for their application to optimization. The field of probabilistic reasoning is also sometimes included under the soft computing umbrella for its control of randomness and uncertainty. The importance of soft computing lies in using these methodologies in partnership – they all offer their own benefits which are generally not competitive and can therefore work together. As a result, several hybrid systems were looked at – systems in which such partnerships exist.

[illegible]

Artificial Neural Network: An Introduction

2

Learning Objectives

- The fundamentals of artificial neural network.
- The evolution of neutal networks.
- Comparison between biological neuron and artificial neuron.
- Basic models of artificial neural networks.
- The different types of connections of neural networks, learning and activation functions are included.
- Various terminologies and notations used throughout the text.
- The basic fundamental neuron model – McCulloch–Pitts neuron and Hebb network.
- The concept of linear separability to form decision boundary regions.

Diff b/w [illegible]

2.1 Fundamental Concept

Neural networks are those information processing systems, which are constructed and implemented to model the human brain. The main objective of the neural network research is to develop a computational device for modeling the brain to perform various computational tasks at a faster rate than the traditional systems. Artificial neural networks perform various tasks such as pattern-matching and classification, optimization function, approximation, vector quantization, and data clustering. These tasks are very difficult for traditional computers, which are faster in algorithmic computational tasks and precise arithmetic operations. Therefore, for implementation of artificial neural networks, high-speed digital computers are used, which makes the simulation of neural processes feasible.

2.1.1 Artificial Neural Network

As already stated in Chapter 1, an artificial neural network (ANN) is an efficient information processing system which resembles in characteristics with a biological neural network. ANNs possess large number of highly interconnected processing elements called *nodes* or *units* or *neurons*, which usually operate in parallel and are configured in regular architectures. Each neuron is connected with the other by a connection link. Each connection link is associated with weights which contain information about the input signal. This information is used by the neuron net to solve a particular problem. ANNs' collective behavior is characterized by their ability to learn, recall and generalize training patterns or data similar to that of a human brain. They have the capability to model networks of original neurons as found in the brain. Thus, the ANN processing elements are called *neurons* or *artificial neurons*.

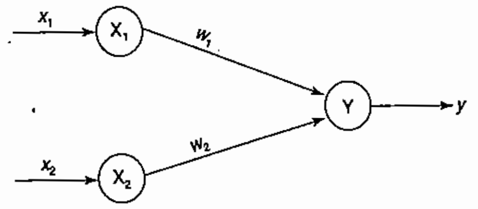

Figure 2-1 Architecture of a simple artificial neuron net.

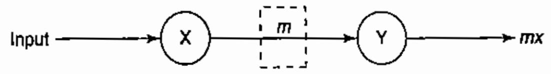

Figure 2-2 Neural net of pure linear equation.

It should be noted that each neuron has an internal state of its own. This internal state is called the *activation* or *activity level* of neuron, which is the function of the inputs the neuron receives. The activation signal of a neuron is transmitted to other neurons. Remember, a neuron can send only one signal at a time, which can be transmitted to several other neurons.

To depict the basic operation of a neural net, consider a set of neurons, say X_1 and X_2, transmitting signals to another neuron, Y. Here X_1 and X_2 are input neurons, which transmit signals, and Y is the output neuron, which receives signals. Input neurons X_1 and X_2 are connected to the output neuron Y, over a weighted interconnection links (W_1 and W_2) as shown in Figure 2-1.

For the above simple neuron net architecture, the net input has to be calculated in the following way:

$$y_{in} = +x_1w_1 + x_2w_2$$

where x_1 and x_2 are the activations of the input neurons X_1 and X_2, i.e., the output of input signals. The output y of the output neuron Y can be obtained by applying activations over the net input, i.e., the function of the net input:

$$y = f(y_{in})$$

$$\text{Output} = \text{Function (net input calculated)}$$

The function to be applied over the net input is called *activation function*. There are various activation functions, which will be discussed in the forthcoming sections. The above calculation of the net input is similar to the calculation of output of a pure linear straight line equation ($y = mx$). The neural net of a pure linear equation is as shown in Figure 2-2.

Here, to obtain the output y, the slope m is directly multiplied with the input signal. This is a linear equation. Thus, when slope and input are linearly varied, the output is also linearly varied, as shown in Figure 2-3. This shows that the weight involved in the ANN is equivalent to the slope of the linear straight line.

2.1.2 Biological Neural Network

It is well-known that the human brain consists of a huge number of neurons, approximately 10^{11}, with numerous interconnections. A schematic diagram of a biological neuron is shown in Figure 2-4.

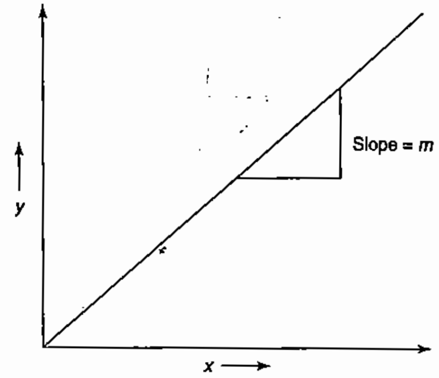

Figure 2-3 Graph for $y = mx$.

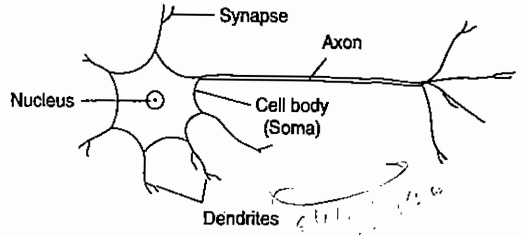

Figure 2-4 Schematic diagram of a biological neuron.

The biological neuron depicted in Figure 2-4 consists of three main parts:

1. *Soma* or *cell body* – where the cell nucleus is located.
2. *Dendrites* – where the nerve is connected to the cell body.
3. *Axon* – which carries the impulses of the neuron.

Dendrites are tree-like networks made of nerve fiber connected to the cell body. An axon is a single, long connection extending from the cell body and carrying signals from the neuron. The end of the axon splits into fine strands. It is found that each strand terminates into a small bulb-like organ called *synapse*. It is through synapse that the neuron introduces its signals to other nearby neurons. The receiving ends of these synapses on the nearby neurons can be found both on the dendrites and on the cell body. There are approximately 10^4 synapses per neuron in the human brain.

Electric impulses are passed between the synapse and the dendrites. This type of signal transmission involves a chemical process in which specific transmitter substances are released from the sending side of the junction. This results in increase or decrease in the electric potential inside the body of the receiving cell. If the electric potential reaches a threshold then the receiving cell fires and a *pulse* or *action potential* of fixed strength and duration is sent out through the axon to the synaptic junctions of the other cells. After firing, a cell has to wait for a period of time called the *refractory period* before it can fire again. The synapses are said to be ***inhibitory*** if they let passing impulses hinder the firing of the receiving cell or *excitatory* if they let passing impulses cause the firing of the receiving cell.

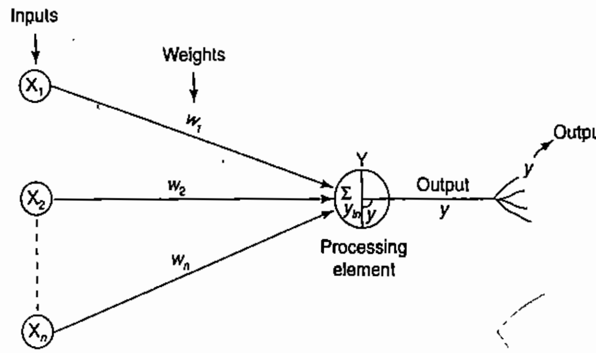

Figure 2-5 Mathematical model of artificial neuron.

Table 2-1 Terminology relationships between biological and artificial neurons

Biological neuron	Artificial neuron
Cell	Neuron
Dendrites	Weights or interconnections
Soma	Net input
Axon	Output

Figure 2-5 shows a mathematical representation of the above-discussed chemical processing taking place in an artificial neuron.

In this model, the net input is elucidated as

$$y_{in} = x_1w_1 + x_2w_2 + \cdots + x_nw_n = \sum_{i=1}^{n} x_iw_i$$

where i represents the ith processing element. The activation function is applied over it to calculate the output. The weight represents the strength of synapse connecting the input and the output neurons. A positive weight corresponds to an excitatory synapse, and a negative weight corresponds to an inhibitory synapse.

The terms associated with the biological neuron and their counterparts in artificial neuron are presented in Table 2-1.

2.1.3 Brain vs. Computer – Comparison Between Biological Neuron and Artificial Neuron (Brain vs. Computer)

A comparison could be made between biological and artificial neurons on the basis of the following criteria:

1. *Speed:* The ~~cycle time of~~ execution in the ANN ~~is of few nanoseconds~~ whereas in the case of biological neuron it is of a few milliseconds. Hence, the artificial neuron modeled using a computer is more faster.

2. *Processing:* Basically, the biological neuron can perform massive parallel operations simultaneously. The artificial neuron can also perform several parallel operations ~~simultaneously, but, in~~ general, the artificial neuron network process is faster than that of the brain.
3. *Size and complexity:* The total number of neurons in the brain is about 10^{11} and the total number of interconnections is about 10^{15}. Hence, it can be noted that the complexity of the brain is comparatively higher, i.e. the computational work takes places not only in the brain cell body, but also in axon, synapse, etc. On the other hand, the size and complexity of an ANN is based on the chosen application and the network designer. The size and complexity of a biological neuron is more than that of an artificial neuron.
4. *Storage capacity (memory):* The biological neuron stores the information in its interconnections or in synapse strength but in an artificial neuron it is stored in its contiguous memory locations. ~~In an artificial~~ neuron, the continuous loading of new information may sometimes overload the memory locations. As a result, some of the addresses containing older memory locations may be destroyed. But in case of the brain, new information can be added in the interconnections by adjusting the strength without destroying the older information. A disadvantage related to brain is that sometimes its memory may fail to recollect the stored information whereas in an artificial neuron, once the information is stored in its memory locations, it can be retrieved. Owing to these facts, the adaptability is more toward an artificial neuron.
5. *Tolerance:* The biological neuron possesses fault tolerant capability whereas the artificial neuron has no fault tolerance. The distributed nature of the biological neurons enables to store and retrieve information even when the interconnections in them get disconnected. Thus biological neurons ~~are fault tolerant~~. But in case of artificial neurons, the information gets corrupted if the network interconnections are disconnected. Biological neurons can accept redundancies, which is not possible in artificial neurons. Even when some cells die, the human nervous system appears to be performing with the same efficiency.
6. *Control mechanism:* In an artificial neuron modeled using a computer, there is a control unit present in Central Processing Unit, which can transfer and control precise scalar values from unit to unit, but there is no such control unit for monitoring in the brain. The strength of a neuron in the brain depends on the active chemicals present and whether neuron connections are strong or weak as a result of structure layer rather ~~than individual~~ synapses. However, the ANN possesses simpler interconnections and is free from chemical actions similar to those taking place in brain (biological neuron). Thus, the control mechanism of an artificial neuron is very simple compared to that of a biological neuron.

So, we have gone through a comparison between ANNs and biological neural networks. In short, we can say that an ANN possesses the following characteristics:

1. It is a neurally implemented mathematical model.
2. There exist a large ~~number of highly~~ interconnected processing elements called *neurons* in an ANN.
3. The interconnections with their weighted linkages hold the informative knowledge.
4. The input signals arrive at the processing elements through connections and connecting weights.
5. The processing elements of the ANN have the ability to learn, recall and generalize from the given data by suitable assignment or adjustment of weights.
6. The computational power can be demonstrated only by the collective behavior of neurons, and it should be noted that no single neuron carries specific information.

The above-mentioned characteristics make the ANNs as connectionist models, parallel distributed processing models, self-organizing systems, neuro-computing systems and neuro-morphic systems.

2.2 Evolution of Neural Networks

The evolution of neural networks has been facilitated by the rapid development of architectures and algorithms that are currently being used. The history of the development of neural networks along with the names of their designers is outlined Tablę 2-2.

