is

$$y = f(x) = \exp\left(\frac{-x^2}{2}\right) \qquad (3.5)$$

Range of signal is 0<y<1 and its derivative can be obtained as

$$y' = f'(x) = -x\ \exp\left(\frac{-x^2}{2}\right) \qquad (3.6)$$

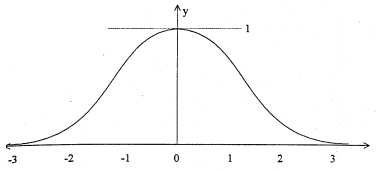


Fig.3.4. A Gaussian function

The function has maximum response, f(x) =1, when the input is x=0, and the response decreases to f(x)=0 as the input approaches x=±∞.

4. **Hard Limiter**: The hard limiter function is the mostly used in classification of patterns, the characteristics of this function is shown in Fig.3.5 and its mathematical description is

$$f(u(t))=\textbf{sign}\,(u(t)) = \begin{cases} +1, & u(t) > 0 \\ -1 & u(t) < 0 \end{cases} \qquad (3.7)$$

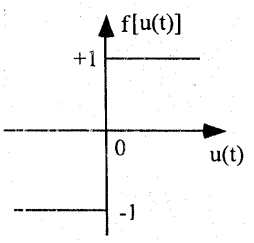


Fig. 3.5 Hard Limiter

This function is not differentiable. Therefore it cannot be used for continuation type of applications.

5. **Piecewise linear**: The piecewise linear function characteristics is shown in Fig.3.6 and its mathematical description is

$$f(u(t)) = \begin{cases} +1 & \text{if } gu > 1 \\ gu & \text{if } |gu| < 1 \\ -1 & \text{if } gu < -1 \end{cases} \qquad (3.8)$$

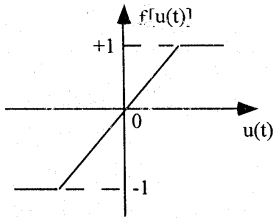


Fig. 3.6 Piecewise linear

6. **Linear**: The Linear function characteristics is shown in Fig.3.8 and its mathematical description is

$$f(u(t)) = gu(t) \qquad (3.10)$$

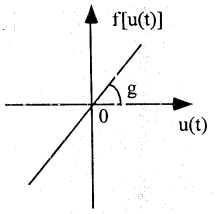


Fig. 3.8 Linear function

It is differentiable and is mostly used for output nodes of the networks.

3.3.0 **Selection of activation function**

The selection of an activation function is depends upon the application to which the neural network used and also the level (in which layer) neuron. The activation functions that are mainly used are the sigmoid (unipolar sigmoidal), the hyperbolic tangent (bipolar sigmoid), radial basis function, hard limiter and linear functions. The sigmoid and hyperbolic tangent functions perform well for the prediction and the process-forecasting types of problems. However, they do not perform as well for classification networks. Instead, the radial basis function proves more effective for those networks, and highly recommended function for any problems involving fault diagnosis and feature categorization. The hard limiter suits well for classification problems. The linear function may be used at output layer in feed forward networks.

Classification of Artificial Neural Networks

4.0.0 Introduction

The development of artificial neuron based on the understanding of the biological neural structure and learning mechanisms for required applications. This can be summarized as (a) development of neural models based on the understanding of biological neurons, (b) models of synaptic connections and structures, such as network topology and (c) the learning

rules. Researchers are explored different neural network architectures and used for various applications. Therefore the classification of artificial neural networks (ANN) can be done based on structures and based on type of data.

A single neuron can perform a simple pattern classification, but the power of neural computation comes from neuron connecting networks. The basic definition of artificial neural networks as physical cellular networks that are able to acquire, store and utilize experimental knowledge has been related to the network's capabilities and performance. The simplest network is a group of neurons are arranged in a layer. This configuration is known as single layer neural networks. There are two types of single layer networks namely, feed-forward and feedback networks. The single linear neural (that is activation function is linear) network will have very limited capabilities in solving nonlinear problems, such as classification etc., because their decision boundaries are linear. This can be made little more complex by selecting nonlinear neuron (that is activation function is nonlinear) in single layer neural network. The nonlinear classifiers will have complex shaped decision boundaries, which can solve complex problems. Even nonlinear neuron single layer networks will have limitations in classifying more close nonlinear classifications and fine control problems. In recent studies shows that the nonlinear neural networks in multi-layer structures can simulate more complicated systems, achieve smooth control, complex classifications and have capabilities beyond those of single layer networks. In this unit we discuss first classifications, single layer neural networks and multi-layer neural networks. The structure of a neural network refers to how its neurons are interconnected.

4.2.0 Applications

Having different types artificial neural networks, these networks can be used to broad classes of applications, such as (i) Pattern Recognition and Classification, (ii) Image Processing and Vision, (iii) System Identification and Control, and (iv) Signal Processing. The details of suitability of networks as follows:

(i) **Pattern Recognition and Classification**: Almost all networks can be used to solve these types of problems.

(ii) **Image Processing and Vision**: The following networks are used for the applications in this area: Static single layer networks, Dynamic single layer networks, BAM, ART, Counter – propagation networks, First – Order dynamic networks.

(iii) **System Identification and Control**: The following networks are used for the applications in this area: Static multi layer networks, Dynamic multi layer networks of types time-delay and Second-Order dynamic networks.

(iv) **Signal Processing:** The following networks are used for the applications in this area: Static multi layer networks of type RBF, Dynamic multi layer networks of types Cellular and Second-Order dynamic networks.

4.3.0 Single Layer Artificial Neural Networks

The simplest network is a group of neuron arranged in a layer. This configuration is known as single layer neural networks. This type of network comprises of two layers, namely the input layer and the output layer. The input layer neurons receive the input signals and the output layer neurons receive the output signals. The synaptic links carrying the weights connect every input neuron to the output neuron but not vice-versa. Such a network is said to'be *feedforward* in type or acyclic in nature. Despite the two layers, the network is termed single layer, since it is the output layer alone Which performs computation. The input layer merely transmits the signals to the output layer. Hence, the name single layer feedforward network. Figure 4.2 illustrates an example network.

There are two types of single layer networks namely, feed-forward and feedback networks.

4.3.1 **Feed forward single layer neural network**

Consider m numbers of neurons are arranged in a layer structure and each neuron receiving n inputs as shown in Fig.4.2.

Output and input vectors are respectively

$$O = [o_1 \; o_2 \; \ldots \; o_m]^T \qquad (4.1)$$

$$X = [x_1 \; x_2 \; \ldots \; x_n]^T$$

Weight w_{ji} connects the j^{th} neuron with the i^{th} input. Then the activation value for j^{th} neuron as

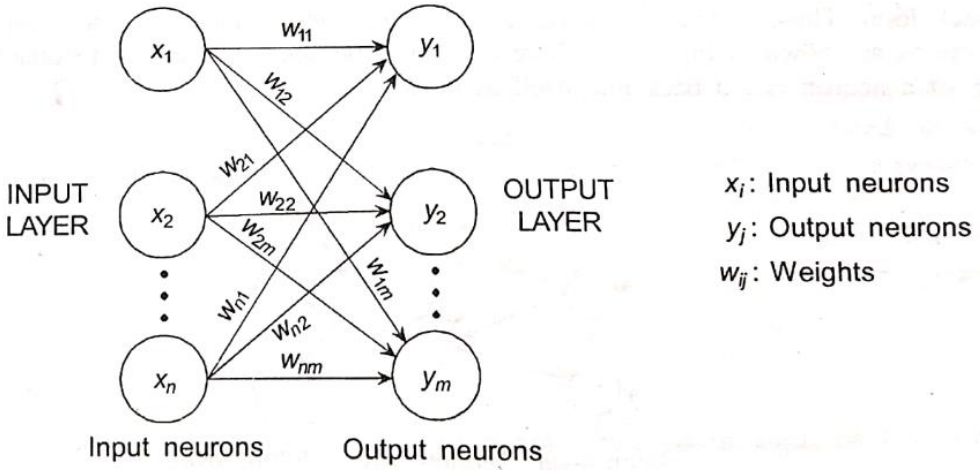
$$net_j = \sum_{i=1}^{n} w_{ji} x_i \qquad \text{for } j = 1,2, \ldots m \qquad (4.2)$$

