CHAPTER III

MULTILAYER FEEDFORWARD ARTIFICIAL NEURAL NETWORK

3.1 INTRODUCTION

In Chapter II, we presented the basic theory of a single layer neural network and the associated training rules. We implemented these networks for computation of logical operations 'AND', 'OR', and 'NOT'.

An important function of ANN is pattern recognition and classification. This Chapter is concerned with how ANN models can be used in classification problems. As is well-known, the problem of classification is one of the basic problems in any scientific research (Kendall, 1980). There are many statistical tools such as, Discriminant Analysis, Cluster Analysis etc. which are available for such problems. Recently, ANN's are also being used widely as an alternative technique to statistical tools for classification problems.

In Section 2 of this Chapter, we discuss the use of single layer ANN as a tool for classification purpose and some of its limitations on what it can classify. Further, we will discuss a generalization called "Multilayer Network" to overcome the

limitations of single layer ANN, followed by the training method called 'Back-propagation training method' in Section 3. Examples are given to illustrate the theory discussed in this Chapter. Finally, Section 4 describes significance of various network parameters.

3.2 ANN AND CLASSIFICATION

The problem of computing 'AND', 'OR', and 'NOT' functions discussed in previous Chapter can also be viewed as classification problems. For example, consider the computation of `AND' function. Here we have a set of four input patterns (x_1, x_2) :

$$\mathcal{X} = \left\{ (x_1, x_2) : (0,0),\ (0,1),\ (1,0),\ (1,1) \right\}$$

The operation x_1.AND.x_2 produces either value 0 or 1. Define two classes π_1 and π_2 as follows :

$$\pi_1 = \left\{ (x_1, x_2) : x_1.\text{AND}.x_2 = 1 \right\}$$

$$\pi_2 = \left\{ (x_1, x_2) : x_1.\text{AND}.x_2 = 0 \right\}$$

Thus, we see that the patterns in the above set belong to either class π_1 or π_2. The four patterns are shown graphically in Fig. 3.1.

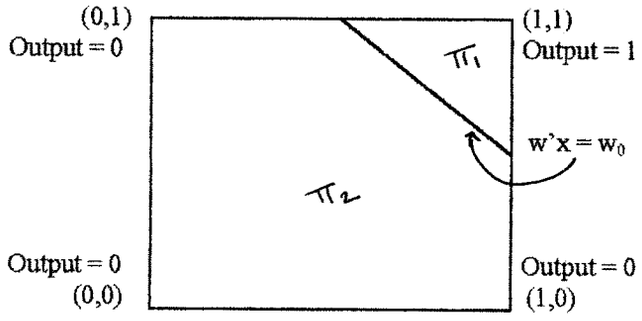

Figure 3.1 Graphical Representation of 'AND' Function

From the above Figure, it is clear that, a line $\underline{w}'\underline{x} = w_o$ for some suitable w_o, separates four patterns based on their outputs into class π_1 and π_2. The above discussion leads to formation of a classifier or rule for classifying the four patterns into class π_1 and π_2 as follows :

$$\text{If } \underline{w}'\underline{x} > w_o \text{ then a pattern } \underline{x} \text{ belongs to } \pi_1$$

$$\text{If } \underline{w}'\underline{x} \leq w_o \text{ then a pattern } \underline{x} \text{ belongs to } \pi_2$$

(3.2.1)

Obviously, the value of $\underline{w}$ and w_o are to be selected suitably so that the patterns are classified into appropriate classes correctly. Note that $\underline{w}'\underline{x}$ is linear in nature. Hence $\underline{w}'\underline{x}$ can be called 'linear classifier'. In the problems wherein if a linear function $\underline{w}'\underline{x}$ exists for classifying a set of patterns, the patterns are said to be 'linearly separable patterns' A more

general definition for R classes is given below. Before doing so, we will reexpress (3.2.1) in the matrix notations for later use.

In general, let $\mathcal{X} = \{\underline{x}_1, \underline{x}_2, \ldots, \underline{x}_p\}$ be a set of patterns belonging to either class π_1 or π_2. Then, we rewrite (3.2.1) as

$$\underline{w}' \underline{x}_i = \begin{cases} > 0, & \underline{x}_i \in \pi_1 \quad i = 1,2,\ldots,P \\ \leq 0, & \underline{x}_i \in \pi_2 \quad i = 1,2,\ldots,P \end{cases} \tag{3.2.2}$$

where the first component of $\underline{w}$ and $\underline{x}_i$ are w_o and -1 respectively. Further, set

$$\underline{x}_i = -\underline{x}_i, \text{ whenever } \underline{x}_i \in \pi_2, \quad i = 1,2,\ldots P. \tag{3.2.3}$$

Then (3.2.2) can equivalently be expressed as follows

$$\underline{x}_i' \underline{w} > 0 , \quad i = 1,2,\ldots,P , \tag{3.2.4}$$

Or in a matrix notation, (3.2.4) can be written as

$$A\underline{w} > \underline{0} \tag{3.2.5}$$

where A is a matrix of 'converted' set of patterns i.e. using (3.2.3),

$$A = \begin{bmatrix} \underline{x}_1' \\ \underline{x}_2' \\ \\ \underline{x}_P' \end{bmatrix}$$

Definition 3.1 Linear Separability (Zurada, 1992)

Let $\mathcal{X} = \{ \underline{x}_1, \underline{x}_2, \ldots, \underline{x}_P \}$ be a set of P patterns. Let the set be divided into non-overlapping R subsets say $\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_R$. If there exists a linear classifier of the type $\underline{w}'\underline{x}$ that classifies the patterns from $\mathcal{X}_i$ as belonging to class i for i = 1,2, ...,R then the patterns are said to be linearly separable.