In the later years, the discovery of the neural net resulted in the implementation of optical neural nets, Boltzmann machine, spatiotemporal nets, pulsed neural networks and support vector machines.

Table 2-2 Evolution of neural networks

Year	Neural network	Designer	Description
1943	McCulloch and Pitts neuron	McCulloch and Pitts	The arrangement of neurons in this case is a combination of logic functions. Unique feature of this neuron is the concept of threshold.
1949	Hebb network	Hebb	It is based upon the fact that if two neurons are found to be active simultaneously then the strength of the connection between them should be increased.
1958, 1959, 1962, 1988	Perceptron	Frank Rosenblatt, Block, Minsky and Papert	Here the weights on the connection path can be adjusted.
1960	Adaline	Widrow and Hoff	Here the weights are adjusted to reduce the difference between the net input to the output unit and the desired output. The result here is very negligible. Mean squared error is obtained.
1972	Kohonen self-organizing feature map	Kohonen	The concept behind this network is that the inputs are clustered together to obtain a fired output neuron. The clustering is performed by winner-take all policy.
1982, 1984, 1985, 1986, 1987	Hopfield network	John Hopfield and Tank	This neural network is based on fixed weights. These nets can also act as associative memory nets.
1986	Back-propagation network	Rumelhart, Hinton and Williams	This network is multi-layer with error being propagated backwards from the output units to the hidden units.
1988	Counter-propagation network	Grossberg	This network is similar to the Kohonen network; here the learning occurs for all units in a particular layer, and there exists no competition among these units.
1987–1990	Adaptive Resonance Theory (ART)	Carpenter and Grossberg	The ART network is designed for both binary inputs and analog valued inputs. Here the input patterns can be presented in any order.
1988	Radial basis function network	Broomhead and Lowe	This resembles a back propagation network but the activation function used is a Gaussian function.
1988	Neo cognitron	Fukushima	This network is essential for character recognition. The deficiency occurred in cognitron network (1975) was corrected by this network.

2.3 Basic Models of Artificial Neural Network

The models of ANN are specified by the three basic entities namely:

1. the model's synaptic interconnections;
2. the training or learning rules adopted for updating and adjusting the connection weights;
3. their activation functions.

2.3.1 Connections

The neurons should be visualized for their arrangements in layers. An ANN consists of a set of highly interconnected processing elements (neurons) such that each processing element output is found to be connected through weights to the other processing elements or to itself; delay lead and lag-free connections are allowed. Hence, the arrangements of these processing elements and the geometry of their interconnections are essential for an ANN. The point where the connection originates and terminates should be noted, and the function of each processing element in an ANN should be specified.

Besides the simple neuron shown in Figure ??, there exist several other types of neural network connections. The arrangement of neurons to form layers and the connection pattern formed within and between layers is called the *network architecture*. There exist five basic types of neuron connection architectures. They are:

1. single-layer feed-forward network;
2. multilayer feed-forward network;
3. single node with its own feedback;
4. single-layer recurrent network;
5. multilayer recurrent network.

Figures 2-6–2-10 depict the five types of neural network architectures. Basically, neural nets are classified into single-layer or multilayer neural nets. A layer is formed by taking a processing element and combining it with other processing elements. Practically, a layer implies a stage, going stage by stage, i.e., the input stage and the output stage are linked with each other. These linked interconnections lead to the formation of various network architectures. When a layer of the processing nodes is formed, the inputs can be connected to these

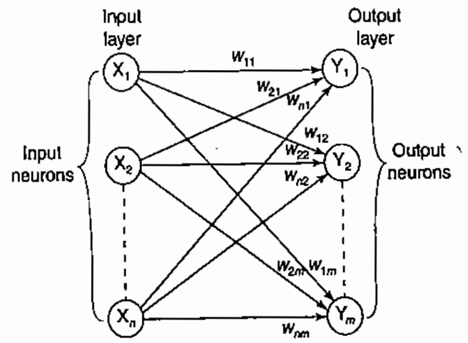

Figure 2-6 Single-layer feed-forward network.

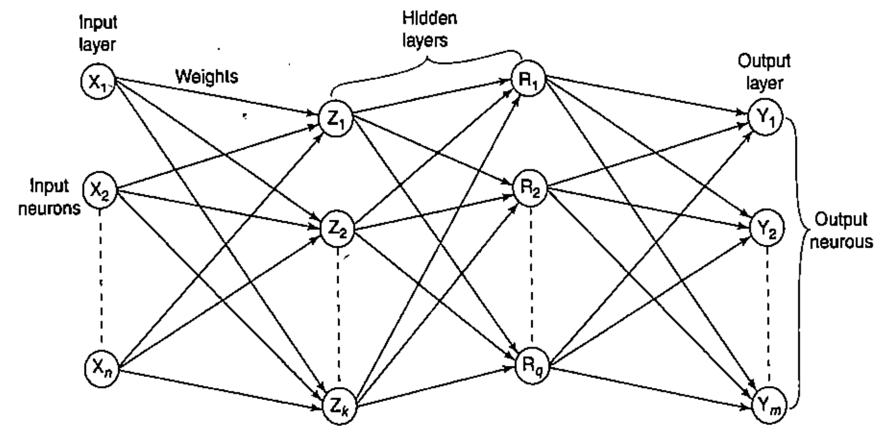

Figure 2-7 Multilayer feed-forward network.

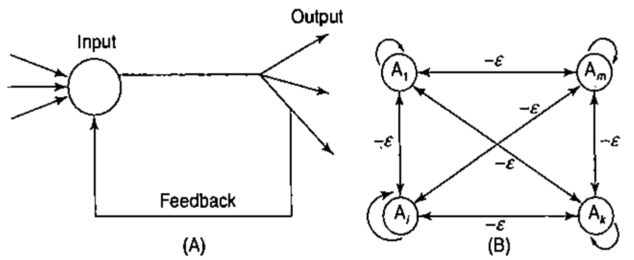

Figure 2-8 (A) Single node with own feedback. (B) Competitive nets.

nodes with various weights, resulting in ~~a series of outputs, one per node.~~ Thus, a single-layer *feed-forward network* is formed.

A multilayer feed-forward network (Figure 2-7) is formed by the interconnection of several layers. The input layer is that which receives the input and this layer has no function except buffering the input signal. The output layer generates the output of the network. Any layer that is formed between the input and output layers is called *hidden layer*. This hidden layer is internal to the network and has no direct contact with the external environment. It should be noted that there may be zero to several hidden layers in an ANN. More the number of the hidden layers, more is the complexity of the network. This may, however, provide an efficient output response. In case of ~~a fully~~ connected network every output from one layer is connected to each and every node in the next layer.

A network is said to be a feed-forward network if no neuron in the output layer is an input to a node in the same layer or in the preceding layer. On the other hand, when outputs can be directed back as inputs to same or preceding layer nodes then it results in the formation of *feedback networks*.

If the feedback of the output of the processing elements is directed back as input to the processing elements in the same layer ~~then it is called *lateral*~~ *feedback*. Recurrent networks are feedback networks with closed loop. Figure 2-8(A) shows a simple recurrent neural network having a single neuron with

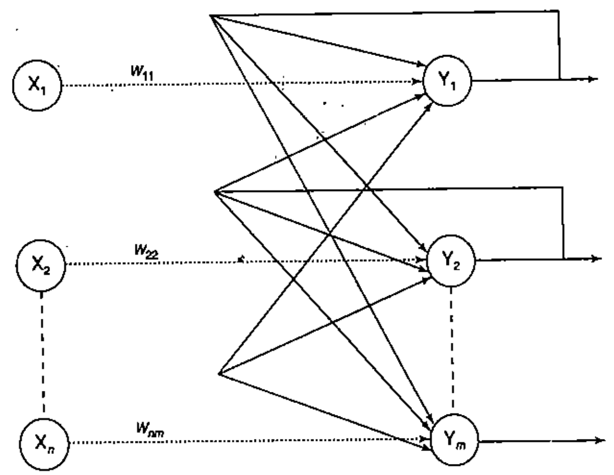

Figure 2-9 Single-layer recurrent network.

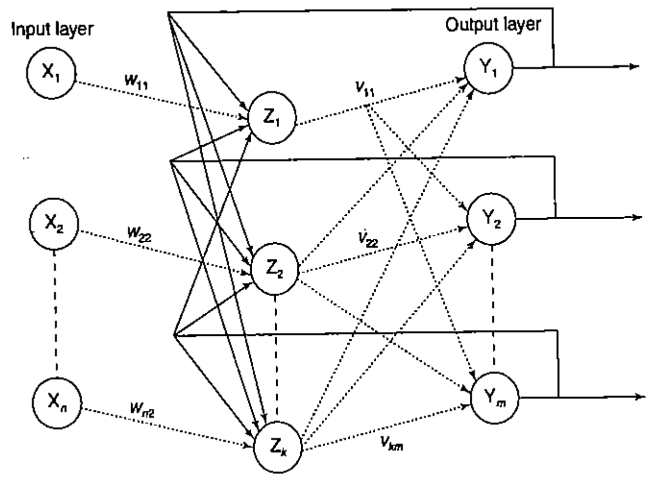

Figure 2-10 Multilayer recurrent network.

feedback to itself. Figure 2-9 shows a single-layer network with a feedback connection in which a processing element's output can be directed back to the processing element itself or to the other processing element or to both.

The architecture of a competitive layer is shown in Figure 2-8(B), the competitive interconnections having fixed weights of $-\varepsilon$. This net is called *Maxnet*, and will be discussed in the unsupervised learning network category. Apart from the network architectures discussed so far, there also exists another type of architecture with lateral feedback, which is called the *on-center-off-surround* or *lateral inhibition structure*. In this

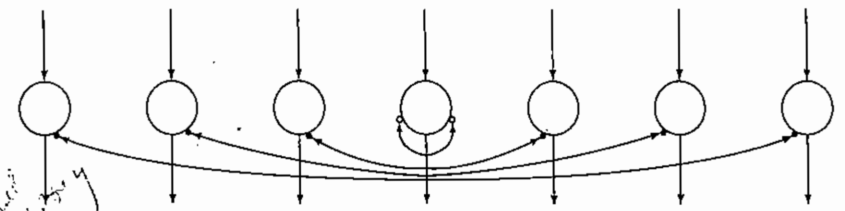

Figure 2-11 Lateral inhibition structure.

structure, each processing neuron receives two different classes of inputs – "excitatory" input from nearby processing elements and "inhibitory" inputs from more distantly located processing elements. This type of interconnection is shown in Figure 2-11.

In Figure 2-11, the connections with open circles are excitatory connections and the links with solid connective circles are inhibitory connections. From Figure 2-10, it can be noted that a processing element output can be directed back to the nodes in a preceding layer, forming a *multilayer recurrent network*. Also, in these networks, a processing element output can be directed back to the processing element itself and to other processing elements in the same layer. Thus, the various network architectures as discussed from Figures 2-6–2-11 can be suitably used for giving effective solution to a problem by using ANN.

2.3.2 Learning

The main property of an ANN is its capability to learn. Learning or training is a process by means of which a neural network adapts itself to a stimulus by making proper parameter adjustments, resulting in the production of desired response. Broadly, there are two kinds of learning in ANNs:

1. *Parameter learning:* It updates the connecting weights in a neural net.
2. *Structure learning:* It focuses on the change in network structure (which includes the number of processing elements as well as their connection types).

The above two types of learning can be performed simultaneously or separately. Apart from these two categories of learning, the learning in an ANN can be generally classified into three categories as: supervised learning; unsupervised learning; reinforcement learning. Let us discuss these learning types in detail.

2.3.2.1 Supervised Learning

The learning here is performed with the help of a teacher. Let us take the example of the learning process of a small child. The child doesn't know how to read/write. He/she is being taught by the parents at home and by the teacher in school. The children are trained and molded to recognize the alphabets, numerals, etc. Their each and every action is supervised by a teacher. Actually, a child works on the basis of the output that he/she has to produce. All these real-time events involve supervised learning methodology. Similarly, in ANNs following the supervised learning, each input vector requires a corresponding target vector, which represents the desired output. The input vector along with the target vector is called *training pair*. The network here is informed precisely about what should be emitted as output. The block diagram of Figure 2-12 depicts the working of a supervised learning network.

