CHAPTER II

SINGLE LAYER ARTIFICIAL NEURAL NETWORK

2.1 INTRODUCTION

In this Chapter, we will discuss the basic structure of ANN called "M-P Neural Network", which was proposed by McCulloch and Pitts (1943). It was the first step in development of ANN modelling. It mimics the characteristics of human brain as discussed in Chapter I.

Starting with definition of M-P Neuron model, Section 2 discusses the single layer ANN with threshold function. Also, in this Section, the M-P model is implemented. For this, we have developed a software in 'C' language and it is given in the Appendix. Section 3 gives the single layer ANN model with continuous activation function. The Section ends with some suitable illustrations.

2.2 McCULLOCH-PITTS ARTIFICIAL NEURON MODEL

The artificial neuron model draws the inspiration from the biological neural network and it was first proposed by McCulloch and Pitts in 1943. In honour of them, neuron model of this type is referred to as "M-P Neuron Model". The M-P Neuron model is simple and was mainly designed to perform the basic logical operations namely AND, OR, NOT functions.

2.2.1 *McCulloch-Pitts Artificial Neuron(AN) Model*

The M-P neuron model or network has four major components namely :

1) Input values denoted as a vector $\underline{x} = (x_1, x_2, \ldots, x_n)'$
2) Vector of weights denoted by $\underline{w} = (w_1, w_2, \ldots, w_n)'$
3) Output value denoted by y
4) Threshold value denoted by T

In this model, x_i's assume either 0 or 1 value, weights w_i's are unknown and y is the single output value which is also 0 or 1 and T is threshold value. The Fig. 2.1 shows the structure of M-P Neuron model.

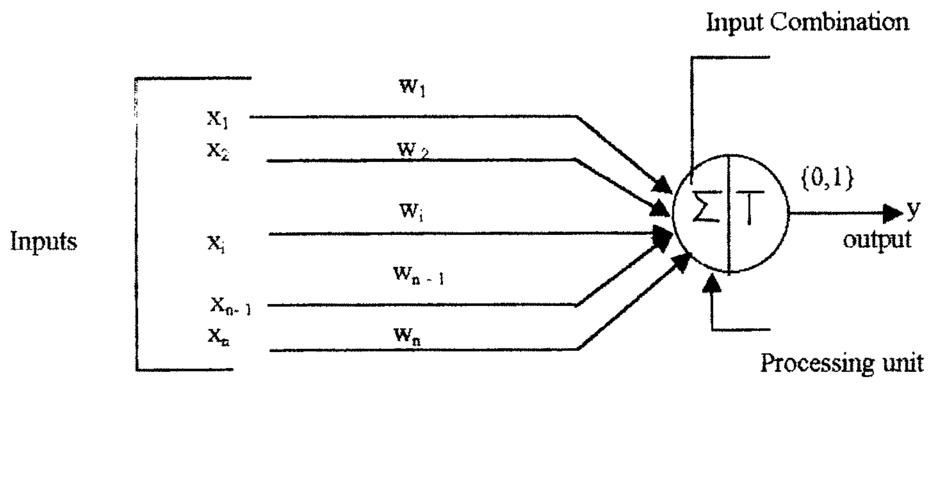

Figure 2.1 McCulloch-Pitts AN Model

Here, the neuron receives a set of inputs $(x_1, x_2, \dots . x_n)$ and produces corresponding output y. The output of the model is computed by a rule :

$$y = \begin{cases} 1 , & \text{if} \quad \sum_{i=1}^{n} w_i x_i > T \\ 0 , & \text{if} \quad \sum_{i=1}^{n} w_i x_i \leq T \end{cases} \qquad (2.2.1)$$

Equivalently, if we set $w_0 = T$ and $x_0 = -1$, (2.2.1) can be expressed as

$$y = \begin{cases} 1 , & \text{if} \quad \sum_{i=0}^{n} w_i x_i > 0 \\ 0 , & \text{if} \quad \sum_{i=0}^{n} w_i x_i \leq 0 \end{cases} \qquad (2.2.2)$$

for evaluating the logical operations like 'AND', 'OR', and 'NOT' by suitably choosing the weights w_i's. In network terminology, w_i's are to be obtained by 'training ' the network with a given set of input-output pair $(\underline{x}, d)$, where d is the actual output corresponding to the known input $\underline{x}$. Once such a set of w_i's are obtained, then corresponding to any future input $\underline{x}$, the ANN model using the rule (2.2.2) produces the output which is 'closer' to the correct output d. This situation is similar to what is done in regression analysis. For instance, in a simple regression model