Now, we will illustrate the above concept with the help of 'OR' function :

let $\mathcal{X} = \left\{ (x_1, x_2) : (0,0), (0,1), (1,0), (1,1) \right\}$ be a set of four input patterns. Consider the operation $x_1.OR.x_2$. The graphical representation of the outputs of $(x_1.OR.x_2)$ is shown below :

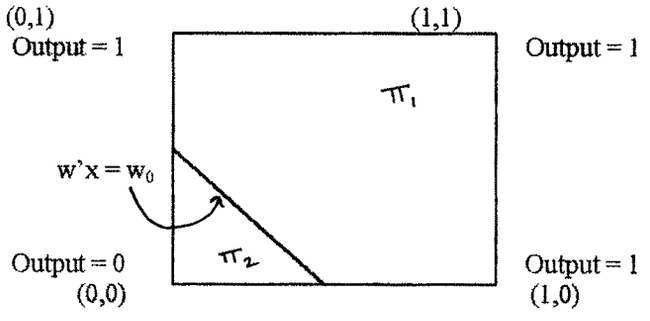

Figure 3.2 Graphical Representation of 'OR' Function

Note that the above set is clearly linearly separable.

Obviously, not all sets of patterns are linearly separable. For instance, consider the problem of computing exclusive OR

(XOR). Let (x_1, x_2) be the input pair each takes value either 0 or 1, and the corresponding output of x_1.XOR.x_2 is given in Table 3.1

TABLE 3.1

x_1	x_2	x_1.XOR. x_2	Class
0	0	0	π_2
0	1	1	π_1
1	0	1	π_1
1	1	0	π_2

Here, input pairs (0,0) and (1,1) belong to class π_2 and (0,1) and (1,0) belong to class π_1. The patterns are shown diagrammatically in Fig. 3.3

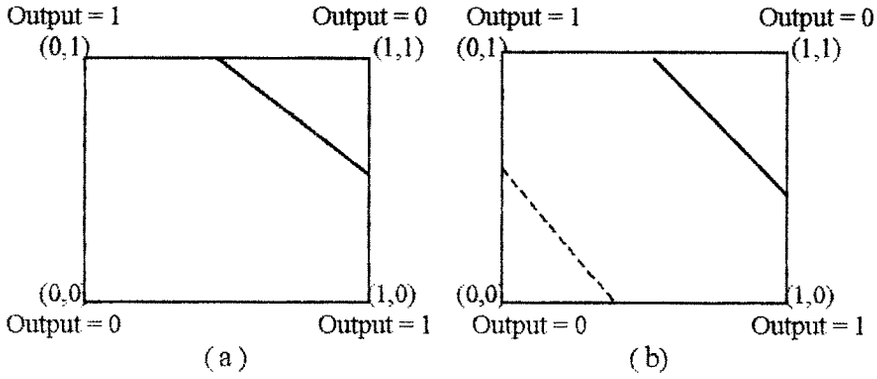

Figure 3.3 'XOR' - Problem

From Figs. 3.3(a) and Fig.3.3(b), it can be observed that the given patterns are not linearly separable, for the reason

that there does not exist any single line which can discriminate between π_1 and π_2. However, if one more line (shown by dotted lines) in Fig.3.3(b) is drawn, then two lines together separate the patterns.

The classification problems of this type are called as 'linearly nonseparable' or 'nonlinearly separable'. Clearly XOR is a linearly nonseparable classification problem. In such cases, finding a 'suitable' classifier is quite a difficult task.

Before proceeding further, below we prove a theorem (Schalkoff, 1997) which is useful to determine whether a given set of patterns is linearly separable or not.

Theorem 3.1 : Let $\mathcal{X} = \{ \underline{x}_1, \underline{x}_2, \ldots, \underline{x}_P \}$ be a set of P patterns. Then, $\mathcal{X}$ is linearly nonseparable if there exists constant (scalar) $q_i \geq 0$, $i = 1,2,\ldots,P$, such that

$$\sum_{i=1}^{P} q_i \underline{x}_i = \underline{0} \qquad (3.2.6)$$

with at least one $q_i > 0$, $i = 1,2,\ldots,P$.

Proof : If possible, suppose that the set

$$\mathcal{X} = \left\{ \underline{x}_1, \underline{x}_2, \ldots \underline{x}_P \right\}$$

is linearly separable set . Then using (3.2.4)

we get

$$\underline{x}_i' \underline{w} > 0 \ , \qquad i = 1,2,\ldots,P \ , \tag{3.2.7}$$

Now, assume that there exist some $q_i \geq 0$, $i = 1,2,\ldots,p$ and at least one $q_i > 0$ such that

$$\sum_{i=1}^{P} q_i \underline{x}_i = \underline{0} \ , \tag{3.2.8}$$

On premultiplying (3.2.8) by $\underline{w}'$, we obtain

$$\underline{w}' \sum_{i=1}^{P} q_i \underline{x}_i = 0 \ , \tag{3.2.9}$$

which implies

$$\sum_{i=1}^{P} q_i \underline{x}_i' \underline{w} = 0 \ , \tag{3.2.10}$$

Since

$$\underline{x}_i' \underline{w} > 0 \ , \qquad i = 1,2,\ldots,P$$

we get

$$q_i = 0, \quad i = 1,2,\ldots,P$$

which contradicts the assumption that $\mathcal{X}$ is linearly separable. Hence, the theorem.

□

Now, we give two examples to demonstrate the utility of the above theorem.

Example 3.1 : 'XOR' Function

The Table 3.2 gives the initial and converted set (using (3.2.3)) of patterns (x_1, x_2) required for computing XOR function.

The vectors have been preprocessed using Eq.(3.2.2)

TABLE 3.2

Initial Set Input $(x_1\ x_2\ x_0)$			Output (d)	Converted Set Input $(x_1\ x_2\ x_0)$		
0	0	-1	0	0	0	1
0	1	-1	1	0	1	-1
1	0	-1	1	1	0	-1
1	1	-1	0	-1	-1	1

Looking at the converted set of Table 3.2, we note that, there exists a set of coefficients $q_i=1$, $i = 1,...4$ that satisfies Eq.(3.2.6). Thus, this set is clearly nonlinearly separable.

Example 3.2 : 'OR' Function.

In a similar fashion, we give the set (x_1,x_2)'s for OR Function in Table 3.3.

TABLE 3.3

Initial Set Input $(x_1\ x_2\ x_0)$			Output (d)	Converted Set Input $(x_1\ x_2\ x_0)$		
0	0	-1	0	0	0	1
0	1	-1	1	0	1	-1
1	0	-1	1	1	0	-1
1	1	-1	1	1	1	-1

Note that there is no positive set of coefficients(for converted set) that satisfies Eq.(3.2.6). Hence, this set is linearly separable.