During training, the input vector is presented to the network, which results in an output vector. This output vector is the actual output vector. Then the actual output vector is compared with the desired (target) output vector. If there exists a difference between the two output vectors then an error signal is generated by

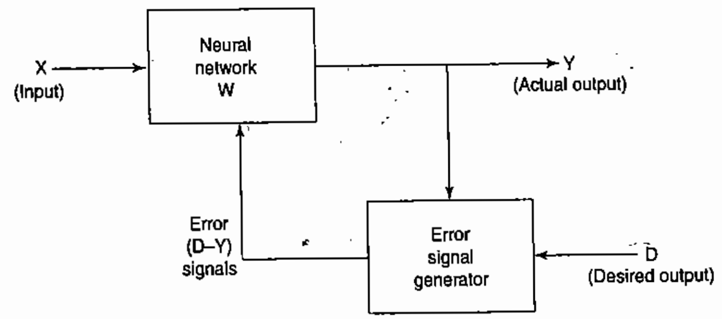

Figure 2-12 Supervised learning.

the network. This error signal is used for adjustment of weights until the actual output matches the desired (target) output. In this type of training, a supervisor or teacher is required for error minimization. Hence, the network trained by this method is said to be using supervised training methodology. In supervised learning, it is assumed that the correct "target" output values are known for each input pattern.

2.3.2.2 Unsupervised Learning

The learning here is performed without the help of a teacher. Consider the learning process of a tadpole, it learns by itself, that is, a child fish learns to swim by itself, it is not taught by its mother. Thus, its learning process is independent and is not supervised by a teacher. In ANNs following unsupervised learning, the input vectors of similar type are grouped without the use of training data to specify how a member of each group looks or to which group a number belongs. In the training process, the network receives the input patterns and organizes these patterns to form clusters. When a new input pattern is applied, the neural network gives an output response indicating the class to which the input pattern belongs. If for an input, a pattern class cannot be found then a new class is generated. The block diagram of unsupervised learning is shown in Figure 2-13.

From Figure 2-13 it is clear that there is no feedback from the environment to inform what the outputs should be or whether the outputs are correct. In this case, the network must itself discover patterns, regularities, features or categories from the input data and relations for the input data over the output. While discovering all these features, the network undergoes change in its parameters. This process is called *self-organizing* in which exact clusters will be formed by discovering similarities and dissimilarities among the objects.

2.3.2.3 Reinforcement Learning

This learning process is similar to supervised learning. In the case of supervised learning, the correct target output values are known for each input pattern. But, in some cases, less information might be available.

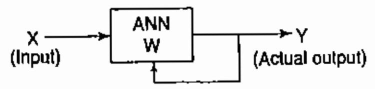

Figure 2-13 Unsupervised learning.

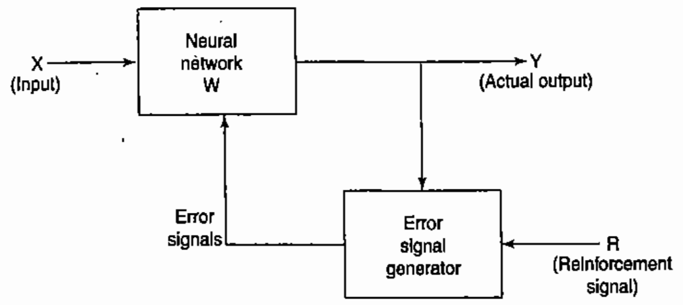

Figure 2-14 Reinforcement learning.

For example, the network might be told that its actual output is only "50% correct" or so. Thus, here only critic information is available, not the exact information. The learning based on this critic information is called *reinforcement learning* and the feedback sent is called *reinforcement signal*.

The block diagram of reinforcement learning is shown in Figure 2-14. The reinforcement learning is a form of supervised learning because the network receives some feedback from its environment. However, the feedback obtained here is only evaluative and not instructive. The external reinforcement signals are processed in the critic signal generator, and the obtained critic signals are sent to the ANN for adjustment of weights properly so as to get better critic feedback in future. The reinforcement learning is also called learning with a critic as opposed to learning with a teacher, which indicates supervised learning.

So, now you've a fair understanding of the three generalized learning rules used in the training process of ANNs.

2.3.3 Activation Functions

To better understand the role of the activation function, let us assume a person is performing some work. To make the work more efficient and to obtain exact output, some force or activation may be given. This activation helps in achieving the exact output. In a similar way, the activation function is applied over the net input ~~to calculate~~ the output of an ANN.

The information processing of a processing element can be viewed as consisting of two major parts: input and output. An integration function (say f) is associated with the input of a processing element. This function serves to combine activation, information or evidence from an external source or other processing elements into a net input to the processing element. The nonlinear activation function is used to ensure that a neuron's response is bounded – that is, the actual response of the neuron is conditioned or dampened as a result of large or small activating stimuli and is thus controllable.

Certain nonlinear functions are used to achieve the advantages of a multilayer network from a single-layer network. When a signal is fed through a multilayer network with linear activation functions, the output obtained remains same as that could be obtained using a single-layer network. Due to this reason, nonlinear functions are widely used in multilayer networks compared to linear functions.

There are several activation functions. Let us discuss a few in this section:

1. *Identity function:* It is a linear function and can be defined as

$$f(x) = x \quad \text{for all } x$$

[illegible]

The output here remains the same as input. The input layer uses the identity activation function.

2. *Binary step function:* This function can be defined as

$$f(x) = \begin{cases} 1 & \text{if } x \geqslant \theta \\ 0 & \text{if } x < \theta \end{cases}$$

where θ represents the threshold value. This function is most widely used in single-layer nets to convert the net input to an output that is a binary (1 or 0).

3. *Bipolar step function:* This function can be defined as

$$f(x) = \begin{cases} 1 & \text{if } x \geqslant \theta \\ -1 & \text{if } x < \theta \end{cases}$$

where θ represents the threshold value. This function is also used in single-layer nets to convert the net input to an output that is bipolar (+1 or −1).

4. *Sigmoidal functions:* The sigmoidal functions are widely used in back-propagation nets because of the relationship between the value of the functions at a point and the value of the derivative at that point which reduces the computational burden during training.
Sigmoidal functions are of two types:

- *Binary sigmoid function:* It is also termed as logistic sigmoid function or unipolar sigmoid function. It can be defined as

$$f(x) = \frac{1}{1 + e^{-\lambda x}}$$

where λ is the steepness parameter. The derivative of this function is

$$f'(x) = \lambda f(x)[1 - f(x)]$$

Here the range of the sigmoid function is from 0 to 1.

- *Bipolar sigmoid function:* This function is defined as

$$f(x) = \frac{2}{1 + e^{-\lambda x}} - 1 = \frac{1 - e^{-\lambda x}}{1 + e^{-\lambda x}}$$

where λ is the steepness ~~parameter~~ and the sigmoid function range is between −1 and +1. The derivative of this function ~~can~~ be

$$f'(x) = \frac{\lambda}{2}[1 + f(x)][1 - f(x)]$$

The bipolar sigmoidal function is closely related to hyperbolic tangent function, which is written as

$$h(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$

The derivative of the hyperbolic tangent function is

$$h'(x) = [1 + h(x)][1 - h(x)]$$

If the network uses a binary data, it is better to convert it to bipolar form and use the bipolar sigmoidal activation function or hyperbolic tangent function.

5. *Ramp function*: The ramp function is defined as

$$f(x) = \begin{cases} 1 & \text{if } x > 1 \\ x & \text{if } 0 \le x \le 1 \\ 0 & \text{if } x < 0 \end{cases}$$

The graphical representations of all the activation functions are shown in Figure 2-15(A)–(F).

2.4 Important Terminologies of ANNs

This section introduces you to the various terminologies related with ANNs.

2.4.1 Weights

In the architecture of an ANN, each neuron is connected to other neurons by means of directed communication links, and each communication link is associated with weights. The weights contain information about the input signal. This information is used by the net to solve a problem. The weight can be represented in terms of matrix. The weight matrix can also be called *connection matrix*. To form a mathematical notation, it is assumed that there are "n" processing elements in an ANN and each processing element has exactly "m" adaptive weights. Thus, the weight matrix W is defined by

$$W = \begin{bmatrix} w_1^T \\ w_2^T \\ \cdot \\ \cdot \\ \cdot \\ w_n^T \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1m} \\ w_{21} & w_{22} & \cdots & w_{2m} \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ w_{n1} & w_{n2} & \cdots & w_{nm} \end{bmatrix}$$

where $w_i = [w_{i1}, w_{i2}, \ldots, w_{im}]^T, i = 1, 2, \ldots, n$, is the weight vector of processing element and w_{ij} is the weight from processing element "i" (source node) to processing element "j" (destination node).

If the weight matrix W contains all the adaptive elements of an ANN, then the set of all W matrices will determine the set of all possible information processing configurations for this ANN. The ANN can be realized by finding an appropriate matrix W. Hence, the weights encode long-term memory (LTM) and the activation states of neurons encode short-term memory (STM) in a neural network.

2.4.2 Bias

The bias included in the network has its impact in calculating the net input. The bias is included by adding a component $x_0 = 1$ to the input vector X. Thus, the input vector becomes

$$X = (1, X_1, \ldots, X_i, \ldots, X_n)$$

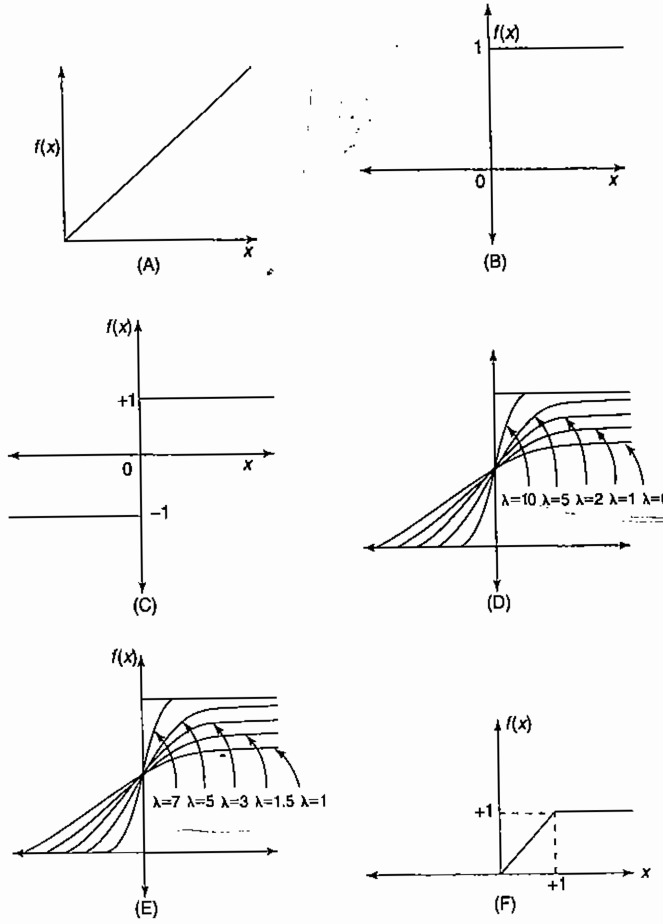

Figure 2-15 Depiction of activation functions: (A) identity function; (B) binary step function; (C) bipolar step function; (D) binary sigmoidal function; (E) bipolar sigmoidal function; (F) ramp function.

