UNIT 6 MULTI-LAYER PERCEPTRON

Structure **Page No**

6.1 INTRODUCTION

In this unit, we extend the Single Layer Neural Network architecture to the Multilayer Feedforward (MLFF) network with backpropagation (BP) learning. This network is also called multilayer perceptron. As its name suggests that this perceptron network has more than one layer. First, we briefly review the perceptron model discussed in unit 4 to show how this is altered to form MLFF networks in Sec. 6.2. In Section 6.3, we derive the generalized delta (backpropagation) learning rule and see how it is implemented in practice. Also, we shall examine the variations in the learning process to improve the efficiency, and ways to avoid some potential problems that can arise during training are described.

Objectives

After studying the unit you should be able to:

- define the multi-layer perceptron;
- formulate the multi-layer model for the given activation functions of input, hidden and output layers;
- implement the backpropagation algorithm.

6.2 MULTI LAYER PERCEPTRON

In Unit 5, Rosenblatts' perceptron i.e. single layer perceptron was introduced and the limitation of this with regard to the solution of linearly inseparable (or nonlinearly separable) problems was discussed.

The first approach to solve such linearly inseparable problems was to have more than one perceptron, each set up identifying small linearly separable sections of the inputs. Then, combining their outputs into another perceptron would produce a final indication of the class to which the input belongs.

Let us start with an example of XOR problem. The combination of perceptrons to solve the XOR problem is represented in Fig. 1. It looks that the arrangement shown in Fig. 1 can solve the problem. On investigation, it is formed that this arrangement of perceptrons in layers will be unable to learn. It is known that each neuron in the architecture takes the weighted sum of inputs, thresholds it and output 1/0. For any perceptron, in the first layer, the inputs come from the actual inputs of the problem, while for the perceptron in the second layer the inputs are nothing but the outputs of the first layer. The perceptrons of the second layer do not know which of the real inputs from the first layer were on or off.

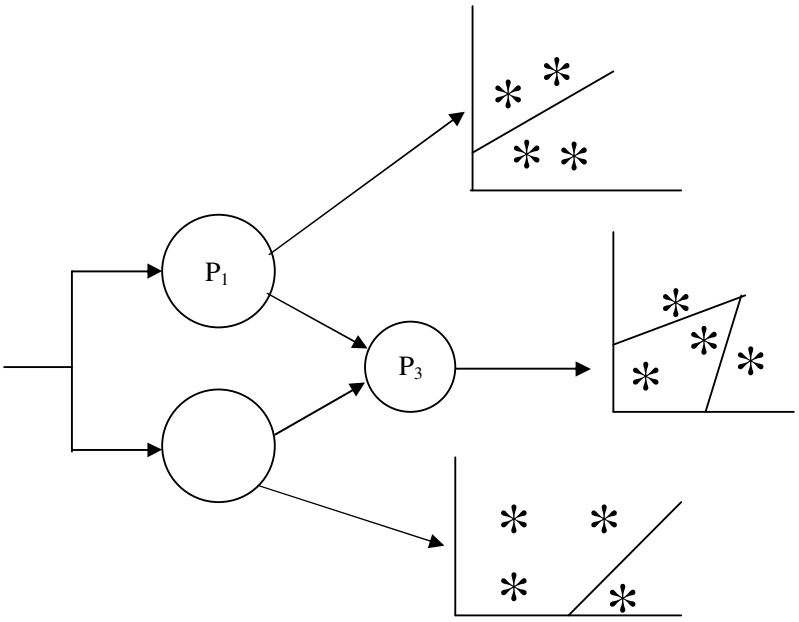


Fig. 1: Combination of perceptrons to solve XOR problem

It is impossible to strengthen the connections between active inputs and strengthen the correct parts of the network. The actual inputs are effectively masked off from the output units by the intermediate layer. The two states of neuron being on or off do not give us any indication of the scale by which we have to adjust the weights. These are shown in Fig. 2, where the threshold is adjusted at θ and at 0. The hard-hitting threshold functions remove the information that is needed if the network is to successfully learn. Hence, the network is not able to find which of the input weights are increased and which one are not and so. Therefore, the network is unable to work to produce a better solution next time. The way to go around the difficulty using the step function as the thresholding process is to adjust it slightly and to use a slightly different nonlinearity.