$$y = \alpha + \beta x + \varepsilon ,$$

using the data (x_i, y_i), α and β are estimated and for a given future value x, the predicted value y is computed. Further, we observe that, essentially what ANN model accomplishes is that it maps the input $\underline{x}$ into the output y. Since $\underline{x}$ is an n-dimensional vector, and y is a scalar taking values either 0 or 1, we get

$$y \; : \; R^n \longrightarrow \{0,1\}$$

In ANN modelling, in general, the output y 'squashes' into discrete values e.g. {0,1} or {-1,1}. The generalization of this part will be discussed in later Chapter.

REMARK 2.1 : The above type of ANN model is referred to as 'single layer ANN model' for the reason that this model has only one processing layer (more about the term "layer" will be discussed in next Chapter).

For successful implementation of ANN model, we need to determine suitable w_i's using a set of training data $(\underline{x}_k, d_k)$, k=1,2,...P. To do this, we first note that the Eq. (2.2.2) is a set of P linear inequalities. When it is viewed geometrically, the problem is to seek a hyperplane which separates two regions. Then it is a linear programming problem. Many procedures for solving such a system of inequalities are available in the literature.

Below we discuss a simple method (Rosenblatt, 1958) for training the M-P neural network, that is for obtaining w_i's which will satisfy Eq. (2.2.2).

2.2.2 Training M-P Neural Network

Let a set of input and corresponding desired output (training set) be as follows :

$$H = \left\{(\underline{x}_1, d_1), (\underline{x}_2, d_2), \ldots\ldots (\underline{x}_P, d_P)\right\} \qquad (2.2.3)$$

where $\underline{x}_k$ is (n X 1) and d_k is a scalar , k=1,2,.....P.

Consider the ANN model as shown in Fig. 2.1 and the rule given by (2.2.2). Define net_k for k-th input vector $\underline{x}_k$ as

$$net_k = \underline{w}' \underline{x}_k, \qquad k = 1,2,\ldots.P \qquad (2.2.4)$$

where

$$\underline{w} = (w_1, w_2, \ldots.w_n)'$$

The objective is to obtain a set of weight w_i's, (i = 1,2,...n) such that neuron output y, given by the rule (2.2.2) is 'close to' the expected or desired output, say d, for all patterns, where the 'closeness' is measured by suitable function like

$$E = \frac{1}{2} (y_k - d_k)^2 .$$

For this, the procedure to obtain such w_i's consists of the following steps :

1. Select an initial set of weights, say $\underline{w}_o$ (normally, the elements in $\underline{w}_o$ are selected at random that is, numbers from U(0,1) or U(-1,1) are taken as initial values). Also set the error term $E = 0$.
2. Further, choose any input vector $\underline{x}_k$, $k=1,2,\ldots P$ and compute

$$net_k = \underline{w}_o{}'\underline{x}_k , \qquad (2.2.5)$$

3. Compute the neuron output y_k, using rule (2.2.2).
4. After applying each input $\underline{x}_k$ to the ANN, weights are updated at r-th stage as follows :

$$\underline{w}^{r+1} = \underline{w}^{r} + c(d_k - y_k)\underline{x}_k \ , \quad k = 1,2,\ldots,P \qquad (2.2.6)$$

where a constant $c > 0$ is called the 'correction term'.
5. Compute error term E as

$$E = \frac{1}{2}(y_k - d_k)^2 + E, \qquad (2.2.7)$$

6. Repeat steps 2 to 5 till all inputs are introduced.
7. If $E < \delta$, where δ is prespecified small number (usually 0.0001) then stop the procedure. Otherwise, initialize error term E to zero and enter the new training cycle by going to step 1.

REMARK 2.2 : In the above method, we observe that first an arbitrary hyperplane is drawn (by selecting $\underline{w}_o$ and computing $\underline{w}_o'\underline{x}_k$) and then afterwards by updating $\underline{w}$'s using the expression (2.2.6), the hyperplane is adjusted so that the output y becomes 'closer to' the expected output.

REMARK 2.3 : At this stage, the question arises whether the above iterative procedure converges. The answer is Yes, and since some more notations and concepts are still needed to prove the convergence, we postpone the proof to the next Chapter.