As mentioned earlier, the ANN models which are discussed in Chapter II are useful for certain types of classification problems. Below, through a theorem, we obtain a condition under which a single layer ANN can classify the given set of patterns and thus such a model can be used as a 'linear classifier'.

Theorem 3.2 : Let $\mathcal{X} = \{ \underline{x}_1, \underline{x}_2, \ldots, \underline{x}_p \}$ be a set of P patterns belonging to either of the two classes with each $\underline{x}_i$ is n x 1. If the patterns are linearly separable by a hyperplane in n-dimensional space R^n, then an ANN model(with the training method given in Section 3 of Chapter II) classifies the patterns correctly.

Proof: Consider the iterative procedure given in Section (2.3). Assume that a solution weight vector $\underline{w}^*$ exists for a given training set and is normalized such that

$$\| \underline{w}^* \| = 1$$

Using this solution, expression (3.2.2) can be written for δ, a small and arbitrary selected constant ($0 < \delta < 1$) as follows :

$$\underline{w}^{*'} \underline{x} > \delta > 0 \qquad \text{for each } \underline{x} \in \pi_1$$

$$\underline{w}^{*'} \underline{x} < -\delta < 0 \qquad \text{for each } \underline{x} \in \pi_2$$

(3.2.11)

Now let

$$\phi(\underline{w}) = \underline{w}^{*'}\,\underline{w} \,/\, \| \underline{w} \| \qquad (3.2.12)$$

Since $\phi(\underline{w})$ is a scalar product of normalized vectors, we have

$$\phi(\underline{w}) = \cos\theta \leq 1$$

where θ is the angle between the vectors $\underline{w}^{*}$ and $\underline{w}$. Rearranging the numerator and denominator of (3.2.12), we obtain for the k-th training step

$$\underline{w}^{*'}\,\underline{w}^{k+1} = \underline{w}^{*'}\,\underline{w}^{k} + \underline{w}^{*'}\,\underline{x} > \underline{w}^{*'}\,\underline{w}^{k} + \delta \ , \qquad (3.2.13)$$

and

$$\| \underline{w}^{k+1} \|^2 = (\underline{w}^{k} + \underline{x})' \ (\underline{w}^{k} + \underline{x}) \leq \| \underline{w}^{k} \|^2 + 1 \ , \qquad (3.2.14)$$

for the normalized pattern used for training, and for learning constant C = 1.

For total k_o training steps, (3.2.13) and (3.2.14) can be rewritten as

$$\underline{w}^{*'}\,\underline{w}^{ko+1} > k_o\,\delta \qquad (3.2.15)$$

and

$$\| \underline{w}^{ko+1} \|^2 < k_o \qquad (3.2.16)$$

The function (3.2.13) now becomes for $\underline{w} = \underline{w}^{ko+1}$

$$\phi(\underline{w}^{ko+1}) = \frac{\underline{w}^{*'}\,\underline{w}^{ko+1}}{\| \underline{w}^{ko+1} \|} \qquad (3.2.17)$$

and from (3.2.15) and (3.2.16) we observe that

$$\phi(\underline{w}^{k_o+1}) > \sqrt{k_o \delta} \qquad (3.2.18)$$

since $\phi(\underline{w}^{k_o+1}) \leq 1$, inequality (3.2.18) would be violated if a solution were not found for $\sqrt{k_o \delta} < 1$. Hence the proof.

□

Thus, from the above theorem, it can be observed that when the patterns are linearly separable, the ANN model(with training rule given in Section 3 of Chapter II) correctly classifies the patterns. There are many applications of this result which are reported in the literature, however, in this dissertation only statistical applications will be discussed later.

Now the question arises: can such a ANN model be used for a general classification problem wherein the patterns are linearly nonseparable?. The answer is, No (which is clear from the 'XOR' example). However, such problems can be solved by a generalization of ANN structure and such a generalization is referred to as "Multilayer Network". The next Section is concerned with such type of network models.

3.3 MULTILAYER NETWORK

The networks which are constructed with layers of units(neurons) are termed as multilayer network. Multilayer networks are mainly constructed with three types of layers as shown in Fig. 3.4 namely :

1. Input Layer

2. Hidden Layer

3. Output Layer

Below, we give definitions and functions of these three types of layers.

1. Input Layer : The first layer of multilayer network is termed as 'input layer'. It consists of input units, $\underset{\sim}{x}_i$. Input layer sends inputs(or input signals) to the next layer of network through synaptic connections.

2. Hidden Layer : The layer between the input layer and output layer are referred to as 'hidden layers', and units contained in these layers are called as 'hidden units'. Hidden layers are those whose outputs are not directly accessible.

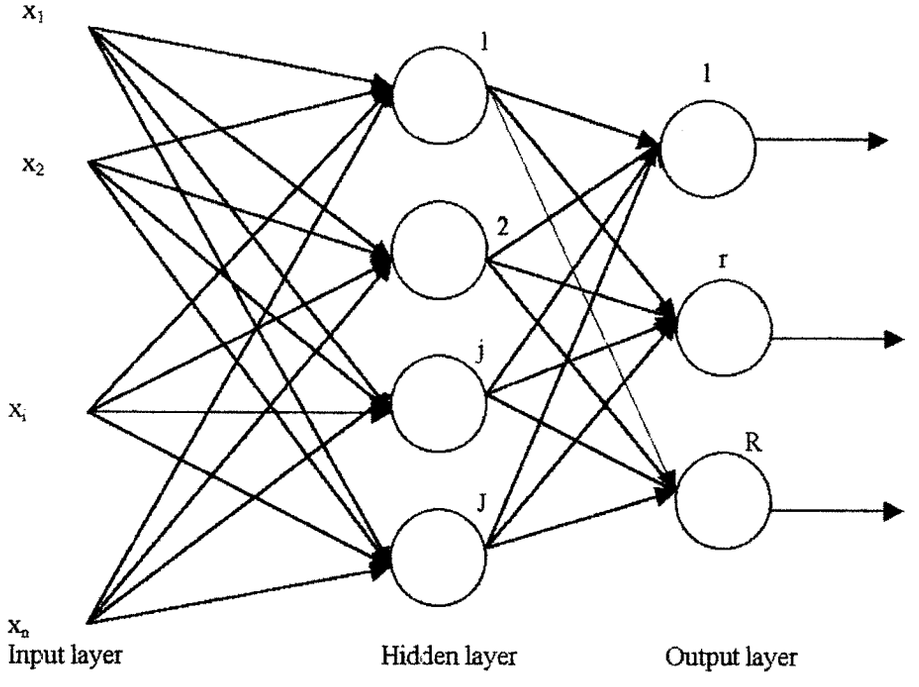

Figure 3.4 Multilayer Neural Network

3. Output Layer: The final layer of multilayer network is 'output layer', which contains 'output units'. It gives the actual output y of the ANN model for a given input.