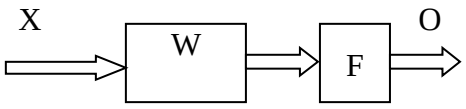
The following nonlinear transformation involving the activation function $f(net_j)$, for j=1,2,. . .m, completes the processing of X. The transformation will be done by each of the m neurons in the network.

$$o_j = f(W_j^t X), \qquad \text{for } j = 1, 2, \ldots m \qquad (4.3)$$

where weight vector w_j contains weights leading toward the j^{th} output node and is defined as follows

$$W_j = [\, w_{j1} \;\; w_{j2} \;\; \ldots \;\; w_{jn}] \qquad (4.4)$$





(b) Block diagram of single layer network

Fig. 4.2 Feed forward single layer neural network

Introducing the nonlinear matrix operator F, the mapping of input space X to output space O implemented by the network can be written as

$$O = F(W\,X) \qquad (4.5a)$$

Where W is the weight matrix and also known as connection matrix and is represented as

$$W = \begin{bmatrix} w_{11} & w_{12} & . & . & w_{1n} \\ w_{21} & w_{22} & . & . & w_{2n} \\ . & . & . & . & . \\ . & . & . & . & . \\ w_{m1} & w_{m2} & . & . & w_{mn} \end{bmatrix} \qquad (4.5b)$$

The weight matrix will be initialized and it should be finalized through appropriate training method.

$$F(.) = \begin{bmatrix} f(.) & 0 & . & . & 0 \\ 0 & f(.) & . & . & 0 \\ . & . & . & . & . \\ . & . & . & . & . \\ 0 & 0 & . & . & f(.) \end{bmatrix} \qquad (4.5c)$$

The nonlinear activation function *f*(.) on the diagonal of the matrix operator F(.) operates component-wise on the activation values net of each neuron. Each activation value is, in turn, a scalar product of an input with the respective weight vector, X is called input vector and O is called output vector. The mapping of an input to an output is shown as in (4.5) is of the feed-forward and instantaneous type, since it involves no delay between the input and the output . Therefore the relation (4.5a) may be written in terms of time t as

$$O(t) = F(W X(t)) \qquad (4.6)$$

This type of networks can be connected in cascade to create a multiplayer network. Though there is no feedback in the feedforward network while mapping from input X(t) to output O(t), the output values are compared with the "teachers" information, which provides the desired output values. The error signal is used for adapting the network weights. The details about will be discussed in later units.

Example: To illustrate the computation of output O(t), of the single layer feed forward network consider an input vector X(t) and a network weight matrix W (say initialized weights), given below. Consider the neurons uses the hard limiter as its activation function.

$$X = [-1 \;\; 1 \;\; -1]^T \qquad W = \begin{bmatrix} 1 & 0 & 1 \\ -1 & 0 & -2 \\ 0 & 1 & 0 \\ 0 & -1 & -3 \end{bmatrix}$$

The out vector may be obtained from the (4.5) as

$$O = F(W X) = [\, \text{sgn}(-1-1) \;\; \text{sgn}(+1+2) \;\; \text{sgn}(1) \;\; \text{sgn}(-1+3) \,]$$

$$= [\, -1 \;\; 1 \;\; 1 \;\; 1 \,]$$

The output vector of the above single layer feedforward network is = [-1 1 1 1].

4.4 Types of connections

There are three different options are available for connecting nodes to one another, as shown in Fig. 4.5 They are:

Intralayer connection: The output from a node fed into other nodes in the same layer.

Interlayer connection: The output from a node in one layer fed into nodes in other layer.

Recurrent connection: The outputs from a node fed into itself.

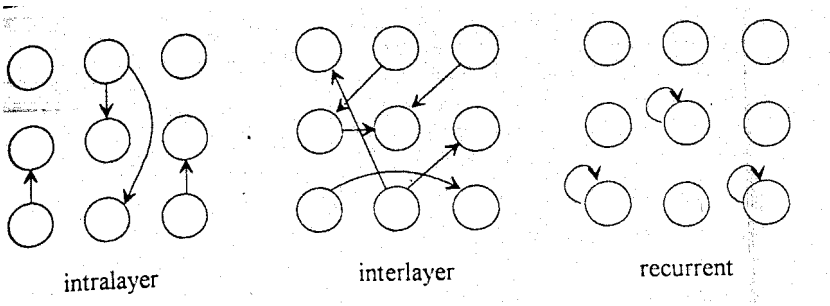


Fig. 4.5 Different connection option of Neural Networks

In general, when building a neural network its structure will be specified. In engineering applications, suitable and mostly preferred structure is the interlayer connection type topology. Within the interlayer connections, there are two more options: (i) feed forward connections and (ii) feedback connections, as shown in Fig. 4.6.

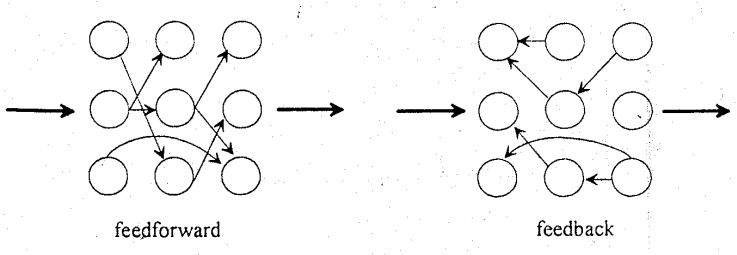


Fig. 4.6 Feed forward and feed back connections of Neural Networks

4.4.0 **Multi Layer Artificial Neural Networks**

Cascading a group of single layers networks can form the feed forward neural network. This type networks also known as feed forward multi layer neural network. In which, the output of one layer provides an input to the subsequent layer. The input layer gets input from outside; the output of input layer is connected to the first hidden layer as input. The output layer receives its input from the last hidden layer. The multi layer neural network provides no increase in computational power over single layer neural networks unless there is a nonlinear activation function between layers. Therefore, due to nonlinear activation function of each neuron in hidden layers, the multi layer neural networks able to solve many of the complex problems, such as, nonlinear function approximation, learning generalization, nonlinear classification etc.

A multi layer neural network consists of input layer, output layer and hidden layers. The number of nodes in input layer depends on the number of inputs and the number of nodes in the output layer depends upon the number of outputs. The designer selects the number of hidden layers and neurons in respective layers. According to the **Kolmogorov's** theorem single hidden layer is sufficient to map any complicated input – output mapping.