Since the necessary software for implementing the above method is not available, we have developed the same in 'C' language and it is enclosed in Appendix A.

Using the M-P model presented in Fig. 2.1, we implement the above iterative procedure for computing 'AND', 'OR' logical operators.

Before we present the examples, for the sake of completeness we describe the above logical operators, which play very important role in computerized data processing.

The 'AND' and 'OR' logical operators are used with two expressions. These operators act upon two operands (that are themselves logical expressions) which are either true or false. Generally, when result is true, an integer '1' is used otherwise

'0'. The result of a logical 'AND' operation will be true only if both operands are true. On the other hand, the result of a logical 'OR' operation will be false only if both operands are false. Further, the logical operator 'NOT' is a negation operator.

Example 2.1 : Computation of 'AND' Function.

Let x_1 and x_2 denote two logical expressions which means that their values are either 1 or 0.

Consider a set of input vectors (x_1, x_2) as shown below. Then the value of $(x_1 .AND. x_2)$ are as follows :

TABLE 2.1

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

Consider the following ANN model that accepts the input (x_1, x_2) and produces the output y

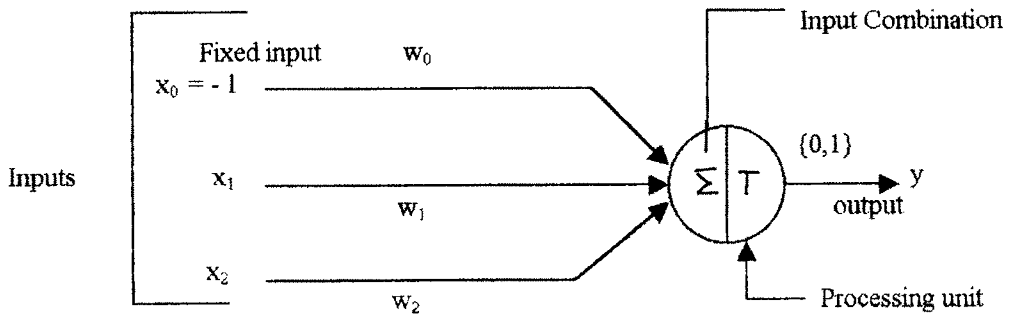

Figure 2.2 ANN Model For Implementation Of 'AND' Operator

Here, the training set consists of

$$\left\{((0,0),\ 0),\ \ ((0,1),\ 0),\ \ ((1,0),\ 0),\ ((1,1),\ 1)\right\}.$$

To train the ANN, we use the program enclosed in Appendix A. The final estimated weights after 2 and 4 iterations are given in Tables 2.2 and 2.3 for different values of correction term(c) :

TABLE 2.2

Weights	w_0	w_1	w_2
Estimated ($\hat{\underline{w}}$)	0.2234	1.0239	1.2697
Number of Iterations: 2			
Error = 0		c = 0.8	

TABLE 2.3

Weights	w_0	w_1	w_2
Estimated($\hat{\underline{w}}$)	0.608	1.4758	1.742
Number of Iterations: 4			
Error = 0		c = 0.5	

CONCLUSION : In a repeated runs, it is observed that, for 'AND' function, iterative process converges in maximum four iterations.

Now, using the estimated weights from Table 2.3, graphical representation of 'AND' function is shown in Fig. 2.3. It clearly shows that the patterns (0,0), (0,1), (1,0) falls below a line defined by $\hat{\underline{w}}'\underline{x}$ and the point (1,1) above the line. This implies that patterns are separated by a line whose equation is $\hat{\underline{w}}'\underline{x}$ where $\hat{\underline{w}} = (\hat{w}_0, \hat{w}_1, \hat{w}_2)'$ with their corresponding classes.