In the multilayer network, connection between any two units are unidirectional and are represented by arrows as shown in Fig. 3.4.

CONVENTION

The convention followed in the literature is to use the term "layer" in reference to actual number of processing neuron layers. Therefore, we will not count the input layer because it does not perform any computations, but simply passes data onto the next layer. So a network shown in Fig 3.4, is a "two-layer network" which contains one hidden layer. It is also referred to as n-J-R model, where n, J, and R stand for number of units contained in input layer, hidden layer, and output layer respectively.

NOTE :- 1. Note that an N-layer network has N-1 layers whose outputs are not directly accessible (i.e. hidden layers).

2. We also note that, there is only one input layer and only one output layer but there will be more than one hidden layers.

3.3.2 Types Of Multilayer Network

Multilayer networks are of two types depending on the manner in which the neurons of ANN are structured, namely :

1. Multilayer Feedforward Network
2. Multilayer Feedback Network

In the present Chapter, we discuss in detail only multilayer feedforward network (as this network is useful in developing the statistical tools in the present dissertation).

1. Multilayer Feedforward Network :

In this network, the input layer supplies input vectors(patterns) which constitute input signals applied to neurons in the first hidden layer. The output of first hidden layer is used as input for second hidden layer and so on. The set of outputs of the neurons in the final layer(output layer) of the network constitute the over all output of the network for given pattern. This network passes(feeds) signals only in forward direction. Hence the name 'Multilayer Feedfoward Network (MFN)'. Fig. 3.5 shows the MFN with two hidden layers.

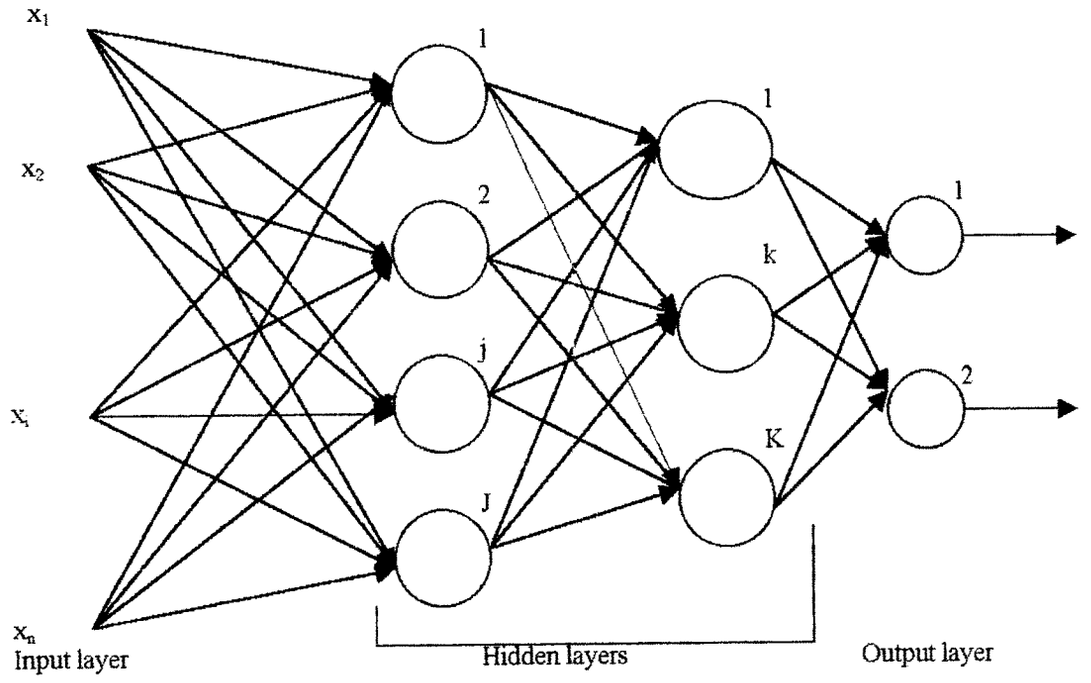

Figure 3.5 MFN With Two Hidden Layers

3.4 TRAINING OF MULTILAYER NETWORK

Unlike ANN model with the training rule given in Chapter II, there was one fundamental problen about multilayer network. Till 1980s, the problem was that there was no convergent learning rule that could be used to train the network. Because of this reason research and development in neural network field slowed down after the concept of multilayer network was first developed in 1960s.

However, Rumelhart, Hinton and, Williams (1986) developed a learning rule and renewed interest in Neural Network theory.

Historically, the rule had been discovered earlier by Werbos(1974) and others, but the research in Neural Network field became intense after the work of Rumelhart and his colleagues. We will now describe this method of training MFN - a method known as the 'Back-propagation training method' which is a dominant learning technique in neural network literature.

As will be seen in the next Section, this method is not 'new' to the statisticians because it is essentially a recursive least-square method.