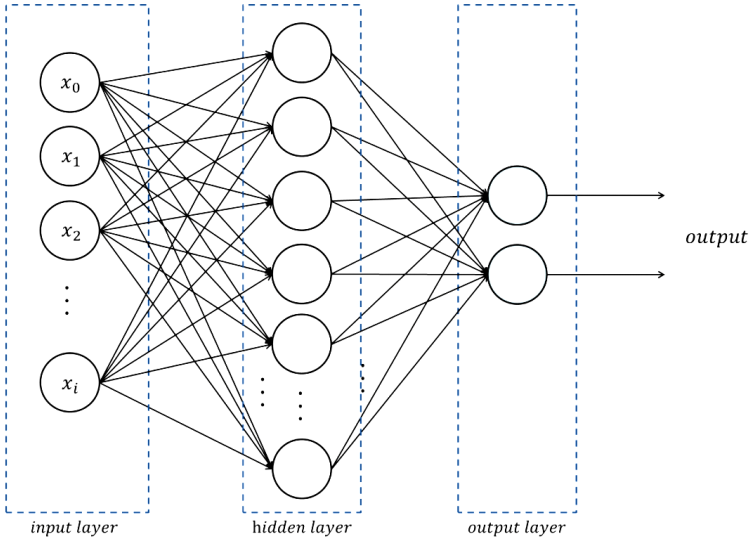


Fig. 4.7. Multilayer feedforward network

Recurrent Networks

These networks differ from feedforward network architectures in the sense that there is atleast one feedback loop. Thus, in these networks, for example, there could exist .one layer with feedback connections as shown in Fig. 4.8. There could also be neurons with self-feedback links, i.e. the output of a neuron is fed back into itself as input.

The idea behind RNNs is to make use of sequential information. In a traditional neural network we assume that all inputs (and outputs) are independent of each other. But for many tasks that's a very bad idea. If you want to predict the next word in a sentence you better know which words came before it. RNNs are called *recurrent* because they perform the same task for every element of a sequence, with the output being depended on the previous computations. Another way to think about RNNs is that they have a "memory" which captures information about what has been calculated so far. In theory RNNs can make use of information in arbitrarily long sequences, but in practice they are limited to looking back only a few steps.

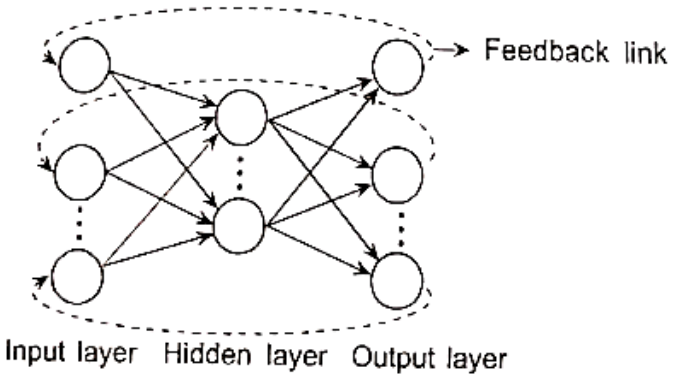


Fig. 4.8. A recurrent neural network.

Training Methods of Artificial Neural Networks

6.0.0 Introduction

The dynamics of neuron consists of two parts. One is the dynamics of the activation state and the second one is the dynamics of the synaptic weights. The Short Term Memory (STM) in neural networks is modeled by the activation state of the network and the Long Term Memory is encoded the information in the synaptic weights due to learning. The main property of artificial neural network is that, the ability of the learning from its environment and history. The network learns about its environment and history through its interactive process of adjustment applied to its synaptic weights and bias levels. Generally, the network becomes more knowledgeable about its environment and history, after completion each iteration of learning process. It is important to distinguish between representation and learning. Representation refers to the ability of a perceptron (or other network) to simulate a specified function. Learning requires the existence of a systematic procedure for adjusting the network weights to produce that function. Here we will discuss most of popular learning rules.

6.1.0 Definition of learning

There are too many activities associated with the notion of learning and we define learning in the context of neural networks [1] as

"*Learning is a process by which the free parameters of neural network are adapted through a process of stimulation by the environment in which the network is embedded. The type of learning is determined by the manner in which the parameter changes takes place*"

Based on the above definition the learning process of ANN can be divided into the following sequence of steps:

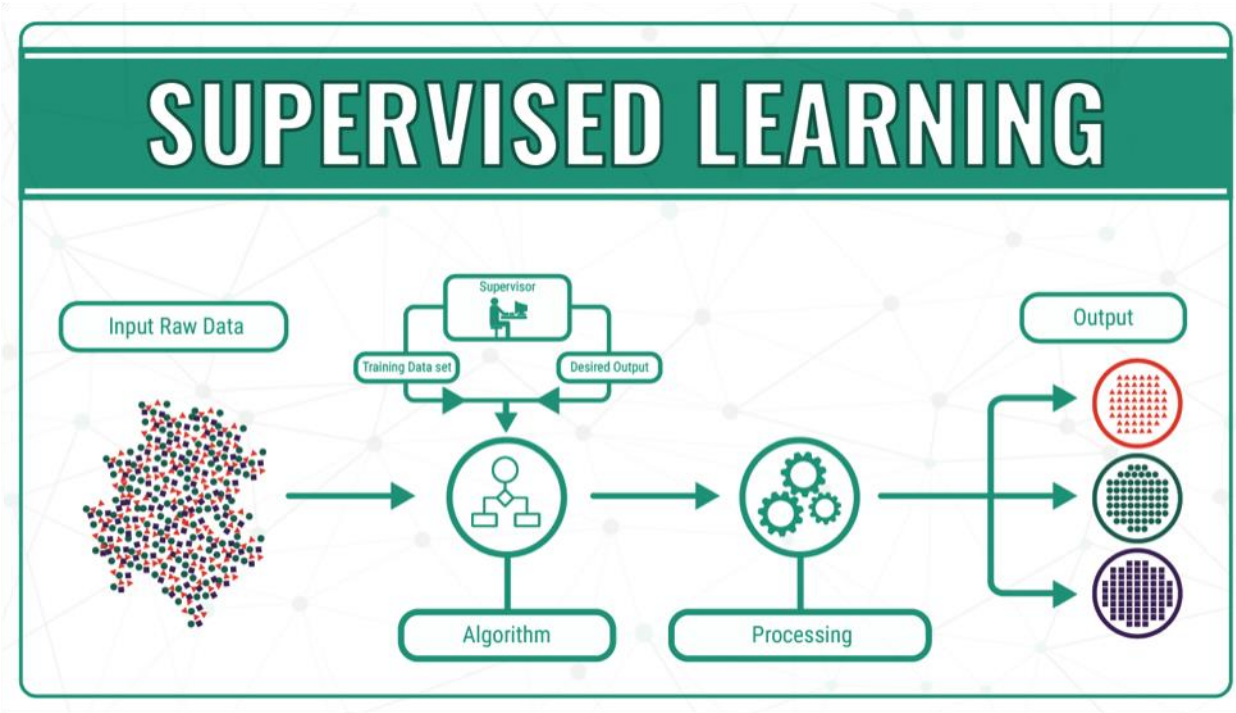
1. The ANN is stimulated by an environment.
2. The ANN undergoes changes in its free parameters as a result of the above stimulation.
3. The ANN responds in a new way to the environment because of the changes that have occurred in its internal structure.

6.1.1 Types of Learning Methods / Learning Strategies

A set of defined rules for the solution of a learning problem is called algorithm. There are different approaches to train an ANN. Most of the methods fall into one of two classes namely supervised learning and unsupervised learning.

***Supervised learning*:** An external signal known as teacher controls learning and incorporates information.

Supervised training requires the pairing of each input vector with a target vector representing the desired output; together these are called a training pair. Usually a network is trained over a number of such training pairs. An input vector is applied, the output of the network is calculated and compared to the corresponding target vector and the difference (error) is fed back through the network and weights are changed according to an algorithm that tends to minimize the error. The vectors of the training set are applied sequentially, and errors are calculated and weights adjusted for each vector, until the error for the entire training set is at the acceptably low value.