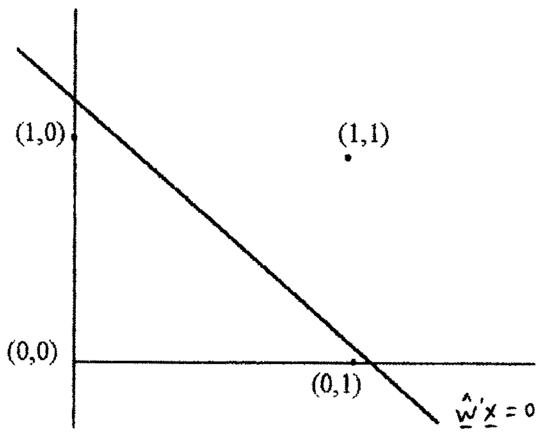

Figure 2.3 Graphical Representation Of 'AND' Function

Once the network is trained in this manner, then in future whenever an input $\underline{x}$ is given to the network, the output produced will be approximately equal to the 'correct' output. For example, suppose we have $\underline{x} = (0,1)'$ then with the above $\hat{\underline{w}}$, we get

$$\hat{\underline{w}}'\underline{x} = 0.134$$

Then following conventions are used to interpret the output :

If the output < 0.5 then it is taken as 0

If the output ≥ 0.5 then it is taken as 1.

In the light of above, for the vectors $\underline{x}_1 = (1,0)'$ and $\underline{x}_2 = (0,0)'$ we get outputs equal to 0 and 0 respectively. This shows that the trained model correctly evaluates the 'AND' operator.

The type of computations done above deserves some more explanation :

It is appropriate at this stage to point out the difference between the neural computations and traditional programmed computations of say, 'AND' operator. In the programmed computation of 'AND' operator, one proceeds as follows :

```
1.  Read the values of $(x_1,x_2)$
2.  If $(x_1=1)$
        If $(x_2=1)$
            output = 1
        else If $(x_2=0)$
            output = 0
3.  Repeat the steps 1 and 2 for all pairs $(x_1,x_2)$.
```

Thus, in the above computations, specific rules are specified and according to these rules, the outputs are computed. Obviously, once the input data is given, as per the instructions, the corresponding output will be computed. On the other hand, in neural computation, a 'learning process' is involved (just like a child learns slowly to recognize a alphabet when he is presented the correct form of a letter). In case of 'AND' computation the 'relationship' between the input pattern and the target pattern is 'learned' and the relationship is stored in the form of weights w_i's.

Example 2.2 : Computation of 'OR' Function.

Consider ANN model shown in Fig. 2.2 and the training set for 'OR' function presented in Table 2.4.

TABLE 2.4

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

After executing the program enclosed in Appendix A, the final weights for different values of c are given in Table 2.5 and 2.6

TABLE 2.5

Weights	w_0	w_1	w_2
Estimated ($\hat{\underline{w}}$)	-1.1225	1.0105	-0.488
Number of Iterations: 2			
Error = 0		c = 0.6	

TABLE 2.6

Weights	w_0	w_1	w_2
Estimated ($\hat{\underline{w}}$)	-0.2247	1.7689	1.296
Number of Iterations: 2			
Error: 0		c: 0.8	

CONCLUSION : From Table 2.5 and 2.6, it is observed that, in a repeated runs for c = 0.8 the procedure converges in two iterations.

Using final weights in Table 2.6, we have Fig. 2.4 and we see that the trained ANN model computes the output correctly for given x

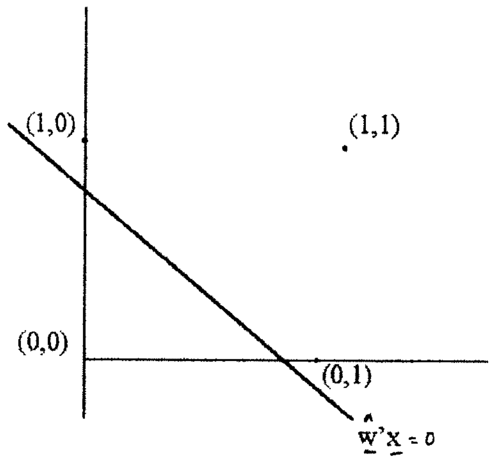

Figure 2.4 Graphical Representation Of 'OR' Function

EXAMPLE 2.3 : Comutation of 'NOT' Function

Training set required for implementation of 'NOT' operator is as follows :

x	0	1
NOT x	1	0

Like 'AND' and 'OR', 'NOT' operator is also implemented using the ANN model and we have obtained correct results.