The bias is considered like another weight, that is, $w_{0j} = b_j$. Consider a simple network shown in Figure 2-16 with bias. From Figure 2-16, the net input to the output neuron Y_j is calculated as

$$y_{inj} = \sum_{i=0}^{n} x_i w_{ij} = x_0 w_{0j} + x_1 w_{1j} + x_2 w_{2j} + \cdots + x_n w_{nj}$$

$$= w_{0j} + \sum_{i=1}^{n} x_i w_{ij}$$

$$y_{inj} = b_j + \sum_{i=1}^{n} x_i w_{ij}$$

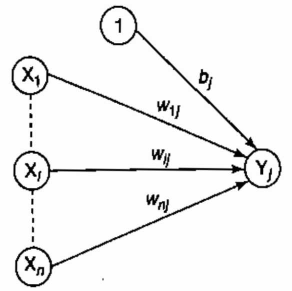

Figure 2-16 Simple net with bias.

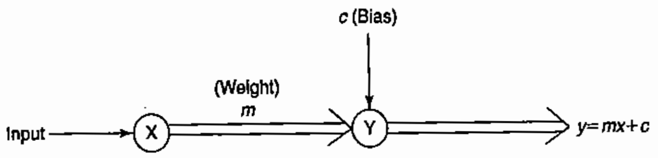

Figure 2-17 Block diagram for straight line.

The activation function discussed in Section 2.3.3 is applied over this net input to calculate the output. The bias can also be explained as follows: Consider an equation of straight line,

$$y = mx + c$$

where x is the input, m is the weight, c is the bias and y is the output. The equation of the straight line can also be represented as a block diagram shown in Figure 2-17. Thus, bias plays a major role in determining the output of the network.

The bias can be of two types: positive bias and negative bias. The positive bias helps in increasing the net input of the network and the negative bias helps in decreasing the net input of the network. Thus, as a result of the bias effect, the output of the network can be varied.

2.4.3 Threshold

Threshold is a set value based upon which the final output of the network may be calculated. The threshold value is used in the activation function. A comparison is made between the calculated net input and the threshold to obtain the network output. For each and every application, there is a threshold limit. Consider a direct current (DC) motor. If its maximum speed is 1500 rpm then the threshold based on the speed is 1500 rpm. If the motor is run on a speed higher than its set threshold, it may damage motor coils. Similarly, in neural networks, based on the threshold value, the activation functions are defined and the output is calculated. The activation function using threshold can be defined as

$$f(\text{net}) = \begin{cases} 1 & \text{if net} \geqslant \theta \\ -1 & \text{if net} < \theta \end{cases}$$

where θ is the fixed threshold value.

2.4.4 Learning Rate

used in perceptron

The learning rate is denoted by "α." It is used to control the amount of weight adjustment at each step of training. The learning rate, ranging from 0 to 1, determines the rate of learning at each time step.

2.4.5 Momentum Factor

Convergence is made faster if a momentum factor is added to the weight updation process. This is generally done in the back propagation network. If momentum has to be used, the weights from one or more previous training patterns must be saved. Momentum helps the net in reasonably large weight adjustments until the corrections are in the same general direction for several patterns.

2.4.6 Vigilance Parameter

The vigilance parameter is denoted by "ρ." It is generally used in adaptive resonance theory (ART) network. The vigilance parameter is used to control the degree of similarity required for patterns to be assigned to the same cluster unit. The choice of ~~vigilance parameter ranges approx~~imately from 0.7 to 1 to perform useful work in controlling the number of clusters.

2.4.7 Notations

The notations mentioned in this section have been used in this textbook for explaining each network.

x_i:	Activation of unit X_i, input signal.
y_j:	Activation of unit Y_j, $y_j = f(y_{inj})$
w_{ij}:	Weight on connection from unit X_i to unit Y_j.
b_j:	Bias acting on unit j. Bias has a constant activation of 1.
W:	Weight matrix, $W = \{w_{ij}\}$
y_{inj}:	Net input to unit Y_j given by $y_{inj} = b_j + \sum_i x_i w_{ij}$
$\|\|x\|\|$:	Norm of magnitude vector X.
θ_j:	Threshold for activation of neuron Y_j.
S:	Training input vector, $S = (s_1, \ldots, s_i, \ldots, s_n)$
T:	Training output vector, $T = (t_1, \ldots, t_j, \ldots, t_n)$
X:	Input vector, $X = (x_1, \ldots, x_i, \ldots, x_n)$
Δw_{ij}:	Change in weights given by $\Delta w_{ij} = w_{ij}(\text{new}) - w_{ij}(\text{old})$
α:	Learning rate; it controls the amount of weight adjustment at each step of training.

2.5 McCulloch–Pitts Neuron

2.5.1 Theory

The McCulloch–Pitts neuron was the earliest neural network discovered in 1943. It is usually called as *M–P neuron*. The M–P neurons are connected by directed weighted paths. It should be noted that the activation of a M–P neuron is binary, that is, at any time step the neuron may fire or may not fire. The weights associated with the communication links may be excitatory (weight is positive) or inhibitory (weight is negative). All the excitatory connected weights entering into a particular neuron will have same weights. The threshold plays a major role in M–P neuron: There is a fixed threshold for each neuron, and if the net input to the neuron is greater than the threshold then the neuron fires. Also, it should be noted that any nonzero inhibitory input would prevent the neuron from firing. The M–P neurons are most widely used in the case of logic functions.

2.5.2 Architecture

A simple M–P neuron is shown in Figure 2-18. As already discussed, the M–P neuron has both excitatory and inhibitory connections. It is excitatory with weight ($w > 0$) or inhibitory with weight $-p(p < 0)$. In Figure 2-18, inputs from x_1 to x_n possess excitatory weighted connections and inputs from x_{n+1} to x_{n+m} possess inhibitory weighted interconnections. Since the firing of the output neuron is based upon the threshold, the activation function here is defined as

$$f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} \geqslant \theta \\ 0 & \text{if } y_{in} < \theta \end{cases}$$

For inhibition to be absolute, the threshold with the activation function should satisfy the following condition:

$$\theta > nw - p$$

The output will fire if it receives say "k" or more excitatory inputs but no inhibitory inputs, where

$$kw \geq \theta > (k-1)w$$

The M–P neuron has no particular training algorithm. An analysis has to be performed to determine the values of the weights and the threshold. Here the weights of the neuron are set along with the threshold to make the neuron perform a simple logic function. The M–P neurons are used as building blocks on which we can model any function or phenomenon, which can be represented as a logic function.

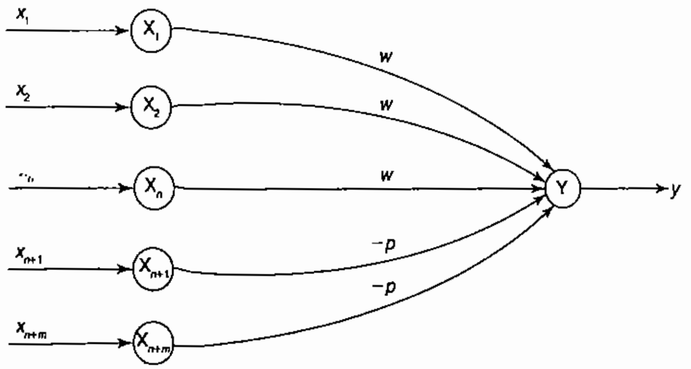

Figure 2-18 McCulloch–Pitts neuron model.

2.6 Linear Separability

An ANN does not give an exact solution for a nonlinear problem. However, it provides possible approximate solutions to nonlinear problems. Linear separability is the concept wherein the separation of the input space into regions is based on whether the network response is positive or negative.

A decision line is drawn to separate positive and negative responses. The decision line may also be called as the *decision-making line* or *decision-support line* or *linear-separable line*. The necessity of the linear separability concept was felt to classify the patterns based upon their output responses. Generally the net input calculated to the output unit is given as

$$y_{in} = b + \sum_{i=1}^{n} x_i w_i$$

For example, if a bipolar step activation function is used over the calculated net input (y_{in}) then the value of the function is 1 for a positive net input and −1 for a negative net input. Also, it is clear that there exists a boundary between the regions where $y_{in} > 0$ and $y_{in} < 0$. This region may be called as *decision boundary* and can be determined by the relation

$$b + \sum_{i=1}^{n} x_i w_i = 0$$

On the basis of the number of input units in the network, the above equation may represent a line, a plane or a hyperplane. The linear separability of the network is based on the decision-boundary line. If there exist weights (with bias) for which the training input vectors having positive (correct) response, +1, lie on one side of the decision boundary and all the other vectors having negative (incorrect) response, −1, lie on the other side of the decision boundary then we can conclude the problem is "linearly separable."

Consider a single-layer network as shown in Figure 2-19 with bias included. The net input for the network shown in Figure 2-19 is given as

$$y_{in} = b + x_1 w_1 + x_2 w_2$$

The separating line for which the boundary lies between the values x_1 and x_2, so that the net gives a positive response on one side and negative response on other side, is given as

$$b + x_1 w_1 + x_2 w_2 = 0$$

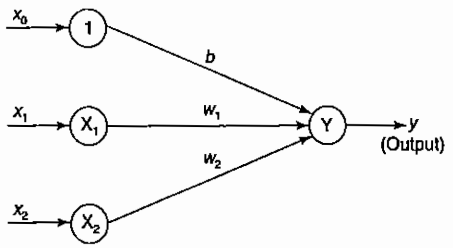

Figure 2-19 A single-layer neural net.

If weight w_2 is not equal to 0 then we get

$$x_2 = -\frac{w_1}{w_2}x_1 - \frac{b}{w_2}$$

Thus, the requirement for the positive response of the net is

$$\boxed{b + x_1w_1 + x_2w_2 > 0}$$

During training process, the values of w_1, w_2 and b are determined so that the net will produce a positive (correct) response for the training data. If on the other hand, threshold value is being used, then the condition for obtaining the positive response from output unit is

$$\text{Net input received} > \theta \text{ (threshold)}$$
$$y_{in} > \theta$$
$$x_1w_1 + x_2w_2 > \theta$$

The separating line equation will then be

$$x_1w_1 + x_2w_2 = \theta$$
$$x_2 = -\frac{w_1}{w_2}x_1 + \frac{\theta}{w_2} \quad (\text{with } w_2 \neq 0)$$

During training process, the values of w_1 and w_2 have to be determined, so that the net will have a correct response to the training data. For this correct response, the line passes close through the origin. In certain situations, even for correct response, the separating line does not pass through the origin.

Consider a network having positive response in the first quadrant and negative response in all other quadrants (AND function) with either binary or bipolar data, then the decision line is drawn separating the positive response region from the negative response region. This is depicted in Figure 2-20.

Thus, based on the conditions discussed above, the equation of this decision line may be obtained. Also, in all the networks that we would be discussing, the representation of data plays a major role.

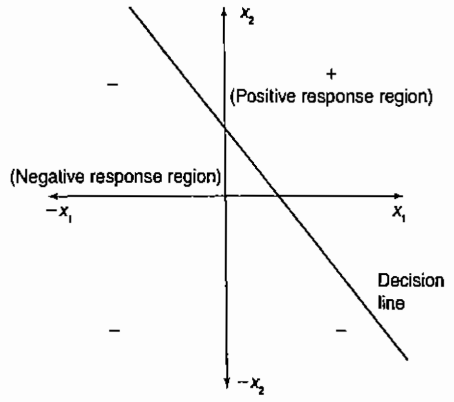

Figure 2-20 Decision boundary line.

However, the data representation mode has to be decided – whether it would be in binary form or in bipolar form. It may be noted that the bipolar representation is better than the binary representation. Using bipolar data representation, the missing data can be distinguished from mistaken data. Missing values are represented by 0 and mistakes can be represented by reversing the input value from +1 to −1 or vice-versa.

2.7 Hebb Network (only one outputunit)

2.7.1 Theory

For a neural net, the Hebb learning rule is a simple one. Let us understand it. Donald Hebb stated in 1949 that in the brain, the learning is performed by the change in the synaptic gap. Hebb explained it: "When an axon of cell A is near enough to excite cell B, and repeatedly or permanently takes place in firing it, some growth process or metabolic change takes place in one or both the cells such that A's efficiency, as one of the cells firing B, is increased."