3.4.1 Back-propagation Training Method

Before we discuss the back-propagation training method, we need the following notations. For this, consider two-layer network as shown in Fig. 3.6

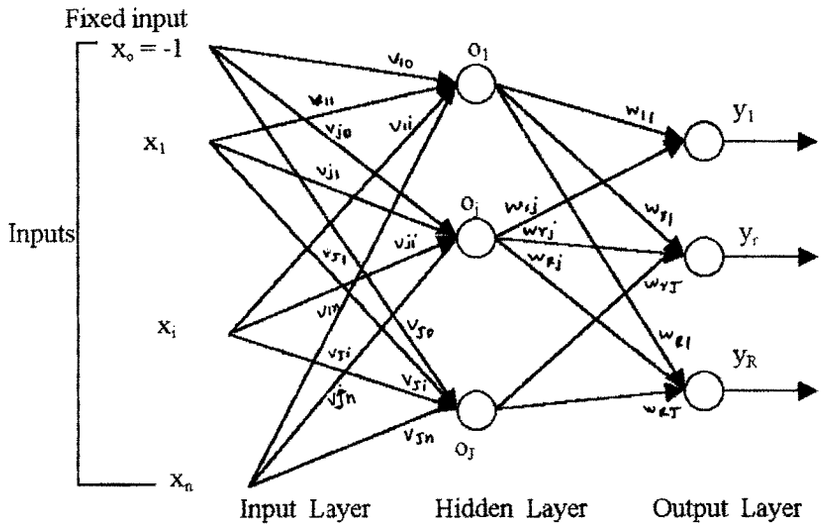

Figure 3.6 Two-Layer Feedforward ANN

Notations:

Let a training set, H be as follows:

$$H = \left\{ (\underline{x}_p , \underline{d}_p), \quad p = 1,2,\ldots P. \right\}$$

where, $\underline{x}_p$ is a p-th training pattern of dimension (n x 1) defined as

$$\underline{x}_p = (x_{p1}, x_{p2}, \ldots, x_{pn})', \qquad p=1,2,\ldots,P.$$

and a vector $\underline{d}_p$ is desired output corresponding to $\underline{x}_p$, of dimension (R x 1) defined as

$$\underline{d}_p = (d_{p1}, d_{p2}, \ldots, d_{pR})', \qquad p=1,2,\ldots,P.$$

The hidden layer response denoted by vector $\underline{o}$ of dimension (J x 1) is given by

$$\underline{o}_p = (o_{p1}, o_{p2}, \ldots, o_{pJ})', \qquad p=1,2,\ldots,P.$$

and the final R-dimensional output of network denoted by vector $\underline{y}_p$ is defined as

$$\underline{y}_p = (y_{p1}, y_{p1}, \ldots, y_{pR})', \qquad p=1,2,\ldots,P.$$

Implementation of Network

The process starts with input values being presented to the input layer. Suppose p-th input pattern is presented. Then the

weighted sum of inputs, denoted by h_{pj} (for p-th pattern), is given by

$$h_{pj} = \sum_{i=1}^{n} v_{ji} x_{pi}, \qquad j=1,2,...J. \tag{3.4.1}$$

Here, n is the total number of input units(observations), v_{ji} is the weight connected from input i to hidden unit j , and x_{pi} is value of the i-th input for pattern p. The output produced by the j-th hidden unit for p-th pattern is

$$o_{pj} = f(h_{pj}), \qquad j=1,2,...,J \tag{3.4.2}$$

here, f(.) is the activation function given in (2.3.4).

Now, the output produced at j-th hidden unit is input signal for the next layer (from Fig. 3.6, next layer is output layer). Therefore, output unit r (r =1,2,...,R) receives a netinput :

$$net_r = g_{pr} = \sum_{j=1}^{J} w_{rj} o_{pj}, \qquad r=1,2,..., R \tag{3.4.3}$$

where, w_{rj} represents the weight connecting from j-th hidden unit to output unit r.

The actual output at output unit (i.e. the final output of network for p-th pattern) is

$$y_{pr} = f(g_{pr}), \qquad r=1,2,...,R \tag{3.4.4}$$

Back-propagation training Rule

Now, using the given training set $(\underline{x}_p, \underline{d}_p)$, p=1,2,...P, the objective is to train the network. For this, the objective function(or error function) to be minimized with respect to weights, is

$$E = \frac{1}{2} \sum_{p=1}^{P} \sum_{r=1}^{R} (d_{pr} - y_{pr})^2, \qquad (3.4.5)$$

NOTE: Eq. (3.4.5) is similar to the Eq. (2.3.8). Also, we observe that (3.4.5) is similar to the Error sum of squares used in Least-Square method.

To minimize (3.4.5), we proceed as follows: First, we differentiate partially (3.4.5) with respect to weight w_{rj}, connecting from j-th hidden unit to r-th output unit. We note that the partial derivative of the error function with respect to a weight represents the rate of change of the error function with respect to weight(that is, back-propagation method uses a gradient search technique to minimize a objective function). Mathematically, this can be represented as

$$\Delta w_{rj} = - \eta \frac{\partial E}{\partial w_{rj}}, \qquad (3.4.6)$$

$$r=1,2,\ldots,R; \; j=1,2,\ldots,J$$

where, η is known as learning rate parameter (or learning

constant) and it simply scales the step size (more about it will be discussed in next Section).

Now, we will derive an expression for (3.4.6) for calculating the adjustment for the weights w_{rj}, connecting the hidden unit j to the output unit r. For this, by substituting Eqs.(3.4.1) to (3.4.4) in Eq. (3.4.5), we get

$$E = \frac{1}{2} \sum_{p=1}^{P} \sum_{r=1}^{R} (d_{pr} - f(\sum_{j=1}^{J} w_{rj} f(\sum_{i=1}^{n} v_{ji} x_{pi})))^2, \tag{3.4.7}$$