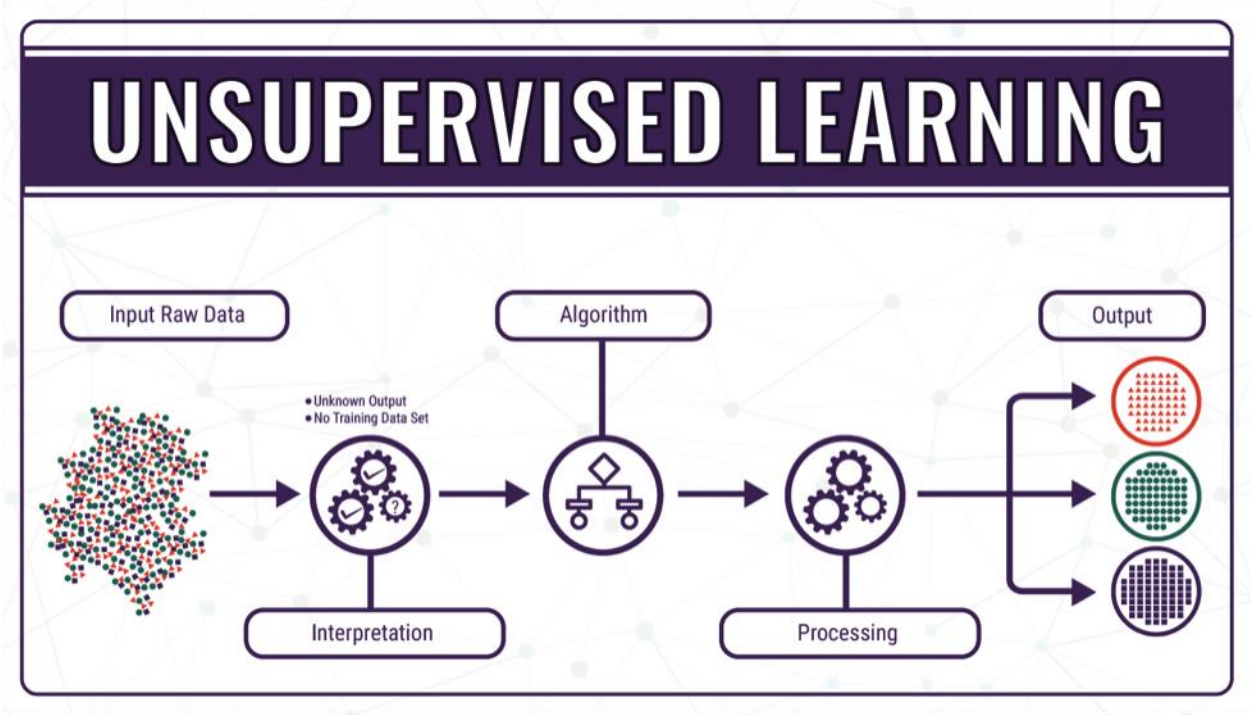


***Unsupervised learning*:** No external signal (teacher) is used in the learning process. The neural network relies upon both internal and local information.

Unsupervised training is a far more plausible model of training in the biological system. Developed by Kohonen (1984) and many others, it requires no target vector for the outputs, and hence, no comparisons to predetermined ideal responses. The training set consists solely

of input vectors. The training algorithm modifies network weights to produce output vectors that consistent; i.e., both application of one of the training vectors and application of a vector that is sufficiently similar to it will produce the same patterns of outputs.

- The training process, therefore, extracts the statistical properties of the training set and group's similar vector into classes.
- Applying a vector from a given class as a input will produce a specific output vector, but there is no way to determine prior to training which specific output pattern will be produced by a given input vector class. Hence, the outputs of such a network must generally be transformed into a comprehensible form subsequent to the training process.



6.1.2 Types of basic learning mechanisms

In general the training of any artificial neural network has to use one of the following basic learning mechanisms.

The basic learning mechanisms of neural networks are:

Hebbian learning rule

In this, the input–output pattern pairs (X_i, Y_i) are associated by the weight matrix W, known as the correlation matrix. It is computed as

$$W = \sum_{i=1}^{n} X_i Y_i^T \tag{2.8}$$

Here, Y_i^T is the transpose of the associated output vector Y_i. Numerous variants of the rule have been proposed (Anderson, 1983; Kosko, 1985; Lippman, 1987; Linsker, 1988).

Gradient descent learning

This is based on the minimization of error E defined in terms of weights and the activation function of the network. Also, it is required that the activation function employed by the network is differentiable, as the weight update is dependent on the gradient of the error E.

Thus, if ΔW_{ij} is the weight update of the link connecting the ith and jth neuron of the two neighbouring layers, then ΔW_{ij} is defined as

$$\Delta W_{ij} = -\eta \frac{\partial E}{\partial W_{ij}} \tag{2.9}$$

where, η is the learning rate parameter and $\partial E/\partial W_{ij}$ is the error gradient with reference to the weight W_{ij}.

The Widrow and Hoffs Delta rule and Backpropagation learning rule are all examples of this type of learning mechanism.

Competitive learning

In this method, those neurons which respond strongly to input stimuli have their weights updated. When an input pattern is presented, all neurons in the layer compete and the winning neuron undergoes weight adjustment. Hence, it is a "winner-takes-all" strategy.

Stochastic learning

In this method, weights are adjusted in a probabilistic fashion. An example is evident in simulated annealing—the learning mechanism employed by Boltzmann and Cauchy machines, which are a kind of NN systems.

Figure 2.11 illustrates the classification of learning algorithms.

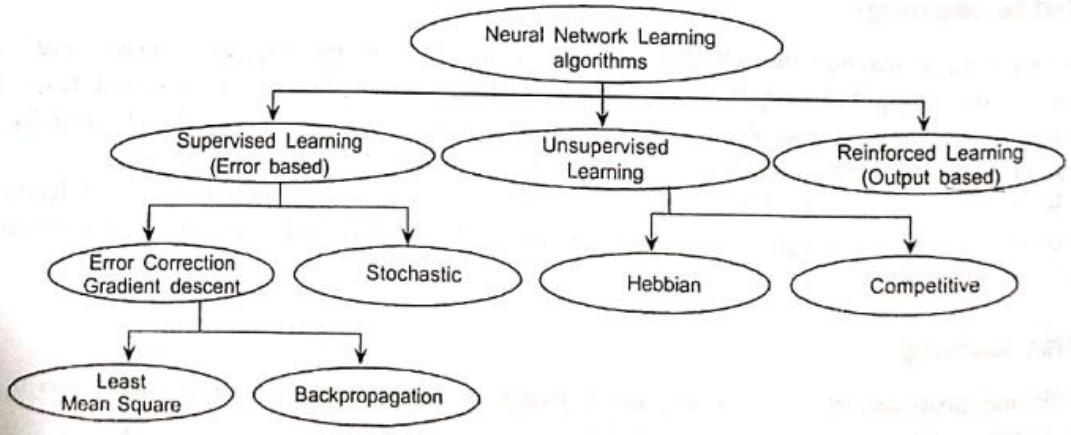


Fig. 2.11 Classification of learning algorithms.

Competitive Learning

In competitive learning, the output of neurons of a neural network competes among themselves to become active (fired). Whereas in a neural network based on Hebbian learning, several output neurons may be active simultaneously, in competitive learning only a single output neuron is active at any one time.

The basic elements in the competitive learning rule are:

- A set of neurons that are all the same except for some randomly distributed synaptic weights, which therefore respond differently to given set of input patterns.
- A limit imposed on the "strength" of each neuron.
- A mechanism that permits two neurons to compete for the right to respond to a given subset of outputs, such that only one output neuron or only one neuron per group is active (i.e. "on") at a time. The neuron that wins the competition is called a "Winner-takes-all neuron".