2.3 ANN WITH NONLINEAR ACTIVATION FUNCTION

In the previous section, the M-P neural network model was developed for computing 'AND', 'OR' and 'NOT' type of functions. A specific rule namely (2.2.2) was used to compute the output of neuron model. At this stage, a natural question arises as to whether there exists any other rule which will perform similar computations. To answer this question, first we rewrite (2.2.2) as follows :

$$y = \begin{cases} 1, & \text{if } f(net) > 0 \\ 0, & \text{if } f(net) \leq 0 \end{cases} \tag{2.3.1}$$

where

$$net = \underline{w}' \underline{x},$$

and

$$f(net) = net \tag{2.3.2}$$

As discussed in Chapter I, there ~~exists~~ exist many functions f(.) called 'activation functions', which maps the input $\underline{x}$ to output y. We note that (2.3.2) represents the identity function. One of the widely used activation functions is sigmoid function (which is nonlinear in nature) among other functions.

REMARK 2.4 : In the literature on neural network, the function f(.) is called 'activation function', since it has some physical meaning. Essentially, as noted earlier f(net) squashes or limits the output value. This is something similar to what happens in responses (or outputs) given by the brain.

Now, we discuss ANN model with sigmoid type nonlinear activation function :

Consider an ANN model with nonlinear activation function as shown in Fig. 2.5

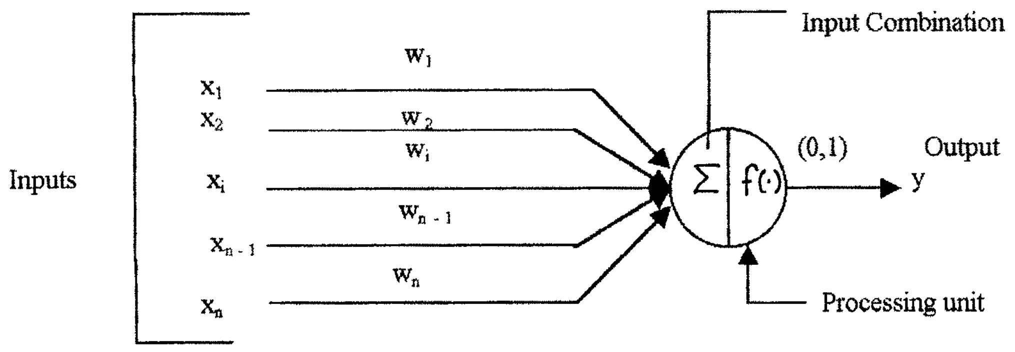

Figure 2.5 ANN Model With Nonlinear Activation Function

The above Fig. shows that the input $\underline{x}$ is mapped into y that is

$$y = f(\underline{w}^t \underline{x}), \qquad (2.3.3)$$

The choice of f(.), for further discussion, will be the sigmoid function, given by

$$f(net) = \frac{2}{1 + \exp(-\lambda net)} - 1 , \qquad (2.3.4)$$

where

13643

$$net = \underline{w}'\underline{x} \ ,$$

and activation constant $\lambda=1$.

Let d be the desired output corresponding to the input $\underline{x}$. Then, the ANN model is so designed that the neuron output y corresponding to $\underline{x}$ is approximately equal to d the desired output, that is

$$y = f(net) \cong d$$

Thus, in order to implement the ANN model, we need to find $\underline{w}$, and below, we discuss a method of training the above ANN.

2.3.1 *Training ANN with Nonlinear Activation function:*

Consider a training data in the form of P pairs $(\underline{x}_k, d_k)$:

$$\{ (\underline{x}_k, d_k),\ k = 1,2,\ldots.P \} \qquad (2.3.5)$$

For a given pattern k, the network maps $\underline{x}_k$ into neuron's output y_k using nonlinear operation as follows

$$y_k = f(\underline{w}'\underline{x}_k), \qquad (2.3.6)$$

or

$$y_k = f(net_k), \qquad (2.3.7)$$

As discussed earlier, the goal of the training the ANN is to produce y_k such that it replicates corresponding expected output d_k, that is

$$y_k = f(net_k) \cong d_k$$

The quality of approximation is determined by the error term

$$E = \frac{1}{2}\sum_{k=1}^{P}(d_k - f(\underline{w}'\underline{x}_k))^2 \qquad (2.3.8)$$

Note that the expression (2.3.8) is a function of only w_i's and hence the problem is to find w_i's such that E is minimum (here the factor 1/2 is taken for simplifying the algebra).

REMARK 2.5 : Finding w_i's by minimizing E is similar to estimation of parameters in regression model using Least-Squares method.