Now, expanding Eq. (3.4.6) using chain rule we get

$$-\eta \frac{\partial E}{\partial w_{rj}} = - \eta \frac{\partial E}{\partial y_{pr}} \frac{\partial y_{pr}}{\partial g_{pr}} \frac{\partial g_{pr}}{\partial w_{rj}} , \tag{3.4.8}$$

but

$$\frac{\partial E}{\partial y_{pr}} = -(d_{pr} - y_{pr}), \tag{3.4.9}$$

$$\frac{\partial y_{pr}}{\partial g_{pr}} = f'(g_{pr}), \tag{3.4.10}$$

and

$$\frac{\partial g_{pr}}{\partial w_{rj}} = o_{pj} , \tag{3.4.11}$$

On substituting these results into Eq. (3.4.6), the change in weights w_{rj} is given by

$$\Delta w_{rj} = -\eta\,[-(d_{pr}-y_{pr})]\,f'(g_{pr})o_{pj} \qquad (3.4.12)$$

for $r=1,2,\ldots,R$; $j=1,2,\ldots,J$

This gives a formula to update the weights w_{rj}, from the hidden units to the output units. The weights are updated as

$$w_{rj} = w_{rj} - \eta\,\frac{\partial E}{\partial w_{rj}}\ , \qquad (3.4.13)$$

for $r=1,2,\ldots,R$; $j=1,2,\ldots,J$

or

$$w_{rj} = w_{rj} + \Delta w_{rj}\ , \quad \text{for} \quad r=1,2,\ldots R;\ j=1,2,\ldots,J \qquad (3.4.14)$$

To update the weights, v_{ji}, connecting the i-th input units to the j-th hidden units, we will follow the similar logic as in Eq. (3.4.12)

$$\Delta v_{ji} = -\eta\,\frac{\partial E}{\partial v_{ji}}\ , \qquad (3.4.15)$$

for $i=1,2,\ldots,n$; $j=1,2,\ldots,J$

expanding (3.4.15) using chain rule we get,

$$-\eta\,\frac{\partial E}{\partial v_{ji}} = -\eta \sum_{r=1}^{R} \frac{\partial E}{\partial y_{pr}}\,\frac{\partial y_{pr}}{\partial g_{pr}}\,\frac{\partial g_{pr}}{\partial o_{pr}}\,\frac{\partial o_{pr}}{\partial h_{pj}}\,\frac{\partial h_{pj}}{\partial v_{ji}} \qquad (3.4.16)$$

Here $\frac{\partial E}{\partial y_{pr}}$ and $\frac{\partial y_{pr}}{\partial g_{pr}}$ are given in Eqs. (3.4.9) and (3.4.10) respectively. Also,

$$\frac{\partial g_{pr}}{\partial o_{pj}} = w_{rj} , \tag{3.4.17}$$

$$\frac{\partial o_{pr}}{\partial h_{pj}} = f'(h_{pj}) , \tag{3.4.18}$$

and

$$\frac{\partial h_{pj}}{\partial v_{ji}} = x_{pi} \tag{3.4.19}$$

On substituting Eqs. (3.4.17) to (3.4.19) into Eq. (3.4.16) we get

$$\Delta v_{ji} = -\eta \sum_{r=1}^{R} (d_{pr} - y_{pr})\, f'(g_{pr}) w_{rj}\, f'(h_{pj}) x_{pi} , \tag{3.4.20}$$

Here summation is taken over the number of output units, because each hidden unit is connected to all the output units. So, if the weight connecting an input to a hidden layer changes, it will affect all the outputs.

Now, the weights from hidden units to output units are updated as

$$v_{ji} = v_{ji} + \Delta v_{ji} , \tag{3.4.21}$$

The computation of weight adjustment factors as in (3.4.12) and (3.4.20) requires the specification for the activation function used is given by (2.3.4) namely

$$f(net) = \frac{2}{1 + \exp(-\lambda net)} - 1 ,$$

with activation constant λ=1 and value of f'(.) is expressed in (2.3.19) as

$$f'(net) = \frac{1}{2} (1 - f(net)*f(net))$$

This completes the description of the Back-propagation method.

The algorithm of this method is enclosed in Appendix C(1). Due to the lack of availability of software for this method, we have developed a software 'Back-Prop' in 'C' language. The source code is enclosed in Appendix C(2).

Illustration: Computation of 'XOR' Function

The following example demonstrates the main features of the back-propagation training algorithm applied to a two-layer network. As already pointed out, 'XOR' problem cannot be solved using single layer ANN. Therefore, consider a two-layer (modified) network for computation of 'XOR' function, as shown in Fig 3.7(a)

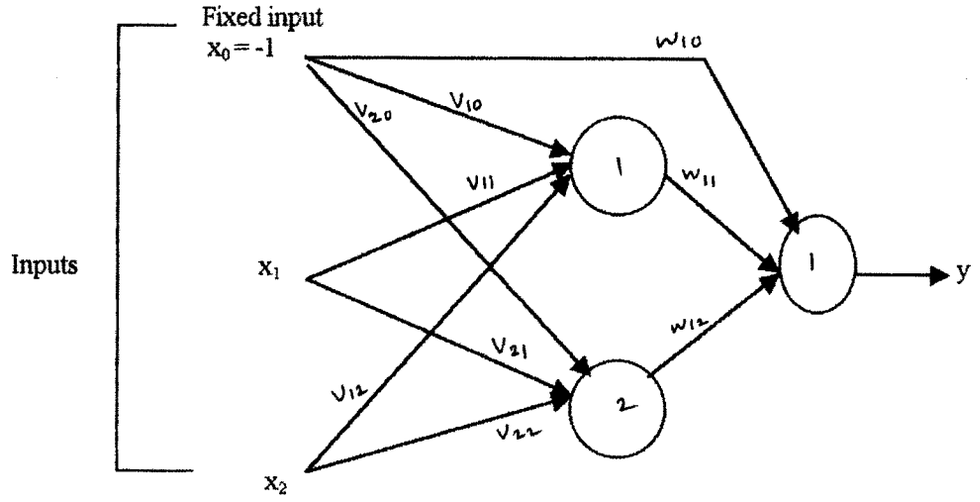

Figure 3.7 (a) ANN Model For Computing 'XOR' Function

Training set required for implementation of 'XOR' function is given in Table 3.4

TABLE 3.4

$\underline{x}$	(0,0)	(0,1)	(1,0)	(1,1)
x_1. XOR. x_2	0	1	1	0

For the sake of simplicity, the weights from output layer to hidden layer and hidden layer to input layer are expressed in matrix w and v respectively(from Fig. 3.7(a)) as follows:

$$w = \begin{bmatrix} w_{10} & w_{11} & w_{12} \end{bmatrix}$$

$$V = \begin{bmatrix} v_{10} & v_{11} & v_{12} \\ v_{20} & v_{21} & v_{22} \end{bmatrix}$$