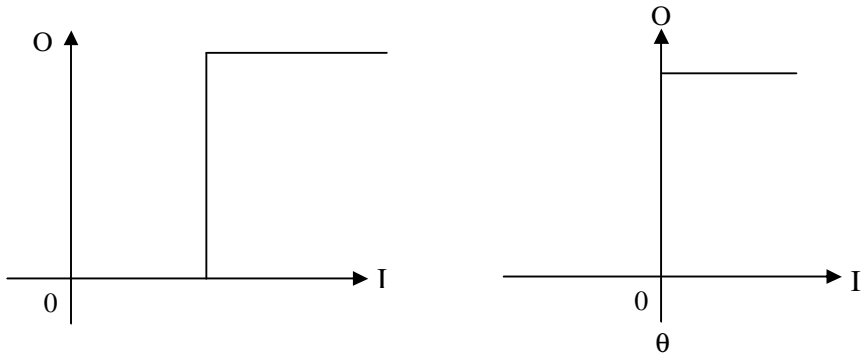


Fig. 2: "Step" or "Heaviside" functions

By smoothing the threshold function, we reach so that it turns on or off as before. It should have a sloping region in the middle that provides some information on the inputs. By this, strengthen or weaken the relevant weights can be determined. Hence, the network can learn as required. Here, $0 = 1$ if $I > \theta = 0$, $I < \theta$ where 0 is the output and I is the input. Also $1 < 0 < 1$ if $I = \theta$ and $u(t) = \sum_{i=1}^{n} w_i I_i$ notations of input and output the model of a neuron is shown in Fig. 3.

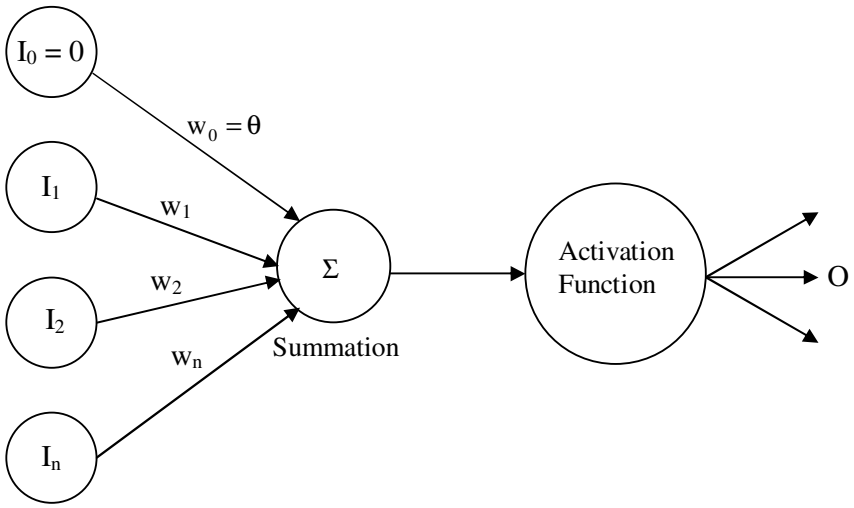


Fig. 3: An artificial neuron

Now let us define few nonlinear activation operators in addition to the earlier defined activation operators.

i) **Linear Function:** The output for linear function is o = gI where $g = \tan \phi$ and ϕ is the angle from the x-axis. The graph is shown in Fig. 4.

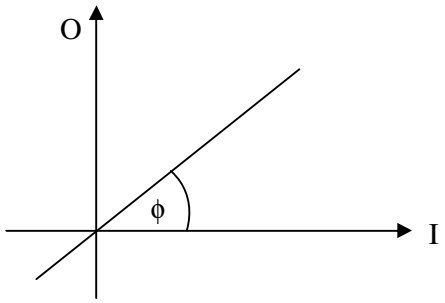


Fig. 4

ii) **Piecewise Linear Function:** The output for piecewise linear functions is defined as $O = \begin{Bmatrix} 1 & \text{if} & mI > 1 \\ gI & \text{if} & |mI < 1| \\ -1 & \text{if} & mI > -1 \end{Bmatrix}$. The corresponding graph is shown in Fig. 5.

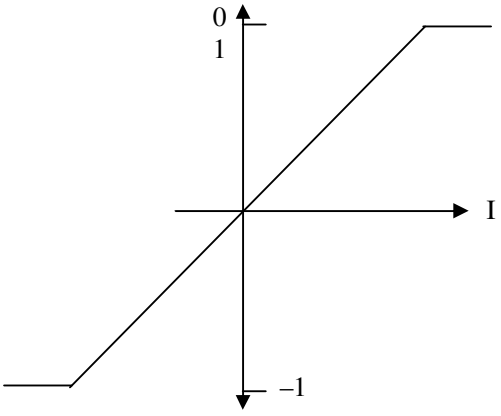


Fig. 5

iii) **Hard Limiter Function:** The output is given as $O = \text{sgn}[I]$, and the corresponding graph is shown in Fig. 6.