According to the Hebb rule, the weight vector is found to increase proportionately to the product of the input and the learning signal. Here the learning signal is equal to the neuron's output. In Hebb learning, if two interconnected neurons are 'on' simultaneously then the weights associated with these neurons can be increased by the modification made in their synaptic gap (strength). The weight update in Hebb rule is given by

$$w_i(\text{new}) = w_i(\text{old}) + x_iy$$

The Hebb rule is more suited for bipolar data than binary data. If binary data is used, the above weight updation formula cannot distinguish two conditions namely:

1. A training pair in which an input unit is "on" and target value is "off."
2. A training pair in which both the input unit and the target value are "off."

Thus, there are limitations in Hebb rule application over binary data. Hence, the representation using bipolar data is advantageous.

2.7.2 Flowchart of Training Algorithm

The training algorithm is used for the calculation and adjustment of weights. The flowchart for the training algorithm of Hebb network is given in Figure 2-21. The notations used in the flowchart have already been discussed in Section 2.4.7.

In Figure 2-21, $s:t$ refers to each training input and target output pair. Till there exists a pair of training input and target output, the training process takes place; else, it is stopped.

2.7.3 Training Algorithm

The training algorithm of Hebb network is given below:

Step 0: First initialize the weights. Basically in this network they may be set to zero, i.e., $w_i = 0$ for $i = 1$ to n where "n" may be the total number of input neurons.

Step 1: Steps 2–4 have to be performed for each input training vector and target output pair, $s:t$.

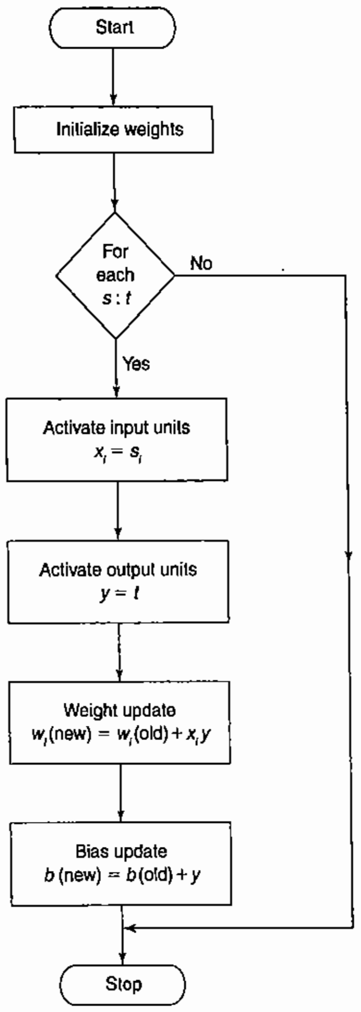

Figure 2-21 Flowchart of Hebb training algorithm.

[illegible]

Step 2: Input units activations are set. Generally, the activation function of input layer is identity function: $x_i = s_i$ for $i = 1$ to n.

Step 3: Output units activations are set: $y = t$.

Step 4: Weight adjustments and bias adjustments are performed:

$$w_i(\text{new}) = w_i(\text{old}) + x_i y$$
$$b(\text{new}) = b(\text{old}) + y$$

The above five steps complete the algorithmic process. In Step 4, the weight updation formula can also be given in vector form as

$$w(\text{new}) = w(\text{old}) + xy$$

Here the change in weight can be expressed as

$$\Delta w = xy$$

As a result,

$$w(\text{new}) = w(\text{old}) + \Delta w$$

The Hebb rule can be used for pattern association, pattern categorization, pattern classification and over a range of other areas.

2.8 Summary

In this chapter we have discussed the basics of an ANN and its growth. A detailed comparison between biological neuron and artificial neuron has been included to enable the reader understand the basic difference between them. An ANN is constructed with few basic building blocks. The building blocks are based on the models of artificial neurons and the topology of few basic structures. Concepts of supervised learning, unsupervised learning and reinforcement learning are briefly included in this chapter. Various activation functions and different types of layered connections are also considered here. The basic terminologies of ANN are discussed with their typical values. A brief description on McCulloch–Pitts neuron model is provided. The concept of linear separability is discussed and illustrated with suitable examples. Details are provided for the effective training of a Hebb network.

2.9 Solved Problems

1. For the network shown in Figure 1, calculate the net input to the output neuron.

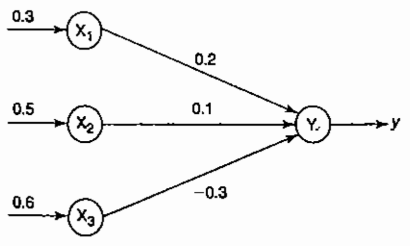

Figure 1 Neural net.

Solution: The given neural net consists of three input neurons and one output neuron. The inputs and weights are

$$[x_1, x_2, x_3] = [0.3, 0.5, 0.6]$$
$$[w_1, w_2, w_3] = [0.2, 0.1, -0.3]$$

The net input can be calculated as

$$\begin{aligned} y_{in} &= x_1 w_1 + x_2 w_2 + x_3 w_3 \\ &= 0.3 \times 0.2 + 0.5 \times 0.1 + 0.6 \times (-0.3) \\ &= 0.06 + 0.05 - 0.18 = -0.07 \end{aligned}$$

2. Calculate the net input for the network shown in Figure 2 with bias included in the network.

Solution: The given net consists of two input neurons, a bias and an output neuron. The inputs are

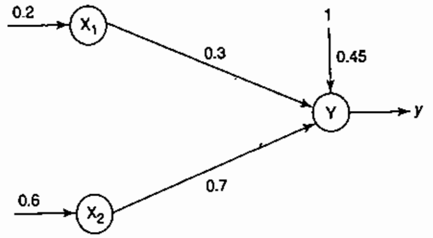

Figure 2 Simple neural net.

$[x_1, x_2] = [0.2, 0.6]$ and the weights are $[w_1, w_2] = [0.3, 0.7]$. Since the bias is included $b = 0.45$ and bias input x_0 is equal to 1, the net input is calculated as

$$
\begin{aligned}
y_{in} &= b + x_1w_1 + x_2w_2 \\
&= 0.45 + 0.2 \times 0.3 + 0.6 \times 0.7 \\
&= 0.45 + 0.06 + 0.42 = 0.93
\end{aligned}
$$

Therefore $y_{in} = 0.93$ is the net input.

3. Obtain the output of the neuron Y for the network shown in Figure 3 using activation functions as: (i) binary sigmoidal and (ii) bipolar sigmoidal.

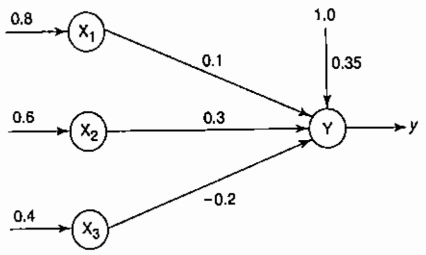

Figure 3 Neural net.

Solution: The given network has three input neurons with bias and one output neuron. These form a single-layer network. The inputs are given as $[x_1, x_2, x_3] = [0.8, 0.6, 0.4]$ and the weights are $[w_1, w_2, w_3] = [0.1, 0.3, -0.2]$ with bias $b = 0.35$ (its input is always 1).

The net input to the output neuron is

$$
\begin{aligned}
y_{in} &= b + \sum_{i=1}^{n} x_iw_i \\
&\quad [n = 3,\ \text{because only 3 input neurons are given}] \\
&= b + x_1w_1 + x_2w_2 + x_3w_3 \\
&= 0.35 + 0.8 \times 0.1 + 0.6 \times 0.3 \\
&\quad + 0.4 \times (-0.2) \\
&= 0.35 + 0.08 + 0.18 - 0.08 = 0.53
\end{aligned}
$$

(i) For binary sigmoidal activation function,

$$y = f(y_{in}) = \frac{1}{1+e^{-y_{in}}} = \frac{1}{1+e^{-0.53}} = 0.625$$

(ii) For bipolar sigmoidal activation function,

$$
\begin{aligned}
y = f(y_{in}) &= \frac{2}{1+e^{-y_{in}}} - 1 = \frac{2}{1+e^{-0.53}} - 1 \\
&= 0.259
\end{aligned}
$$

4. Implement AND function using McCulloch–Pitts neuron (take binary data).

Solution: Consider the truth table for AND function (Table 1).

Table 1

x_1	x_2	y
1	1	1
1	0	0
0	1	0
0	0	0

In McCulloch–Pitts neuron, only analysis is being performed. Hence, assume the weights be $w_1 = 1$ and $w_2 = 1$. The network architecture is shown in Figure 4. With these assumed weights, the net input is calculated for four inputs: For inputs

$$
\begin{aligned}
(1, 1),\quad & y_{in} = x_1w_1 + x_2w_2 = 1 \times 1 + 1 \times 1 = 2 \\
(1, 0),\quad & y_{in} = x_1w_1 + x_2w_2 = 1 \times 1 + 0 \times 1 = 1 \\
(0, 1),\quad & y_{in} = x_1w_1 + x_2w_2 = 0 \times 1 + 1 \times 1 = 1 \\
(0, 0),\quad & y_{in} = x_1w_1 + x_2w_2 = 0 \times 1 + 0 \times 1 = 0
\end{aligned}
$$

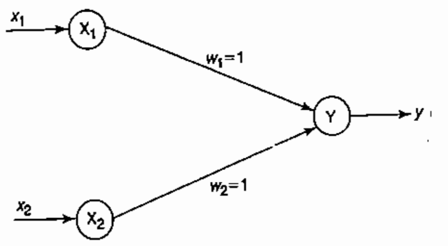

Figure 4 Neural net.

For an AND function, the output is high if both the inputs are high. For this condition, the net input is calculated as 2. Hence, based on this net input, the threshold is set, i.e. if the threshold value is greater than or equal to 2 then the neuron fires, else it does not fire. So the threshold value is set equal to $2(\theta = 2)$. This can also be obtained by

$$\theta \geqslant nw - p$$

Here, $n = 2$, $w = 1$ (excitatory weights) and $p = 0$ (no inhibitory weights). Substituting these values in the above-mentioned equation we get

$$\theta \geqslant 2 \times 1 - 0 \Rightarrow \theta \geqslant 2$$

Thus, the output of neuron Y can be written as

$$y = f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} \geqslant 2 \\ 0 & \text{if } y_{in} < 2 \end{cases}$$

where "2" represents the threshold value.

5. Implement ANDNOT function using McCulloch–Pitts neuron (use binary data representation).

Solution: In the case of ANDNOT function, the response is true if the first input is true and the second input is false. For all other input variations, the response is false. The truth table for ANDNOT function is given in Table 2.

Table 2

x_1	x_2	y
0	0	0
0	1	0
1	0	1
1	1	0

The given function gives an output only when $x_1 = 1$ and $x_2 = 0$. The weights have to be decided only after the analysis. The net can be represented as shown in Figure 5.

ANDNOT i0010

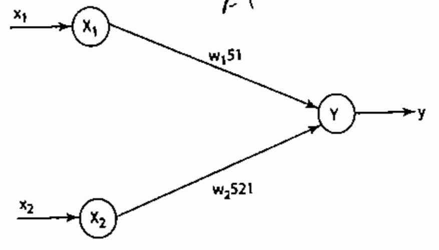

Figure 5 Neural net (weights fixed after analysis).