Finding Weights w_i's

To find minimum of (2.3.8) with respect to $\underline{w}$, we proceed as follows :

On differentiating equation (2.3.8) with respect to weight $\underline{w}$, we get,

$$\frac{\partial E}{\partial \underline{w}} = \frac{\partial}{\partial \underline{w}}\left[\frac{1}{2}(d - f(\underline{w}'\underline{x}))^2\right] \qquad (2.3.9)$$

Here, for the sake of convenience we have suppressed the subscript k.

Equivalently, we get

$$\frac{\partial E}{\partial \underline{w}} = \frac{\partial}{\partial \underline{w}}\left[\frac{1}{2}(d - f(net))^2\right], \qquad (2.3.10)$$

Now, expanding (2.3.10) using chain rule, we get

$$\frac{\partial E}{\partial \underline{w}} = \frac{\partial E}{\partial f(net)} \frac{\partial f(net)}{\partial\ net} \frac{\partial net}{\partial\ \underline{w}}\ , \qquad (2.3.11)$$

but,

$$\frac{\partial E}{\partial f(net)} = -(d- f(net))\ , \qquad (2.3.12)$$

$$\frac{\partial f(net)}{\partial net} = -\ f'(net) \qquad (2.3.13)$$

Since, net = $\underline{w}'\underline{x}$, we have

$$\frac{\partial(net)}{\partial \underline{w}} = \underline{x}\ , \qquad (2.3.14)$$

and (2.3.11) can be rewritten as

$$\frac{\partial E}{\partial \underline{w}} = -(d- f(net))\ f'(net)\ \underline{x}, \qquad (2.3.15)$$

Or for i-th component in $\underline{w}$, we get

$$\frac{\partial E}{\partial w_i} = -(d- f(net))\ f'(net)\ x_i\ ,$$

(2.3.16)

We observe that, a solution for equation (2.3.16) is not in the closed form. Therefore, an iterative procedure is used to find w_i's.

Initially, choosing arbitrary weights w_i's, the weight adjustment for iterative procedure is given by

$$\underline{w} = \underline{w} - \eta \, dE/\partial \underline{w}, \qquad (2.3.17)$$

where η is positive constant called 'learning constant' or 'learning rate parameter' (more on this will be discussed later).

The computation of weight adjustment as in (2.3.16) requires the specification for the activation function used.

Let us express f'(net) in terms of nonlinear activation function (2.3.4) we obtain

$$f'(net) = \frac{2 \exp(-net)}{[1 + \exp(-net)]^2}, \qquad (2.3.18)$$

After rearranging the terms in (2.3.18) and letting y =f(net), we get

$$\frac{2 \exp(-net)}{[1 + \exp(-net)]^2} = \frac{1}{2} (1-y^2), \qquad (2.3.19)$$

Then (2.3.15) become

$$\partial E/\partial \underline{w} = - \frac{1}{2} (d - y) (1-y^2) \underline{x} \qquad (2.3.20)$$

Therefore, at r-th training stage, we have

$$\underline{w}^{r+1} = \underline{w}^{r} + \frac{1}{2} \eta (d - y) (1-y^2) \underline{x} \qquad (2.3.21)$$

And the process is terminated as soon as

$$E < \delta$$

where δ is a small allowable error for a given problem.

Since the Software to implement the above procedure is not available here, we have developed a Program 'SANN' in 'C' language for implementing the same . The source code is given in Appendix B.

Here, we illustrate the above model with given training procedure for computation of logical operators using sigmoid function.

Example 2.4 : Computation of 'OR' Function

Consider the ANN model with nonlinear activation function as shown in Fig. 2.6. The training set required for computation of 'OR' function as given in Table 2.4.