After executing the program 'Back-prop' enclosed in Appendix C(2), the resulting weights obtained for run 1 (1106 iterations) are summarized below for η = 0.8

$$\hat{w} = \begin{bmatrix} -4.67 & 1.644 & 4.244 \end{bmatrix}$$

$$\hat{v} = \begin{bmatrix} 1.531 & 4.69 & 4.786 \\ 3.447 & 2.35 & 2.358 \end{bmatrix}$$

The results obtained for run 2 (1350 iterations) and η = 1, (with another set of initial weights) are :

$$\hat{w} = \begin{bmatrix} 4.564 & 1.57 & 4.474 \end{bmatrix}$$

$$\hat{v} = \begin{bmatrix} 1.462 & 4.531 & 4.431 \\ -3.753 & -2.627 & -2.62 \end{bmatrix}$$

Based on the above results, we can analyze the implemented mapping of input to output space. From the result of run 1, we get,

$$4.686x_1 + 4.786x_2 - 1.5306 = 0$$

$$2.345x_1 + 2.358x_2 - 3.4474 = 0$$

The graphical representation of above lines are shown in Fig. 3.7(b). It can be seen that the patterns (0,0), (1,1) and (1,0), (0,1) are positioned at different sides of the above lines.

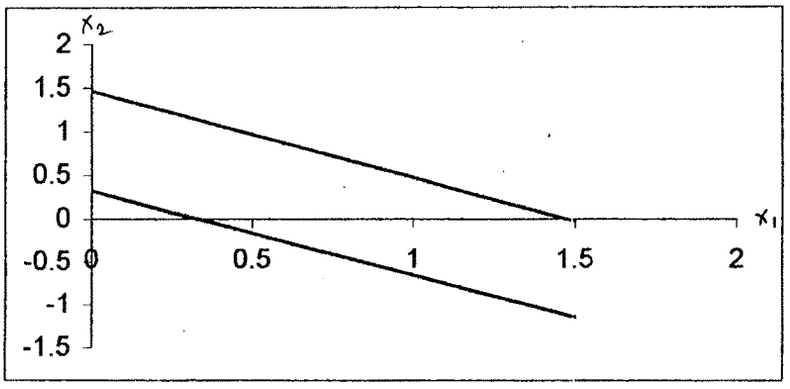

Figure 3.7(b)

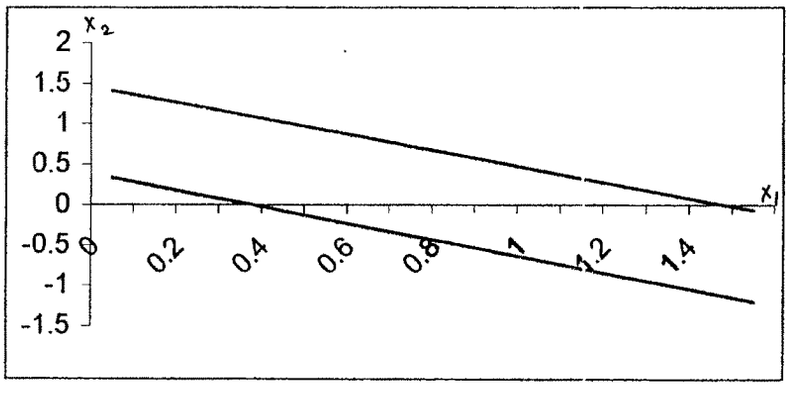

Figure 3.7(c)

Using the result of run 2, we have Fig. 3.7(c). And, we observed that the results of run 2 are very similar to those obtained in run 1. As done in case of 'AND' computation, here also we can test the trained network with the input vector $\underline{x}$. Then, for $\underline{x} = (1,0)'$, we get the neural output as 0.98 which is greater than 0.5, and hence the output is taken as 1 which

matches the 'correct' output. For $\underline{x} = (0,0)'$, we get $0.05<0.5$ which means that output is zero.

REMARK : The above example indicates that how linearly nonseparable patterns can be correctly classified by ANN model using back-propagation training method.

IMPORTANT NOTE : Cybenko (1989) has shown that any continuous function defined on compact set in R^n can be uniformly approximated by a multilayer ANN with one hidden (unit) layer.

3.5 ADDITIONAL COMMENTS ON TRAINING PARAMETERS

It has been empirically observed that the values of various training parameters namely, initial weights, the form of neurons activation function, selection of the learning constant, selection of necessary number of hidden neurons, and training error term affect the time required for training of the network. The role of these parameters in training the ANN is discussed below. The discussion is based on Zurada (1992), Wasserman (1992), Haykin (1994), and Mehta and Mhatre (1996).

1. Initial Weights:

The error back-propagation algorithm requires that small weights be chosen initially at random. In various studies, it

has been found out that these initial weights strongly affect the final training of the network. If equal weights are chosen at the beginning of training, the network may not train correctly and results in constant and equal output. Hence, the weights should be chosen at random initially (usually, the weights are selected from Uniform distribution over (-1,1) or (0,1)).

2. Learning constant(η):

The convergence and effectiveness of the back-propagation learning algorithm depend significantly on the value of learning constant η. The size of the step taken in gradient direction is determined by the learning constant. It is observed that, the convergence will be slow, if η is too small. Whereas, if it is too large, the network might never converge at all. However, a small value of η always guarantee the convergence. When the function has a broad minima, a large value of η(step size) can help to converge, but for narrow and steep minima, a small η must be chosen to avoid divergent solution (Zurada, 1992).

Hence, there is no single learning constant suitable for all training cases. Initially, a small value of $\eta \approx 0.8$ should be chosen to guarantee convergence. As an illustration, consider the following:

Table 3.5 shows, how various values of learning constants(η)

affect (in 'XOR' problem) an error factor for different number of iterations.