Case 1: Assume that both weights w_1 and w_2 are excitatory, i.e.,

$$w_1 = w_2 = 1$$

Then for the four inputs calculate the net input using

$$y_{in} = x_1w_1 + x_2w_2$$

For inputs

$$
\begin{aligned}
(1, 1),\quad & y_{in} = 1 \times 1 + 1 \times 1 = 2 \\
(1, 0),\quad & y_{in} = 1 \times 1 + 0 \times 1 = 1 \\
(0, 1),\quad & y_{in} = 0 \times 1 + 1 \times 1 = 1 \\
(0, 0),\quad & y_{in} = 0 \times 1 + 0 \times 1 = 0
\end{aligned}
$$

From the calculated net inputs, it is not possible to fire the neuron for input (1, 0) only. Hence, these weights are not suitable. after applying [illegible]

Assume one weight as excitatory and the other as inhibitory, i.e.,

$$w_1 = 1,\ w_2 = -1$$

Now calculate the net input. For the inputs

$$(1,1),\quad y_{in} = 1\times 1 + 1\times -1 = 0$$
$$(1,0),\quad y_{in} = 1\times 1 + 0\times -1 = 1$$
$$(0,1),\quad y_{in} = 0\times 1 + 1\times -1 = -1$$
$$(0,0),\quad y_{in} = 0\times 1 + 0\times -1 = 0$$

From the calculated net inputs, now it is possible to fire the neuron for input (1, 0) only by fixing a threshold of 1, i.e., $\theta \geq 1$ for Y unit. Thus,

$$w_1 = 1;\ w_2 = -1;\ \theta \geq 1$$

Note: The value of θ is calculated using the following:

$$\theta \geq nw - p$$
$$\theta \geq 2\times 1 - 1 \quad [\text{for "}p\text{" inhibitory only magnitude considered}]$$
$$\theta \geq 1$$

Thus, the output of neuron Y can be written as

$$y = f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} \geq 1 \\ 0 & \text{if } y_{in} < 1 \end{cases}$$

6. Implement XOR function using McCulloch–Pitts neuron (consider binary data).

Solution: The truth table for XOR function is given in Table 3.

Table 3

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

In this case, the output is "ON" for only odd number of 1's. For the rest it is "OFF." XOR function cannot be represented by simple and single logic function; it is represented as

$$y = x_1\bar{x}_2 + \bar{x}_1 x_2$$
$$y = z_1 + z_2$$

where

$$z_1 = x_1\bar{x}_2 \quad \text{(function 1)}$$
$$z_2 = \bar{x}_1 x_2 \quad \text{(function 2)}$$
$$y = z_1 (\text{OR}) z_2 \quad \text{(function 3)}$$

A single-layer net is not sufficient to represent the function. An intermediate layer is necessary.

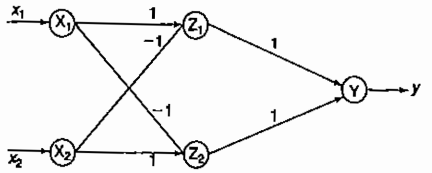

Figure 6 Neural net for XOR function (the weights shown are obtained after analysis).

- **First function** ($z_1 = x_1\bar{x}_2$): The truth table for function z_1 is shown in Table 4.

Table 4

x_1	x_2	z_1
0	0	0
0	1	0
1	0	1
1	1	0

The net representation is given as

Case 1: Assume both weights as excitatory, i.e.,

$$w_{11} = w_{21} = 1$$

Calculate the net inputs. For inputs,

$$(0,0),\ z_{1in} = 0\times 1 + 0\times 1 = 0$$
$$(0,1),\ z_{1in} = 0\times 1 + 1\times 1 = 1$$
$$(1,0),\ z_{1in} = 1\times 1 + 0\times 1 = 1$$
$$(1,1),\ z_{1in} = 1\times 1 + 1\times 1 = 2$$

Hence, it is not possible to obtain function z_1 using these weights.

Case 2: Assume one weight as excitatory and the other as inhibitory, i.e.,

$$w_{11} = 1;\quad w_{21} = -1$$

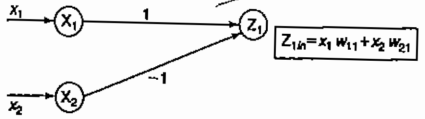

Figure 7 Neural net for Z_1.

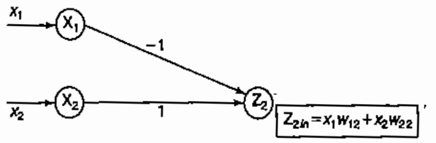

Figure 8 Neural net for Z_2.

Calculate the net inputs. For inputs 217324

$$(0,0),\ z_{1in} = 0\times 1 + 0\times -1 = 0$$
$$(0,1),\ z_{1in} = 0\times 1 + 1\times -1 = -1$$
$$(1,0),\ z_{1in} = 1\times 1 + 0\times -1 = 1$$
$$(1,1),\ z_{1in} = 1\times 1 + 1\times -1 = 0$$

On the basis of this calculated net input, it is possible to get the required output. Hence,

$$w_{11} = 1$$
$$w_{21} = -1$$
$$\theta \geq 1 \quad \text{for the } Z_1 \text{ neuron}$$

- **Second function** ($z_2 = \bar{x}_1 x_2$): The truth table for function z_2 is shown in Table 5.

Table 5

x_1	x_2	z_2
0	0	0
0	1	1
1	0	0
1	1	0

The net representation is given as follows:

Case 1: Assume both weights as excitatory, i.e.,

$$w_{12} = w_{22} = 1$$

Now calculate the net inputs. For the inputs

$$(0,0),\ z_{2in} = 0\times 1 + 0\times 1 = 0$$
$$(0,1),\ z_{2in} = 0\times 1 + 1\times 1 = 1$$
$$(1,0),\ z_{2in} = 1\times 1 + 0\times 1 = 1$$
$$(1,1),\ z_{2in} = 1\times 1 + 1\times 1 = 2$$

Hence, it is not possible to obtain function z_2 using these weights.

Case 2: Assume one weight as excitatory and the other as inhibitory, i.e.,

$$w_{12} = -1;\quad w_{22} = 1$$

Now calculate the net inputs. For the inputs

$$(0,0),\ z_{2in} = 0\times -1 + 0\times 1 = 0$$
$$(0,1),\ z_{2in} = 0\times -1 + 1\times 1 = 1$$
$$(1,0),\ z_{2in} = 1\times -1 + 0\times 1 = -1$$
$$(1,1),\ z_{2in} = 1\times -1 + 1\times 1 = 0$$

Thus, based on this calculated net input, it is possible to get the required output. Hence,

$$w_{12} = -1$$
$$w_{22} = 1$$
$$\theta \geq 1 \quad \text{for the } Z_2 \text{ neuron}$$

SIV

- **Third function** ($y = z_1$ OR z_2): The truth table for this function is shown in Table 6.

Table 6

x_1	x_2	y	z_1	z_2
0	0	0	0	0
0	1	1	0	1
1	0	1	1	0
1	1	0	0	0

Here the net input is calculated using

$$y_{in} = z_1 v_1 + z_2 v_2$$

Case 1: Assume both weights as excitatory, i.e.,

$$v_1 = v_2 = 1$$

Now calculate the net input. For inputs

$$(0,0),\ y_{in} = 0\times 1 + 0\times 1 = 0$$
$$(0,1),\ y_{in} = 0\times 1 + 1\times 1 = 1$$
$$(1,0),\ y_{in} = 1\times 1 + 0\times 1 = 1$$
$$(1,1),\ y_{in} = 0\times 1 + 0\times 1 = 0$$

(because for $x_1 = 1$ and $x_2 = 1, z_1 = 0$ and $z_2 = 0$)

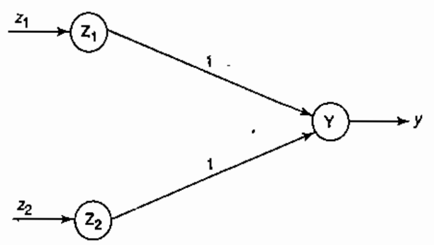

Figure 9 Neural net for $Y(Z_1 \text{ OR } Z_2)$.

Setting a threshold of $\theta \geq 1$, $v_1 = v_2 = 1$, which implies that the net is recognized. Therefore, the analysis is made for XOR function using M–P neurons. Thus for XOR function, the weights are obtained as

$$w_{11} = w_{22} = 1 \quad \text{(excitatory)}$$
$$w_{12} = w_{21} = -1 \quad \text{(inhibitory)}$$
$$v_1 = v_2 = 1 \quad \text{(excitatory)}$$

7. Using the linear separability concept, obtain the response for OR function (take bipolar inputs and bipolar targets).

Solution: Table 7 is the truth table for OR function with bipolar inputs and targets.

Table 7

x_1	x_2	y
1	1	1
1	−1	1
−1	1	1
−1	−1	−1

The truth table inputs and corresponding outputs have been plotted in Figure 10. If output is 1, it is denoted as "+" else "−." Assuming the coordinates as (−1, 0) and (0, −1); (x_1, y_1) and (x_2, y_2), the slope "m" of the straight line can be obtained as

$$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{-1-0}{0+1} = \frac{-1}{1} = -1$$

We now calculate c:

$$c = y_1 - mx_1 = 0 - (-1)(-1) = -1$$

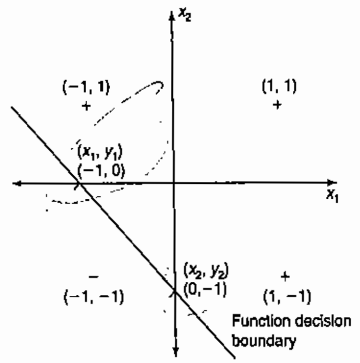

Figure 10 Graph for 'OR' function.

Using this value the equation for the line is given as

$$y = mx + c = (-1)x - 1 = -x - 1$$

Here the quadrants are not x and y but x_1 and x_2, so the above equation becomes

$$x_2 = -x_1 - 1 \tag{2.1}$$

This can be written as

$$x_2 = \frac{-w_1}{w_2}x_1 - \frac{b}{w_2} \tag{2.2}$$

Comparing Eqs. (2.1) and (2.2), we get

$$\frac{w_1}{w_2} = \frac{1}{1}; \quad \frac{b}{w_2} = \frac{1}{1}$$

Therefore, $w_1 = 1$, $w_2 = 1$ and $b = 1$. Calculating the net input and output of OR function on the basis of these weights and bias, we get entries in Table 8.

Table 8

x_1	x_2	b	$y_{in} = b + x_1w_1 + x_2w_2$	y
1	1	1	3	1
1	−1	1	1	1
−1	1	1	1	1
−1	−1	1	−1	−1

Thus, the output of neuron Y can be written as

$$y = f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} \geqslant 1 \\ 0 & \text{if } y_{in} < 1 \end{cases}$$

where the threshold is taken as "1" ($\theta = 1$) based on the calculated net input. Hence, using the linear separability concept, the response is obtained for "OR" function.

8. Design a Hebb net to implement logical AND function (use bipolar inputs and targets).

Solution: The training data for the AND function is given in Table 9.

Table 9

Inputs			Target
x_1	x_2	b	y
1	1	1	1
1	−1	1	−1
−1	1	1	−1
−1	−1	1	−1

The network is trained using the Hebb network training algorithm discussed in Section 2.7.3. Initially the weights and bias are set to zero, i.e.,

$$w_1 = w_2 = b = 0$$

- First input $[x_1\ x_2\ b] = [1\ 1\ 1]$ and target $= 1$ [i.e., $y = 1$]: Setting the initial weights as old weights and applying the Hebb rule, we get

$$w_i(\text{new}) = w_i(\text{old}) + x_iy$$
$$w_1(\text{new}) = w_1(\text{old}) + x_1y = 0 + 1 \times 1 = 1$$
$$w_2(\text{new}) = w_2(\text{old}) + x_2y = 0 + 1 \times 1 = 1$$
$$b(\text{new}) = b(\text{old}) + y = 0 + 1 = 1$$

The weights calculated above are the final weights that are obtained after presenting the first input. These weights are used as the initial weights when the second input pattern is presented. The weight change here is $\Delta w_i = x_iy$. Hence weight changes relating to the first input are

$$\Delta w_1 = x_1y = 1 \times 1 = 1$$
$$\Delta w_2 = x_2y = 1 \times 1 = 1$$
$$\Delta b = y = 1$$

- Second input $[x_1\ x_2\ b] = [1\ -1\ 1]$ and $y = -1$: The initial or old weights here are the final (new) weights obtained by presenting the first input pattern, i.e.,

$$[w_1\ w_2\ b] = [1\ 1\ 1]$$

The weight change here is

$$\Delta w_1 = x_1y = 1 \times -1 = -1$$
$$\Delta w_2 = x_2y = -1 \times -1 = 1$$
$$\Delta b = y = -1$$

The new weights here are

$$w_1(\text{new}) = w_1(\text{old}) + \Delta w_1 = 1 - 1 = 0$$
$$w_2(\text{new}) = w_2(\text{old}) + \Delta w_2 = 1 + 1 = 2$$
$$b(\text{new}) = b(\text{old}) + \Delta b = 1 - 1 = 0$$

Similarly, by presenting the third and fourth input patterns, the new weights can be calculated. Table 10 shows the values of weights for all inputs.