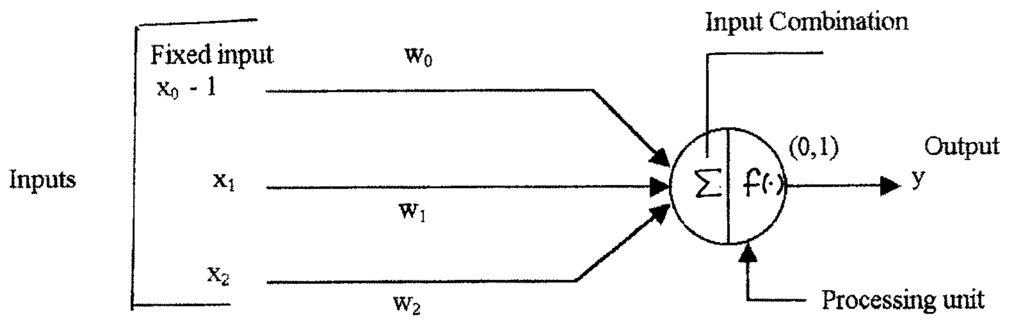

Figure 2.6 ANN Model With Nonlinear Activation Function For Implementation Of 'OR' Function

Table 2.7 and 2.8 gives the estimated weights after execution of the Program enclosed in Appendix B.

TABLE 2.7

Weights	w_0	w_1	w_2
Estimated ($\hat{\underline{w}}$)	-0.0054	4.3161	4.3191
Number of Iterations: 1000			
Error = 0.006		η = 1	

TABLE 2.8

Weights	w_0	w_1	w_2
Estimated ($\hat{\underline{w}}$)	-0.03814	3.293	3.3048
Number of Iterations: 180			
Error = 0.017		η = 0.8	

CONCLUSION: From the above tables, we conclude that error decreases as number of iterations increases. In a repeated runs we observed that for learning constant(η) mostly in between 0.8 to 1.2 error become very small in maximum iterations 1000. Also, note that the number of iterations required for this procedure is more as compared to earlier results given in Table 2.5 and 2.6.

Example 2.5 : Computation of 'AND' Function.

For computing 'AND' function training set is given in Table 2.1 and ANN model shown in Fig.2.6.

After executing program enclosed in Appendix B we get an estimated weights as shown in Table 2.9 and 2.10 for different values of learning constant(η).

TABLE 2.9

Weights	w_0	w_1	w_2
Estimated ($\hat{\underline{w}}$)	0.4899	1.2224	1.1007
Number of Iterations: 876			
Error = 0.01		η = 0.8	

TABLE 2.10

Weights	w_0	w_1	w_2
Estimated	0.628	1.256	1.118
Number of Iterations: 1000			
Error = 0.058		η = 1	

CONCLUSION: From the above tables, we conclude that process converges in maximum 1000 iteration.

Example 2.5: Computation of 'NOT' Operator.

Note that 'NOT' operation is applied on a single operand, that is, it is an unary operator, therefore a set of input vector contains only one variable x. The training set for 'NOT' operator is given in Table 2.11

TABLE 2.11

x	0	1
NOT x	1	0

Consider the following ANN model useful for computation of 'NOT' operator

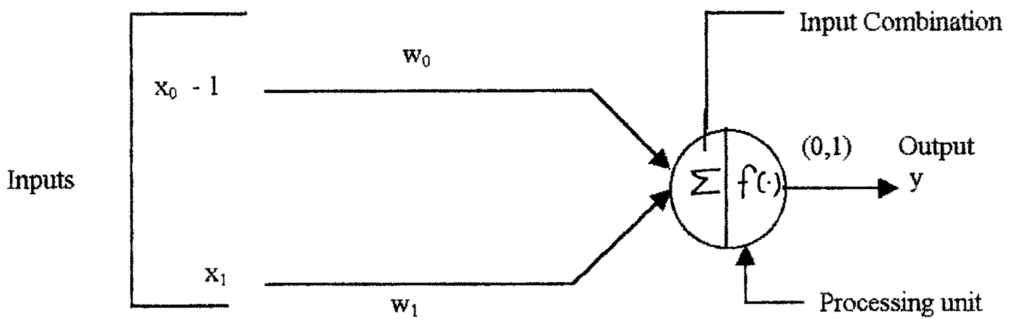

Figure 2.7 ANN Model For Implementation of 'NOT' Operator

Table 2.12 gives the estimated weights after execution of Program enclosed in Appendix B

TABLE 2.12

Weights	w_0	w_1
Estimated ($\hat{\underline{w}}$)	-3.7806	-3.7767
Number of Iterations: 617		
Error = 0.01	η = 0.9	

Like 'AND' and 'OR', here alsc we see that the ANN model correctly evaluates the 'NOT' operator.

In the next Chapter, we discuss how single layer ANN models can be useful for certain types of classification problems. Also, applications of single layer ANN in statistical data analysis will be discussed in Chapter IV.

■