TABLE 3.5

η \ k	1000	1500	2000	3000
0.1	0.131	0.027	0.017	0.012
0.8	0.005	0.004	0.0035	0.0028
1.2	0.008	0.006	0.0043	0.0063

Foot Note : η = Learning Constant
k = Number of Iterations

3. Steepness of Activation Function:

As mentioned in Chapter I, the continuous activation function f(.),(usually the sigmoid function (2.3.4), is characterized by activation constant(steepness factor) λ. Also, the derivative term of the activation function f'(.) is one of the factors in error signal component (or weight adjustment term). Thus, both choice and shape of the activation function would strongly affect the speed of network training. The activation constant(λ) determines the slope of the activation function. Fig. 3.8 shows that for a fixed learning constant(η), all adjustment of weights are in proportion to the steepness factor λ.

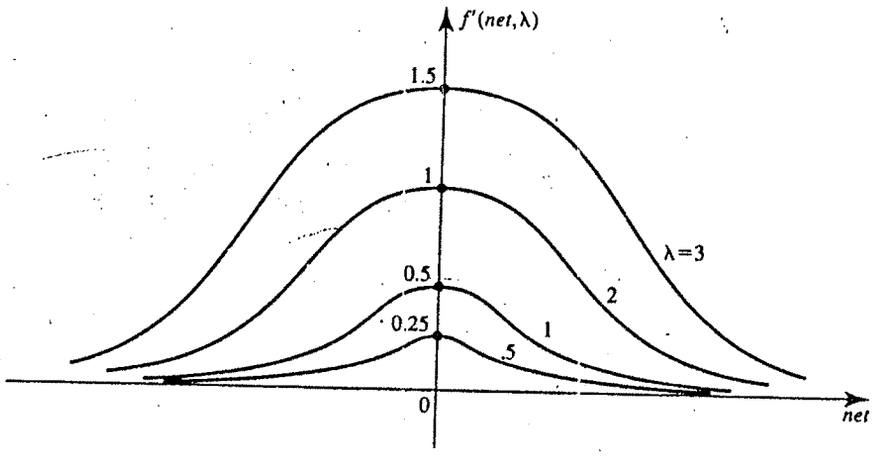

Figure 3.8 Slope of Logistic Function For Various λ Values.

This implies that, using activation function with large λ may yield results similar as in the case of large η. So, Zurada (1992) suggested that λ be fixed constant at 1 and η be adjusted rather than varying both λ and η.

Table 3.6 shows number of iterations required to converge the procedure at fixed λ(=1) and various values of learning constant(η).

TABLE 3.6

Error = 0.005	Activation Constant(λ) = 1
η	Number of Iterations
0.1	4732
0.8	1150
1.2	1287

The above table indicates that η should be taken initially 0.8 at λ = 1

4. *Number Of Hidden Neurons:*

The choice of hidden layer neurons is important to the networks discrimination ability. This number depends on the dimension of the input vector and the number of separable regions in the input space. Mirchandini and Cao (1989) gives a formula for calculating the necessary number of hidden layer neurons and it is as follows :

If the n-dimensional Euclidean input space is linearly separable into M disjoint regions, each belonging to one of the R classes such that $R \leq M$, then there exists a relationship between M, n and J, where J is number of hidden layer neurons. One of the three parameters can be calculated given the other two. The maximum number of linearly separable regions can be given by

$$M(J,n) = \sum_{k=0}^{n} {}^{J}c_{k} \quad , \quad \text{where} \quad {}^{J}c_{k} = 0 \quad \text{for} \quad J < k \tag{3.5.1}$$

For large-size input vectors compared to the number of hidden units, or when $n \geq J$, we have from (3.5.1)

$$M = \binom{J}{0} + \binom{J}{1} + \ldots\ldots + \binom{J}{J} = 2^{J} \tag{3.5.2}$$

or

$$J = \log_2 M \qquad (3.5.3)$$

Using minimum number of hidden neurons, network has ability to learn the patterns properly, while two many increase the training period. (Too many neurons may also lead to a problem known as "overfitting", which occurs when the network has no much information processing capacity). Thus, it is important to use a minimum number of neurons in the hidden layer.

As an illustration we present the following example :

Example 3.3: In this example, we will use the above discussed guidelines to chose suitable network for the two-dimensional XOR-problem. In this problem, we have M=4 and n=2, so, using (3.5.3) we get

$$J = 2.$$

This implies that, for XOR-problem minimum hidden nodes are two.

From Table 3.7 and 3.8 we observe that, for XOR problem though the procedure converges with two hidden units, it converges faster with more than two hidden units.

TABLE 3.7

Number of Hidden Neurons = 3				
η\k	1000	1500	2000	3000
0.1	0.131	0.027	0.017	0.012
0.8	0.005	0.004	0.0035	0.0028
1.2	0.008	0.006	0.0043	0.0063

TABLE 3.8

Number of Hidden Neurons = 2				
η\k	1000	1500	2000	3000
0.1	0.071	0.033	0.0197	0.0129
0.8	0.005	0.004	0.0035	0.0027
1.0	0.007	0.004	0.0039	0.0032

Foot Note: η = Learning Constant
k = number of iterations.

5. Training Error:

Note that, the 'cumulative error' is computed over the back-propagation training cycle, and it is expressed as

$$E = \frac{1}{2} \sum_{p=1}^{P} \sum_{r=1}^{R} (d_{pr} - y_{pr})^2 \quad , \qquad (3.5.4)$$

Eq.(3.5.4) gives the accuracy of the neural network mapping after each of training cycle. Such a definition of error, however, it not very effective for comparing the performance of

networks with different numbers of training patterns P and having a number of output neurons. Network with output units R, when trained using large number of patterns in the training set, will usually produce large error (E) due to the large number of terms in the sum. For similar reason, network with large P trained using the same training set would usually produce large cumulative error. Thus, a more adequate measure of error can be used as follows :

$$E_{rms} = \frac{1}{PR} \sqrt{\sum_{k=1}^{P} \sum_{r=1}^{R} (d_{kr} - y_{kr})^2} \quad , \tag{3.5.5}$$

The above error term is called as 'root mean-square normalized error'. This seems to be more effective than E as given in (3.5.4) when comparing the outcome of training the neural networks.

In our software, therefore we have used (3.5.5) as the error function.

CONCLUSION :

Upto this part, we have discussed the basic theory of ANN and illustrated its use in computation of simple functions. In the following Chapters, we will be concerned with the applications of ANN, as an alternative tool to many widely used statistical data analysis techniques. ■