Table 10

Inputs				Weight changes			Weights		
x_1	x_2	b	y	Δw_1	Δw_2	Δb	w_1	w_2	b
							(0	0	0)
1	1	1	1	1	1	1	1	1	1
1	−1	1	−1	−1	1	−1	0	2	0
−1	1	1	−1	1	−1	−1	1	1	−1
−1	−1	1	−1	1	1	−1	2	2	−2

The separating line equation is given by

$$x_2 = \frac{-w_1}{w_2}x_1 - \frac{b}{w_2}$$

For all inputs, use the final weights obtained for each input to obtain the separating line. For the first input [1 1 1], the separating line is given by

$$x_2 = \frac{-1}{1}x_1 - \frac{1}{1} \Rightarrow x_2 = -x_1 - 1$$

Similarly, for the second input [1 −1 1], the separating line is

$$x_2 = \frac{-0}{2}x_1 - \frac{0}{2} \Rightarrow x_2 = 0$$

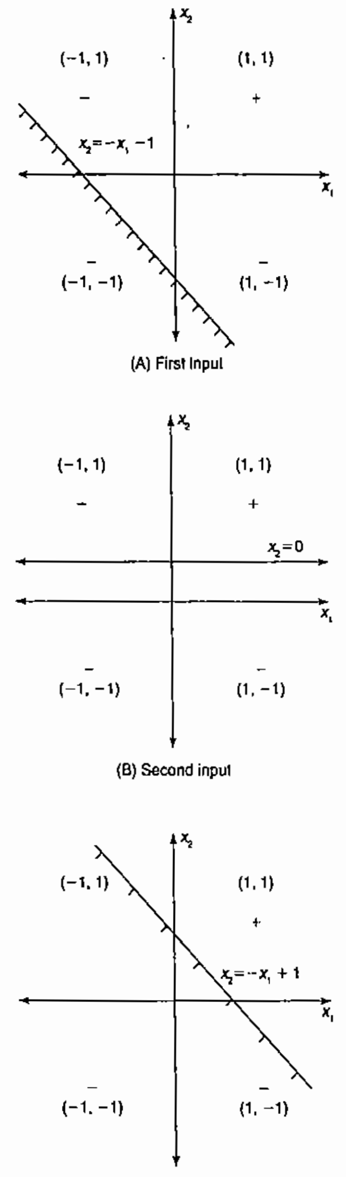

Figure 11 Decision boundary for AND function using Hebb rule for each training pair.

For the third input [−1 1 1], it is

$$x_2 = \frac{-1}{1}x_1 + \frac{1}{1} \Rightarrow x_2 = -x_1 + 1$$

Finally, for the fourth input [−1 − 1 1], the separating line is

$$x_2 = \frac{-2}{2}x_1 + \frac{2}{2} \Rightarrow x_2 = -x_1 + 1$$

The graphs for each of these separating lines obtained are shown in Figure 11. In this figure "+" mark is used for output "1" and "−" mark is used for output "−1." From Figure 11, it can be noticed that for the first input, the decision boundary differentiates only the first and fourth inputs, and not all negative responses are separated from positive responses. When the second input pattern is presented, the decision boundary separates (1, 1) from (1, −1) and (−1, −1) and not (−1, 1). But the boundary line is same for the both third and fourth training pairs. And, the decision boundary line obtained from these input training pairs separates the positive response region from the negative response region. Hence, the weights obtained from this are the final weights and are given as

$$w_1 = 2; \quad w_2 = 2; \quad b = -2$$

The network can be represented as shown in Figure 12.

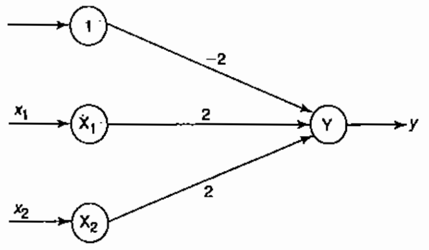

Figure 12 Hebb net for AND function.

9. Design a Hebb net to implement OR function (consider bipolar inputs and targets).

Solution: The training pair for the OR function is given in Table 11.

Table 11

Inputs			Target
x_1	x_2	b	y
1	1	1	1
1	−1	1	1
−1	1	1	1
−1	−1	1	−1

Initially the weights and bias are set to zero, i.e.,

$$w_1 = w_2 = b = 0$$

The network is trained and the final weights are outlined using the Hebb training algorithm discussed in Section 2.7.3. The weights are considered as final weights if the boundary line obtained from these weights separates the positive response region and negative response region.

By presenting all the input patterns, the weights are calculated. Table 12 shows the weights calculated for all the inputs.

Table 12

Inputs				Weight changes			Weights		
x_1	x_2	b	y	Δw_1	Δw_2	Δb	w_1	w_2	b
							(0	0	0)
1	1	1	1	1	1	1	1	1	1
1	−1	1	1	1	−1	1	2	0	2
−1	1	1	1	−1	1	1	1	1	3
−1	−1	1	−1	1	1	−1	2	2	2

Using the final weights, the boundary line equation can be obtained. The separating line equation is

$$x_2 = \frac{-w_1}{w_2}x_1 - \frac{b}{w_2} = \frac{-2}{2}x_1 - \frac{2}{2} = -x_1 - 1$$

The decision region for this net is shown in Figure 13. It is observed in Figure 13 that straight line $x_2 = -x_1 - 1$ separates the pattern space into two regions. The input patterns [(1, 1), (1, −1), (−1, 1)] for which the output response is "1" lie on one side of the boundary, and the input pattern (−1, −1) for which the output response is "−1" lies on the other side of the boundary. Thus, the final weights are

$$w_1 = 2; \quad w_2 = 2; \quad b = 2$$

final weights are used to draw separable line

The network can be represented as shown in Figure 14.

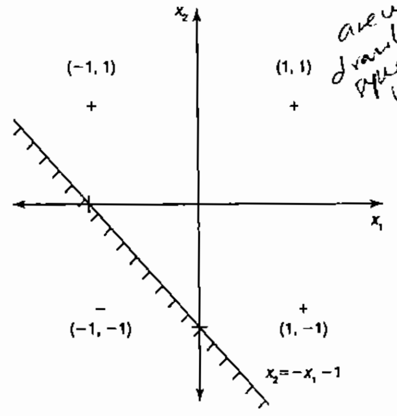

Figure 13 Decision boundary for OR function.

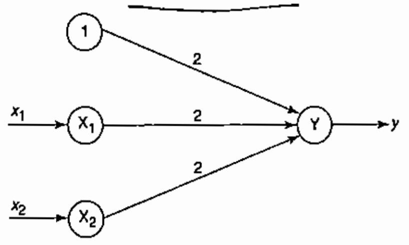

Figure 14 Hebb net for OR function.

10. Use the Hebb rule method to implement XOR function (take bipolar inputs and targets).

Solution: The training patterns for an XOR function are shown in Table 13.

Table 13

Inputs			Target
x_1	x_2	b	y
1	1	1	−1
1	−1	1	1
−1	1	1	1
−1	−1	1	−1

Here, a single-layer network with two input neurons, one bias and one output neuron is considered. In this case also, the initial weights are assumed to be zero:

$$w_1 = w_2 = b = 0$$

By using the Hebb training algorithm, the network is trained and the final weights are calculated as shown in the following Table 14.

Table 14

Inputs				Weight changes			Weights		
x_1	x_2	b	y	Δw_1	Δw_2	Δb	w_1 (0	w_2 0	b 0)
1	1	1	−1	−1	−1	−1	−1	−1	−1
1	−1	1	1	1	−1	1	0	−2	0
−1	1	1	1	−1	1	1	−1	−1	1
−1	−1	1	−1	1	1	−1	0	0	0

The final weights obtained after presenting all the input patterns do not give correct output for all patterns. Figure 15 shows that the input patterns are linearly non-separable. The graph shown in Figure 15 indicates that the four input pairs that are present cannot be divided by a single line to separate them into two regions. Thus XOR function is a case of a pattern classification problem, which is not linearly separable.

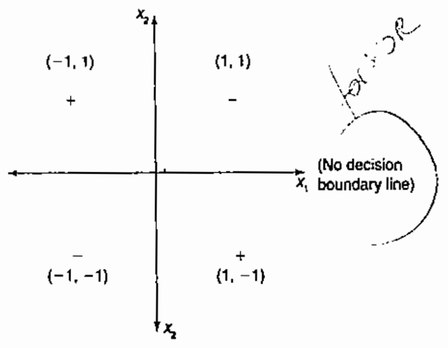

Figure 15 Decision boundary for XOR function.

The XOR function can be made linearly separable by solving it in a manner as discussed in Problem 6. This method of solving will result in two decision boundary lines for separating positive and negative regions of XOR function.

11. Using the Hebb rule, find the weights required to perform the following classifications of the given input patterns shown in Figure 16. The pattern is shown as 3 × 3 matrix form in the squares. The "+" symbols represent the value "1" and empty squares indicate "−1." Consider "I" belongs to the members of class (so has target value 1) and "O" does not belong to the members of class (so has target value −1).

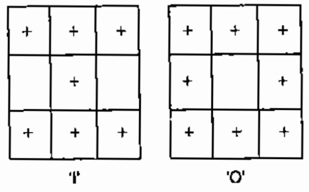

Figure 16 Data for input patterns.

Solution: The training input patterns for the given net (Figure 16) are indicated in Table 15.

Table 15

Pattern	Inputs										Target
	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	b	y
I	1	1	1	−1	1	−1	1	1	1	1	1
O	1	1	1	1	−1	1	1	1	1	1	−1

Here a single-layer network with nine input neurons, one bias and one output neuron is formed. Set the initial weights and bias to zero, i.e.,

$$w_1 = w_2 = w_3 = w_4 = w_5$$
$$= w_6 = w_7 = w_8 = w_9 = b = 0$$

Case 1: Presenting first input pattern (I), we calculate change in weights:

$$\Delta w_i = x_i y, \quad i = 1 \text{ to } 9$$
$$\Delta w_1 = x_1 y = 1 \times 1 = 1$$

$$\Delta w_2 = x_2 y = 1 \times 1 = 1$$
$$\Delta w_3 = x_3 y = 1 \times 1 = 1$$
$$\Delta w_4 = x_4 y = -1 \times 1 = -1$$
$$\Delta w_5 = x_5 y = 1 \times 1 = 1$$
$$\Delta w_6 = x_6 y = -1 \times 1 = -1$$
$$\Delta w_7 = x_7 y = 1 \times 1 = 1$$
$$\Delta w_8 = x_8 y = 1 \times 1 = 1$$
$$\Delta w_9 = x_9 y = 1 \times 1 = 1$$
$$\Delta b = y = 1$$

We now calculate the new weights using the formula

$$w_i(\text{new}) = w_i(\text{old}) + \Delta w_i$$

Setting the old weights as the initial weights here, we obtain

$$w_1(\text{new}) = w_1(\text{old}) + \Delta w_1 = 0 + 1 = 1$$
$$w_2(\text{new}) = w_2(\text{old}) + \Delta w_2 = 0 + 1 = 1$$
$$w_3(\text{new}) = w_3(\text{old}) + \Delta w_3 = 0 + 1 = 1$$

Similarly, calculating for other weights we get

$w_4(\text{new}) = -1$, $w_5(\text{new}) = 1$, $w_6(\text{new}) = -1$, $w_7(\text{new}) = 1$, $w_8(\text{new}) = 1$, $w_9(\text{new}) = 1$, $b(\text{new}) = 1$