In the simplest form of competitive learning the neural network has a single layer of output neurons each of which is fully connected to the input nodes. The network may include of feedback connections among the neurons as indicated in Fig.6.6.

In the network architecture described herein the feedback connections perform lateral inhibition, with each neuron tending to exhibit the neuron to which it is laterally connected.

The feed forward and synaptic connections in the above network are all excitatory.

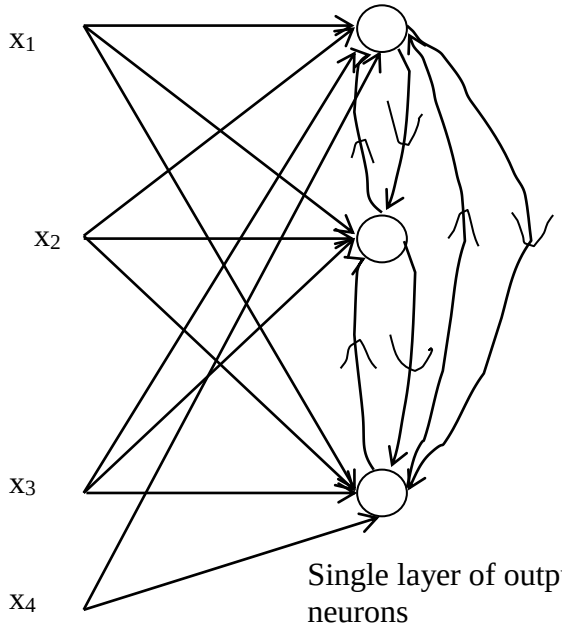


Figure 6.6. Architectural graph of a simple competitive learning network with feedforward (excitory) connections from the source nodes to the neurons, and lateral (inhibirory) connections among the neurons, the lateral connection are signified by open arrows[1].

Mathematical Relations of competitive learning

For a neuron k to be winning neuron, its induced local field v_k for a specified input pattern X must be the largest among all the neurons in the network. The output signal o_k of winning neuron k is set equal to one and the output signals of all the neurons that lose the competition are set equal to zero. This can be written as

$$o_k = \begin{cases} 1 & \text{if } v_k > v_j \quad \text{for all } j,\ j \neq k \\ 0 & \text{otherwise} \end{cases} \qquad (6.18)$$

where the induced local field v_k represents the combined action of all the forward and feedback inputs to the neuron k. Let w_{kj} denote the synaptic weight connecting input node j to neuron k. Suppose that each neuron is allotted a fixed amount of synaptic weight (i.e.; all synaptic weights are positive), which is distributed among its input nodes, i.e.

$$\sum_j w_{kj} = 1 \qquad \text{for all k} \qquad (6.19)$$

A neuron then learns by shifting synaptic weights from its inactive to active input nodes. If a neuron does not respond to a particular input pattern, no learning takes places in that neuron. If a particular neuron wins the competition, each input nodes of that neuron relinquish some proportion of its synaptic weight, and the weight relinquished is then distributed equally among the active input nodes.

According this rule, the change Δw_{kj} applied to synaptic weight w_{kj} is defined by

$$\Delta w_{kj} = \begin{cases} \eta(x_j - w_{kj}) & \text{if neuron k wins the competitio n.} \\ 0 & \text{if neuron k loses the competitio n.} \end{cases} \qquad (6.20)$$

where η is the learning rate parameter. This rule has the overall effect of moving the synaptic weight vector w_k of winning neuron k toward the input pattern X. The example of this mechanism is "winner-take-all learning rule" and discussed in the following section.

6.7.0 Boltzmann Learning

The Boltzman learning rule, named on honor of Ludwig Boltzman, is a statistical learning algorithm derived from ideas rooted in statistical machines. In a Boltzman machine the neurons forms a recurrent structure and they operate in a binary manner. That is, neuron 'on' state denoted by '+1' or in an "off" state denoted by –1.

The machine is characterized by an energy function, E the value of which is determined by the particular states occupied by the individual neurons of the machine, as given by

$$E = -\frac{1}{2}\sum_{j}\sum_{\substack{k \\ j \neq k}} w_{kj}\ x_k\ x_j \qquad (6.23)$$

where x_j is the state of neuron j, and w_{kj} is the synaptic weight connecting neuron j to neuron k. The fact that $j \neq k$ means simply none of the neurons in the machine has self-feedback. The machine operates by choosing a neuron at random for example, neuron k at some step of learning process, then flipping the state of neuron k from state x_k to state $-x_k$ at some temperate T with probability.

$$P(x_k \rightarrow -x_k) = \frac{1}{1+\exp(-\Delta E_k/T)} \qquad (6.24)$$

where ΔE_k is the energy change (i.e. the change in the energy function of the machine) resulting from such a flip.

Note that T is not a physical temperature, but rather a pseudo temperature. If this rule is applied repeatedly, the machine will reach thermal equilibrium.

The neurons of a Boltzman machine partition into two functional groups : visible and hidden. The visible neurons provide an interface between the network and the environment in which it operates, whereas the hidden neurons always operate freely.

There are two modes of operation to be considered.

- Clamped condition in which the visible neurons are all clamped into specific states determined by environment.
- Free running condition in which all the neurons (visible and hidden) are allowed to operate freely.

Let P_{kj}^{+} denote the correlation between the states of neurons j and k, with the network in its clamped condition. Let P_{kj}^{-} denote the correlation between the states of neurons j and k with the network in its free-running condition.

Both correlations are averaged over all possible states of the machine when it is in thermal equilibrium. Then, according to the Boltzmann learning rule, the change Δw_{kj} applied to the synaptic weight w_{kj} from the neuron j to neuron k defined by

$$\Delta w_{kj} = \eta\left(P_{kj}^{+} - P_{kj}^{-}\right), \qquad j \neq k \qquad (6.25)$$

where η is the learning rate parameter. Both P_{kj}^{+} and P_{kj}^{-} range in value from –1 to +1.

PERCEPTRONS

8.0.0 **Introduction**

We know that perceptron is one of the early models of artificial neuron. It was proposed by Rosenblatt in 1958. It is a single layer neural network whose weights and biases could be trained to produce a correct target vector when presented with the corresponding input vector. The perceptron is a program that learns *concepts*, i.e. it can learn to respond with *True* (1) or *False* (0) for inputs we present to it, by repeatedly "studying" examples presented to it. The training technique used is called *the perceptron learning rule*. The perceptron generated great interest due to its ability to *generalize* from its training vectors and work with randomly distributed connections. Perceptrons are especially suited for simple problems in pattern classification. In this also we give the perceptron convergence theorem.

8.1.0 Perceptron Model

In the 1960, perceptrons created a great deal of interest and optimism. Rosenblatt (1962) proved a remarkable theorem about perceptron learning. Widrow (Widrow 1961, 1963, Widrow and Angell 1962, Widrow and Hoff 1960) made a number of convincing demonstrations of perceptron like systems. Perceptron learning is of the supervised type. A perceptron is trained by presenting a set of patterns to its input, one at a time, and adjusting the weights until the desired output occurs for each of them.

The schematic diagram of perceptron is shown inn Fig. 8.1. Its synaptic weights are denoted by $w_1, w_2, \ldots w_n$. The inputs applied to the perceptron are denoted by $x_1, x_2, \ldots . x_n$. The externally applied bias is denoted by b.