The weights after presenting first input pattern are

$$W_{(\text{new})} = [1\ 1\ 1\ -1\ 1\ -1\ 1\ 1\ 1\ 1]$$

Case 2: Now we present the second input pattern (O). The initial weights used here are the final weights obtained after presenting the first input pattern. Here, the weights are calculated as shown below ($y = -1$ with the initial weights being $[1\ 1\ 1\ -1\ 1\ -1\ 1\ 1\ 1\ 1]$).

$$w_i(\text{new}) = w_i(\text{old}) + \Delta x_i \qquad [\Delta w_i = x_i y]$$
$$w_1(\text{new}) = w_1(\text{old}) + x_1 y = 1 + 1 \times -1 = 0$$
$$w_2(\text{new}) = w_2(\text{old}) + x_2 y = 1 + 1 \times -1 = 0$$
$$w_3(\text{new}) = w_3(\text{old}) + x_3 y = 1 + 1 \times -1 = 0$$
$$w_4(\text{new}) = w_4(\text{old}) + x_4 y = -1 + 1 \times -1 = -2$$
$$w_5(\text{new}) = w_5(\text{old}) + x_5 y = 1 + -1 \times -1 = 2$$
$$w_6(\text{new}) = w_6(\text{old}) + x_6 y = -1 + 1 \times -1 = -2$$
$$w_7(\text{new}) = w_7(\text{old}) + x_7 y = 1 + 1 \times -1 = 0$$
$$w_8(\text{new}) = w_8(\text{old}) + x_8 y = 1 + 1 \times -1 = 0$$
$$w_9(\text{new}) = w_9(\text{old}) + x_9 y = 1 + 1 \times -1 = 0$$
$$b(\text{new}) = b(\text{old}) + y = 1 + 1 \times -1 = 0$$

The final weights after presenting the second input pattern are given as

$$W_{(\text{new})} = [0\ 0\ 0\ -2\ 2\ -2\ 0\ 0\ 0\ 0]$$

The weights obtained are indicated in the Hebb net shown in Figure 17.

12. Find the weights required to perform the following classifications of given input patterns using the Hebb rule. The inputs are "1" where "+" symbol is present and "−1" where "." is present. "L" pattern belongs to the class (target value +1) and "U" pattern does not belong to the class (target value −1).

Solution: The training input patterns for Figure 18 are given in Table 16.

Table 16

Pattern	Inputs										Target
	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	b	y
L	1	−1	−1	1	−1	−1	1	1	1	1	1
U	1	−1	1	1	−1	1	1	1	1	1	−1

A single-layer network with nine input neurons, one bias and one output neuron is formed. Set the initial weights and bias to zero, i.e.,

$$w_1 = w_2 = w_3 = w_4 = w_5$$
$$= w_6 = w_7 = w_8 = w_9 = b = 0$$

The weights are calculated using

$$w_i(\text{new}) = w_i(\text{old}) + x_i y$$

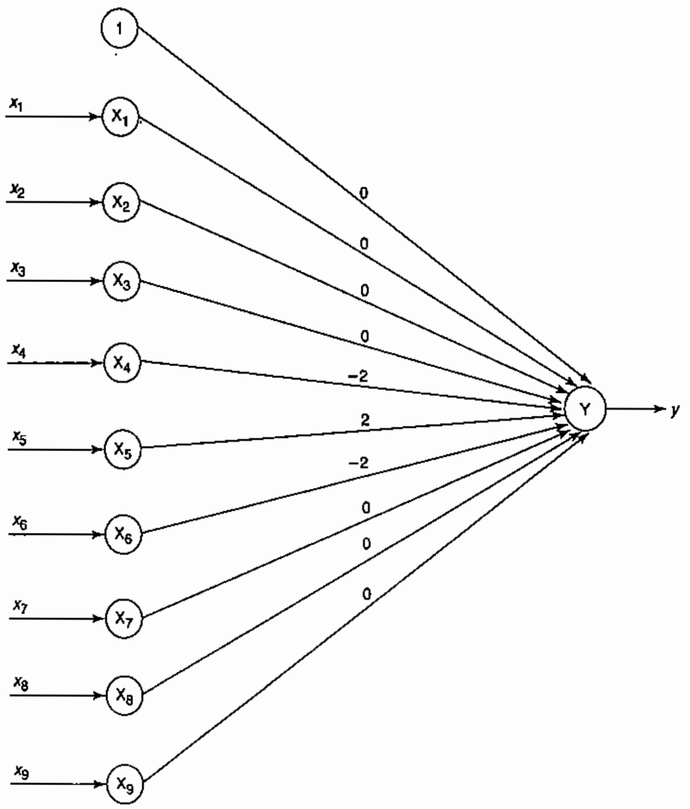

Figure 17 Hebb net for the data matrix shown in Figure 16.

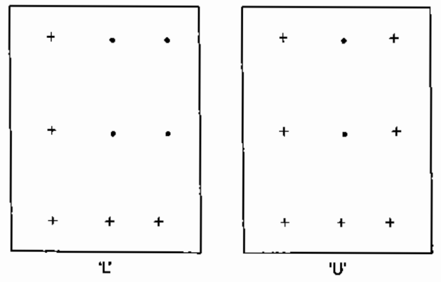

Figure 18 Input data for given patterns.

The calculated weights are given in Table 17.

Table 17

Inputs										Target	Weights									
x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	b	y	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	b
											(0	0	0	0	0	0	0	0	0	0)
1	−1	−1	1	−1	−1	1	1	1	1	1	1	−1	−1	1	−1	−1	1	1	1	1
1	−1	1	1	−1	1	1	1	1	1	−1	0	0	−2	0	0	−2	0	0	0	0

The final weights after presenting the two input patterns are

$$W_{(\text{new})} = [0\ 0\ -2\ 0\ 0\ -2\ 0\ 0\ 0\ 0]$$

The obtained weights are indicated in the Hebb net shown in Figure 19.

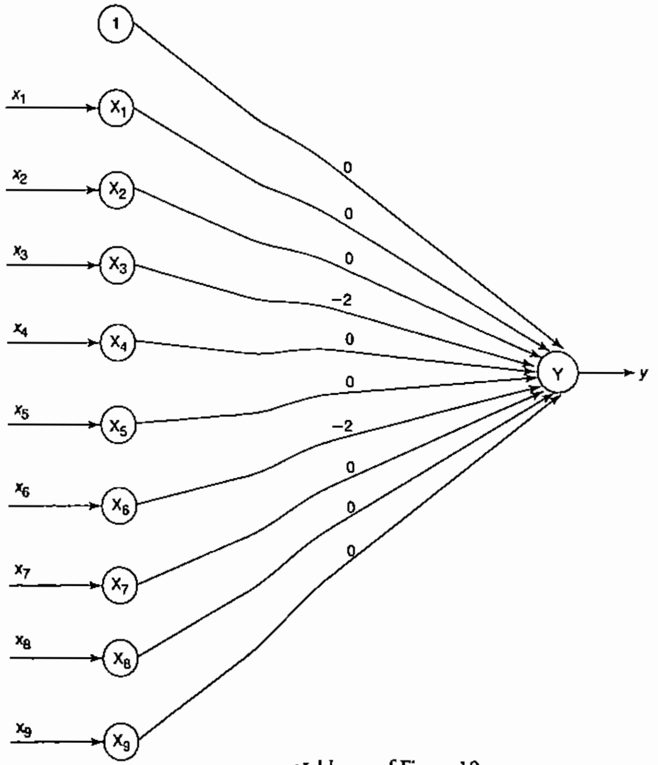

Figure 19 Hebb net of Figure 18.

2.10 Review Questions

1. Define an artificial neural network.
2. State the properties of the processing element of an artificial neural network.
3. How many signals can be sent by a neuron at a particular time instant?
4. Draw a simple artificial neuron and discuss the calculation of net input.
5. What is the influence of a linear equation over the net input calculation?
6. List the main components of the biological neuron.
7. Compare and contrast biological neuron and artificial neuron.
8. State the characteristics of an artificial neural network.
9. Discuss in detail the historical development of artificial neural networks.
10. What are the basic models of an artificial neural network?
11. Define net architecture and give its classifications.
12. Define learning.
13. Differentiate between supervised and unsupervised learning.
14. How is the critic information used in the learning process?
15. What is the necessity of activation function?
16. List the commonly used activation functions.
17. What is the impact of weight in an artificial neural network?
18. What is the other name for weight?
19. Define bias and threshold.
20. What is a learning rate parameter?
21. How does a momentum factor make faster convergence of a network?
22. State the role of vigilance parameter in ART network.
23. Why is the McCulloch–Pitts neuron widely used in logic functions?
24. Indicate the difference between excitatory and inhibitory weighted interconnections.
25. Define linear separability.
26. Justify – XOR function is non-linearly separable by a single decision boundary line.
27. How can the equation of a straight line be formed using linear separability?
28. In what ways is bipolar representation better than binary representation?
29. State the training algorithm used for the Hebb network.
30. Compare feed-forward and feedback network.

2.11 Exercise Problems

1. For the network shown in Figure 20, calculate the net input to the output neuron.

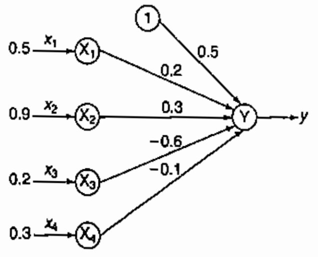

Figure 20 Neural net.

2. Calculate the output of neuron Y for the net shown in Figure 21. Use binary and bipolar sigmoidal activation functions.

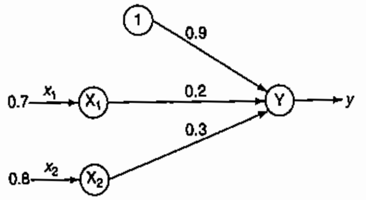

Figure 21 Neural net.

3. Design neural networks with only one M-P neuron that implements the three basic logic operations:
   (i) NOT (x_1);
   (ii) OR (x_1, x_2);
   (iii) NAND (x_1, x_2), where x_1 and $x_2 \in \{0, 1\}$.
4. (a) Show that the derivative of unipolar sigmoidal function is
$$f'(x) = \lambda f(x)[1 - f(x)]$$
   (b) Show that the derivative of bipolar sigmoidal function is
$$f'(x) = \frac{\lambda}{2}[1 + f(x)][1 - f(x)]$$
5. (a) Construct a feed-forward network with five input nodes, three hidden nodes and four output nodes that has lateral inhibition structure in the output layer.
   (b) Construct a recurrent network with four input nodes, three hidden nodes and two output nodes that has feedback links from the hidden layer to the input layer.
6. Using linear separability concept, obtain the response for NAND function.
7. Design a Hebb net to implement logical AND function with
   (a) binary inputs and targets and
   (b) binary inputs and bipolar targets.
8. Implement NOR function using Hebb net with
   (a) bipolar inputs and targets and
   (b) bipolar inputs and binary targets.
9. Classify the input patterns shown in Figure 22 using Hebb training algorithm.

```
   +    +   +          +    +   +
   +    •   +          +    •   •
   +    +   +          +    +   +
   +    •   +          +    •   •
   +    •   +          +    +   +
       'A'                 'E'
Target value +1            −1
```

Figure 22 Input pattern.

10. Using Hebb rule, find the weights required to perform following classifications. The vectors $(1\ -1\ 1\ -1)$ and $(1\ 1\ 1\ -1)$ belong to class (target value $+1$); vectors $(-1\ -1\ 1\ 1)$ and $(1\ 1\ -1\ -1)$ do not belong to class (target value -1). Also using each of training x vectors as input, test the response of net.

2.12 Projects

1. Write a program to classify the letters and numerals using Hebb learning rule. Take a pair of letters or numerals of your own. Also, after training the network, test the response of the net using suitable activation function. Perform the classification using bipolar data as well as binary data.
2. Write suitable programs for implementing logic functions using McCulloch–Pitts neuron.
3. Write a computer program to train a Madaline to perform AND function, using MRI algorithm.
4. Write a program for implementing BPN for training a single-hidden-layer back-propagation