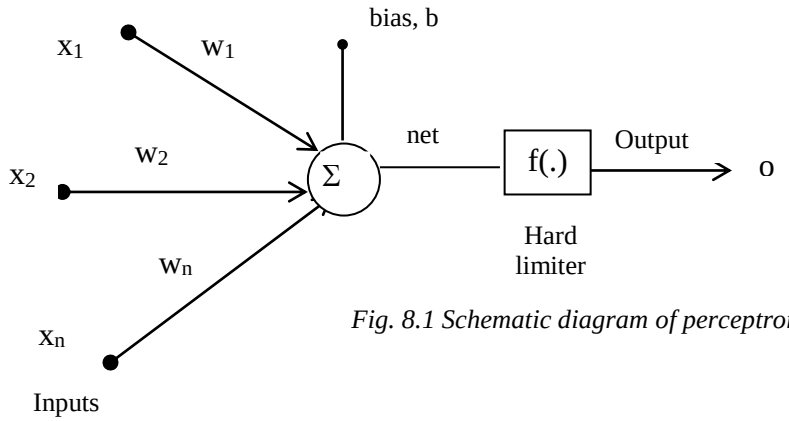


Fig. 8.1 Schematic diagram of perceptron

The net input to the activation of the neuron is written as

$$net = \sum_{i=1}^{n} w_i \, x_i + b \qquad (8.1)$$

The output of perceptron is written as $o = f(net)$ (8.2)

where f(.) is the activation function of perceptron. Depending upon the type of activation function, the perceptron may be classified into two types

i) Discrete perceptron, in which the activation function is ***hard limiter*** or ***sgn(·)*** function

ii) Continuous perceptron, in which the activation function is sigmoid function, which is differentiable. The input-output relation may be rearranged by considering w_0=b and fixed bias x_0 = 1.0. Then

$$net = \sum_{i=0}^{n} w_i x_i = WX \qquad (8.3)$$

where W = [w_0, w_1, w_2, ... w_n] and X = [x_0, x_1, x_2, . . . x_n]^T.

The learning rule for perceptron has been discussed in unit 7. Specifically the learning of these two models is discussed in the following sections.

8.2.0 Single Layer Discrete Perceptron Networks

For discrete perceptron the activation function should be *hard limiter* or *sgn()* function. The popular application of discrete perceptron is a pattern classification. To develop insight into the behavior of a pattern classifier, it is necessary to plot a map of the decision regions in n-dimensional space, spanned by the n input variables. The two decision regions separated by a hyper plane defined by

$$\sum_{i=0}^{n} w_i x_i = 0 \qquad (8.4)$$

This is illustrated in Fig. 8.2 for two input variables x_1 and x_2, for which the decision boundary takes the form of a straight line.

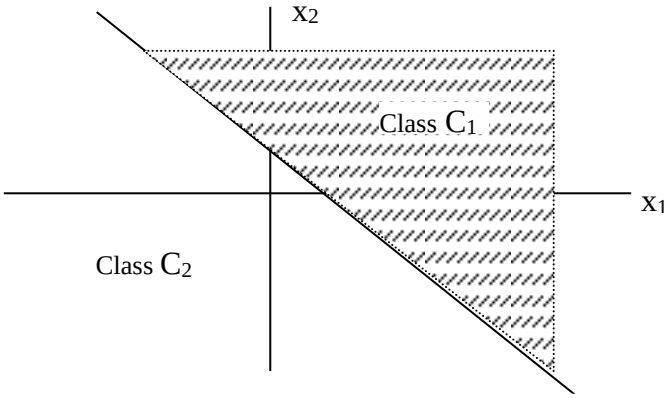


Fig. 8.2 Illustration of the hyper plane (in this example, a straight lines) as decision boundary for a two dimensional, two-class patron classification problem.

For the perceptron to function properly, the two classed C_1 and C_2 must be linearly separable. This in turn, means that the patterns to be classified must be sufficiently separated from each

other to ensure that the decision surface consists of a hyper plane. This is illustrated in Fig. 8.3.

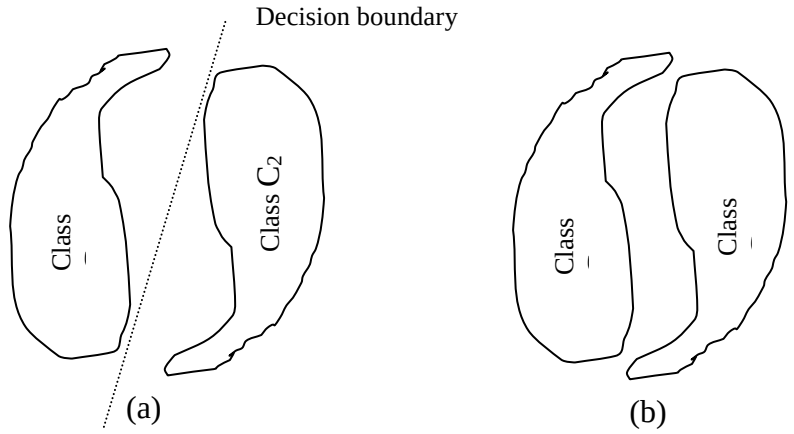


Fig. 8.3 (a) A pair of linearly separable patterns
(b) A pair of nonlinearly separable

In Fig. 8.3(a), the two classes C_1 and C_2 are sufficiently separated from each other to draw a hyper plane (in this it is a straight line) as the decision boundary. If however, the two classes C_1 and C_2 are allowed to move too close to each other, as in Fig. 8.3 (b), they become nonlinearly separable, a situation that is beyond the computing capability of the perceptron.

Suppose then that the input variables of the perceptron originate from two linearly separable classes. Let æ$_1$ be the subset of training vectors $X_1(1)$, $X_1(2)$, ... , that belongs to class C_1 and æ$_2$ be the subset of train vectors $X_2(1)$, $X_2(2)$, ... , that belong to class C_2. The union of æ$_1$ and æ$_2$ is the complete training set æ. Given the sets of vectors æ$_1$ and æ$_2$ to train the classifier, the training process involves the adjustment of the W in such a way that the two classes C_1 and C_2 are linearly separable. That is, there exists a weight vector W such that we may write,

$$\left.\begin{array}{l} WX > 0 \quad \text{for every input vect or X belonging to class } C_1 \\ WX \le 0 \quad \text{for every input vect or X belonging to class } C_2 \end{array}\right\} \qquad (8.5)$$

In the second condition, it is arbitrarily chosen to say that the input vector X belongs to class C_2 if WX = 0.

The algorithm for updating the weights may be formulated as follows:

1. If the k^{th} member of the training set, X_k is correctly classified by the weight vector W(k) computed at the k^{th} iteration of the algorithm, no correction is made to the weight vector of perceptron in accordance with the rule.

$W^{k+1} = W^k$ if $W^k X_k > 0$ and X_k belongs to class C_1 (8.6)

$W^{k+1} = W^k$ if $W^k X_k \le 0$ and X_k belongs to class C_2 (8.7)

2. Otherwise, the weight vector of the perceptron is updated in accordance with the rule.

$W^{(k+1)T} = W^{kT} - \eta X_k$ if $W^k X_k > 0$ and X_k belongs to class C_2 (8.8a)

$W^{(k+1)T} = W^{kT} + \eta X_k$ if $W^k X_k \leq 0$ and X_k belongs to class C_1 (8.8b)

where the learning rule parameter η controls the adjustment applied to the weight vector. Equations (8.8a) and (8.8b) may be written general expression as

$$W^{(k+1)} = W^{kT} + \frac{\eta}{2}(d_k - o_k)X_k \quad (8.9)$$

8.2.1 Summary of the discrete perceptron training algorithm

Given are P training pairs of patterns

{X_1, d_1, X_2, d_2, ... Xp, dp}, where X_i is (n×1), d_i is (1×1), i = 1, 2, . . . P. Define w_0=b is bias and X_0 = 1.0 , then the size of augmented input vector is X_i ((n+1)×).

In the following, k denotes the training step and p denotes the step counter with the training cycle.

Step 1: $\eta > 0$ is chosen and define E_{max}.

Step 2: Initialize the weights at small random values, W = [w_{ij}] , augmented size is (n+1)×1 and initialize counters and error function as:

$k \leftarrow 1$, $p \leftarrow 1$ $E \leftarrow 0$.

Step 3: The training cycle begins. Apply input and compute the output:

$X \leftarrow X_p$, $d \leftarrow d_p$, $o \leftarrow sgn(WX)$

Step 4: Update the weights: $W^T \leftarrow W^T + \eta (d - o) X$

Step 5: Compute the cycle error: $E \leftarrow \frac{1}{2}(d-o)^2 + E$

Step 6: If p < P, the p $\leftarrow$ p+1, k $\leftarrow$ k+1 and go to step 3, otherwise go to step 7.

Step 7: The training cycle is completed. For E< E_{max} terminates the training session with output weights and k. If E > E_{max} , then $E \leftarrow 0$, $p \leftarrow 1$ and enter the new training cycle by going to step 3.

In general, a continuous perceptron element with sigmoidal activation function will be used to facilitate the training of multi layer feed forward networks used for classification and recognition.

8.3.0 Single-Layer Continuous Perceptron networks

In this, the concept of an error function in multidimensional weight space has been introduced. Also the hard limiter (*sgn(.)*) with weights will be replaced by the continuous perceptron. By introduction of this continuous activation function, there are two advantages

(i) finer control over the training procedure and (ii) differential characteristics of the activation function, which is used for computation of the error gradient.

The gradient or steepest descent is used in updating weights starting from any arbitrary weight vector W, the gradient $\nabla E(W)$ of the current error function is computed. The next value of W as obtained by moving in the direction of the negative gradient along the multidimensional error surface. Therefore the relation of modification of weight vector may be written as

$$W^{(k+1)T} = W^{kT} - \eta \nabla E(W^k) \qquad (8.10)$$

where η is the learning constant and is the positive constant and the superscript k denotes the step number. Let us define the error function between the desired output d_k and actual output o_k as

$$E_k = \frac{1}{2}(d_k - o_k)^2 \qquad (8.11a)$$

or

$$E_k = \frac{1}{2}\left[d_k - f\left(W^k X\right)\right]^2 \qquad (8.11b)$$

where the coefficient ½ in from of the error expression is only for convenience in simplifying the expression of the gradient value and it does not effect the location of the error function minimization. The error minimization algorithm (8.10) requires computation of the gradient of the error function (8.11) and it may be written as

$$\nabla E(W^k) = \frac{1}{2}\nabla\left[d - f(net_k)\right]^2 \qquad (8.12)$$

The n+1 dimensional gradient vector is defined as

$$\nabla E(W^k) = \begin{bmatrix} \frac{\partial E}{\partial w_0} \\ \frac{\partial E}{\partial w_1} \\ . \\ . \\ \frac{\partial E}{\partial w_n} \end{bmatrix} \qquad (8.13)$$

Using (8.12), we obtain the gradient vector as

$$\nabla E(W^k) = -(d_k - o_k) f'(net_k) \begin{bmatrix} \frac{\partial(net_k)}{\partial w_0} \\ \frac{\partial(net_k)}{\partial w_1} \\ . \\ . \\ \frac{\partial(net_k)}{\partial w_n} \end{bmatrix} \qquad (8.14)$$

Since $net_k = W^k X$, we have

$$\frac{\partial(net_k)}{\partial w_i} = x_i, \qquad \text{for i =0, 1, . . . n.} \qquad (8.15)$$

(x_0=1 for bias element) and equation (8.15) can be written as

$$\nabla E(W^k) = -(d_k - o_k) f'(net_k) X \qquad (8.16a)$$

or

$$\frac{\partial E}{\partial w_i} = -(d_k - o_k) f'(net_k) x_i \qquad \text{for i = 0, 1, . . . n} \qquad (8.16b)$$

$$\therefore \Delta w_i^k = -\eta \nabla E(W^k) = \eta (d_k - o_k) f'(net_k) x_i \qquad (8.17)$$

Equation (8.17) is the training rule for the continuous perceptron. Now the requirement is how to calculate $f'(net)$ in terms of continuous perceptron output. Consider the bipolar activation function f(net) of the form

$$f(net) = \frac{2}{1+\exp(-net)} - 1 \qquad (8.18)$$

Differentiating the equation (8.18) with respect to net: $f'(net) = \frac{2 \times \exp(-net)}{[1+\exp(-net)]^2}$ (8.19)

The following identity can be used in finding the derivative of the function.

$$\frac{2 \times \exp(-net)}{[1+\exp(-net)]^2} = \frac{1}{2}(1 - o^2) \qquad (8.20)$$

The relation (8.20) may be verified as follows:

$$\frac{1}{2}(1 - o^2) = \frac{1}{2}\left[1 - \left(\frac{1-\exp(-net)}{1+\exp(-net)}\right)^2\right] \qquad (8.21)$$

The right side of (8.21) can be rearranged as

$$\frac{1}{2}\left[1 - \left(\frac{1-\exp(-net)}{1+\exp(-net)}\right)^2\right] = \frac{2\exp(-net)}{[1+\exp(-net)]^2} \qquad (8.22)$$

This is same as that of (8.20) and now the derivative may be written as

$$\therefore f'(net_k) = \frac{1}{2}(1 - o_k^2) \quad (8.23)$$

The gradient (8.16a) can be written as $\nabla E(W^k) = -\frac{1}{2}(d_k - o_k)(1 - o_k^2)X$ (8.24)

and the complete delta training for the bipolar continuous activation function results from (8.24) as

$$W^{(k+1)T} = W^{kT} + \frac{1}{2}\eta(d_k - o_k)(1 - o_k^2)X_k \quad (8.25)$$

where k denotes the reinstated number of the training step.

The weight adjustment rule (8.25) corrects the weights in the same direction as the discrete perceptron learning rule as in equation (8.8). The main difference between these two is the presence of the moderating factor $(1-o_k^2)$. This scaling factor is always positive and smaller than 1. Another main difference between the discrete and continuous perceptron training is that the discrete perceptron training algorithm always leads to a solution for linearly separable problems. In contrast to this property, the negative gradient-based training does not guarantee solutions for linearly separable patterns.

8.3.1 Summary of the Single Continuous Perceptron Training Algorithm

Given are P training pairs

$\{X_1, d_1, X_2, d_2, \ldots . X_p, d_p\}$, where X_i is $((n+1)\times 1)$, d_i is (1×1), for i = 1, 2, . . . P

$$X_i = \begin{bmatrix} x_{i0} \\ x_{i1} \\ . \\ . \\ x_{in} \end{bmatrix}, \text{ where } x_{i0} = 1.0 \text{ (bias element)}$$

Let k is the training step and p is the step counter within the training cycle.

Step 1: $\eta > 0$ and $E_{max} > 0$ chosen.

Step 2: Weights are initialized at W at small random values, $W = [w_{ij}]$ is $(n+1)\times 1$. Counter and error function are initialized.

$k \leftarrow 1, \quad p \leftarrow 1 \quad E \leftarrow 0.$

Step 3: The training cycle begins. Input is presented and output is computed.

$X \leftarrow X_p, \quad d \leftarrow d_p, \quad o \leftarrow f(WX)$

Step 4: Weights are updated: $W^T \leftarrow W^T + \frac{1}{2}\eta(d - o)(1 - o^2)X$

Step 5: Cycle error is computed: $E \leftarrow \frac{1}{2}(d - o)^2 + E$

Step 6: If p < P, the p $\leftarrow$ p+1, k $\leftarrow$ k+1 and go to step 3, otherwise go to step 7.

Step 7: The training cycle is completed. For E < E_{max} terminated the training session with output weights, k and E. If $E \geq E_{max}$, then E $\leftarrow$0, p$\leftarrow$1 and enter the new training cycle by going to step 3.

8.4.0 Perceptron Convergence Theorem

(Separate document sent)

8.5.0 Problems and Limitations of the perceptron training algorithms

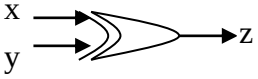
It may be difficult to determine if the caveat regarding linear separability is satisfied for the particular training set at hand. Further more, in many real world situations the inputs are often time varying and may be separable at one time and not at another. Also, these is no statement in the proof of the perceptron learning algorithm that indicates how many steps will be required to train the network. It is small consolation; to know that training will only take a finite number of steps if the time it takes is measured in geological units.

Further more, there is no proof that perceptron training algorithm is faster than simply trying all possible adjustment of the weights; in some cases this brute force approach may be superior.

8.5.1 Limitations of perceptrons

There are limitations to he capabilities of perceptrons however. They will learn the solution, if there is a solution to be found. First, the output values of a perceptron can take on only one of two values (True or False). Second, perceptrons can only classify *linearly separable* sets of vectors. If a straight line or plane can be drawn to separate the input vectors into their correct categories, the input vectors are linearly separable and the perceptron will find the solution. If the vectors are not linearly separable learning will never reach a point where all vectors are classified properly. The most famous example of the perceptron's inability to solve problems with linearly non-separable vectors is the boolean exclusive-OR problem.

Consider the case of the exclusive-or (XOR) problem. The XOR logic function has two inputs and one output, how below.



(a) Exclusive –OR gate

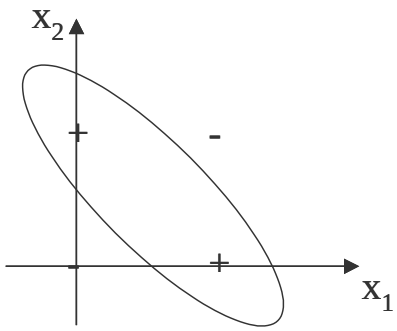
x	y	Z
0	0	0
0	1	1
1	0	1
1	1	0

(b) Truth Table

Fig. 8.4

It produces an output only if either one or the other of the inputs is on, but not if both are **off** or both are **on.** It is shown in above table. We can consider this has a problem that we want the perceptron to learn to solve; output a 1 of the x is **on** and y is **off** or y is **on** and x is **off**, otherwise output a '0'. It appears to be a simple enough problem.

We can draw it in pattern space as shown in Fig. (8.5). The x-axis represents the value of x, the y-axis represents the value of y. The shaded circles represent the inputs that produce an output of 1, whilst the un-shaded circles show the inputs that produce an output of 0. Considering the shaded circles and un-shaded circles as separate classes, we find that, we cannot draw a straight line to separate the two classes. Such patterns are known as linearly inseparable since no straight line can divide them up successfully. Since we cannot divide them with a single straight line, the perceptron will not be able to find any such line either, and so cannot solve such a problem. In fact, a single-layer perceptron cannot solve any problem that is linearly inseparable.



2.9.2 ADALINE Network

The Adaptive Linear Neural Element Network framed by Bernard Widrow of Stanford University, makes use of supervised learning. Figure 2.19 illustrates a simple ADALINE network. Here, there is only one output neuron and the output values are bipolar (−1 or +1). However, the inputs x_i could be binary, bipolar or real valued. The bias weight is w_0 with an input link of $x_0 = +1$. If the weighted sum of the inputs is greater than or equal to 0 then the output is 1 otherwise it is −1.

The supervised learning algorithm adopted by the network is similar to the perceptron learning algorithm. Devised by Widrow-Hoff (1960), the learning algorithm is also known as the Least Mean Square (LMS) or Delta rule. The rule is given by

$$W_i^{new} = W_i^{old} + \alpha (t - y)x_i \tag{2.15}$$

where, α is the learning coefficient, t is the target output, y is the computed output, and x_i is the input.

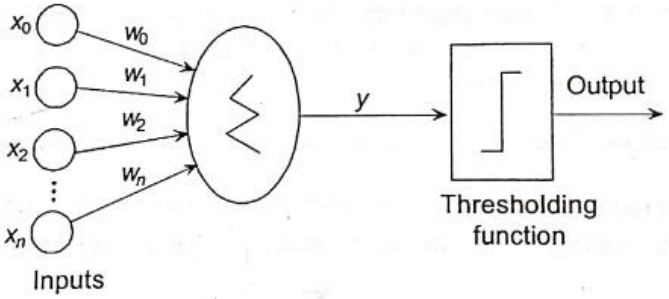


Fig. 2.19 A simple ADALINE network.

ADALINE network has had the most successful applications because it is used virtually in all high speed modems and telephone switching systems to cancel the echo in long distance communication circuits.

2.9.3 MADALINE Network

A MADALINE (Many ADALINE) network is created by combining a number of ADALINES. The network of ADALINES can span many layers. Figure 2.20 illustrates a simple MADALINE

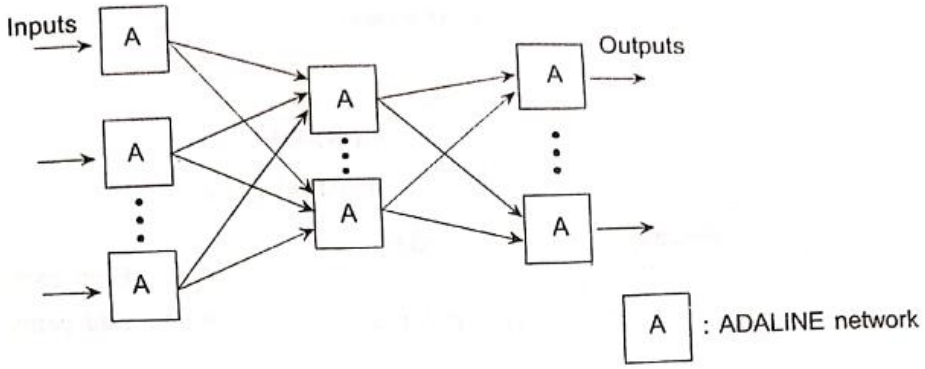


Fig. 2.20 MADALINE network.

network. *The use of multiple ADALINES helps counter the problem of non-linear separability.* For example, the MADALINE network with two units exhibits the capability to solve the XOR problem (refer Fig. 2.21). In this, each ADALINE unit receives the input bits x_1, x_2 and the bias input $x_0 = 1$ as its inputs. The weighted sum of the inputs is calculated and passed on to the bipolar threshold units. The logical 'and'ing (bipolar) of the two threshold outputs are computed to obtain the final output. Here, if the threshold outputs are both +1 or −1 then the final output is +1. If the threshold outputs are different, (i.e.) (+1, −1) then the final output is −1. Inputs which are of even parity produce positive outputs and inputs of odd parity produce negative outputs. Figure 2.22 shows the decision boundaries for the XOR problem while trying to classify the even parity inputs (positive outputs) from the odd parity inputs (negative outputs).

9.0.0 **Backpropagation Algorithm**

(Separate document Attached)