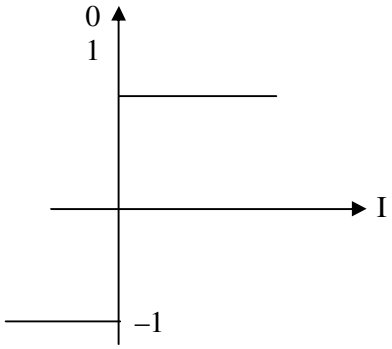


Fig. 6

iv) **Unipolar Sigmoidal and Bipolar Sigmoidal Function:** The output of unipolar sigmoidal function is given as $O = \frac{1}{(1+\exp(-\lambda I))}$ whereas the output of the bipolar function is given as $O = \tanh[\lambda I]$.

v) **Unipolar Multimodal Function:** The output is given as

$$O = \frac{1}{2}\left[1 + \frac{1}{M}\sum_{m=1}^{M} \tanh\left(g^m\left(I - W_O^m\right)\right)\right].$$

The graph is depicted in Fig. 7.

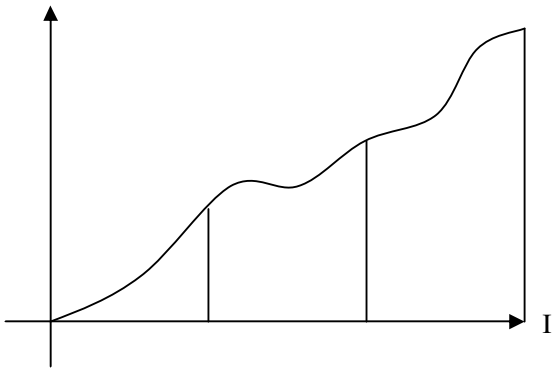


Fig. 7

vi) **Radial Basis Function (RBF):** The output is $O = \exp(I)$ where

$$I = \left[\frac{-\sum_{i=1}^{N} W_i(t) - X_i(t)^2}{2\sigma^2}\right].$$ This curve is same as the normal curve.

Considering threshold θ, the relative input to the neuron is given by

$$u(t) = W_1 I_1 + W_2 I_2 + \cdots + W_n I_n - \theta$$

$$= \sum_{i=0}^{n} W_i I_i \text{ where } W_0 = -\theta;\ I_0 = 1$$

And the output $O = f(u)$ where f is the nonlinear transfer function.

So far, we have discussed the mathematical details of a neuron at a single level. Although a single neuron can perform certain simple pattern detection problems, we need larger networks to offer greater computational capabilities. In order to mimic the layered structure of certain portions of the brain, let us explain the single feedforward neural network as shown in Fig. 8. Consider a single layer feedforward neural network shown in Fig. 8 consisting of an input layer to receive the inputs and an output layer to output the vectors respectively. The input layer consists of 'n' neurons and the output layer consists of 'm' neurons. Indicate the weight of the synapse connecting i^{th} input neuron to the j^{th} output neuron as W_{ij}. Now let us recall the single layer neural network with these notations, which is shown in Fig. 8. The inputs of the input layer and the corresponding outputs of the output layer are given as

$$\text{Consider} \rightarrow I_I = \begin{Bmatrix} I_{I1} \\ I_{I2} \\ \vdots \\ I_{Im} \end{Bmatrix}; \text{ and } O_O = \begin{Bmatrix} O_{O1} \\ O_{O2} \\ \vdots \\ O_{On} \end{Bmatrix}$$

Here input layer consists of m neurons and the output layer consists of n neurons that the input layer use linear transfer function and the output layer use unipolar sigmoidal function. Accordingly, $\underset{m\times 1}{\{O_I\}} = \underset{n\times 1}{\{I_I\}}$ or $I_{O_j} = \sum_{i=I}^{n} w_{ij}\, I_{Ii}$. The input to the output layer can be given as

$$[I_O] = [W]^t [O_I] = [W]^t [I_I]$$
$$n\times 1 \quad n\times m\, m\times 1 \quad n\times m\, m\times 1$$

$$O_{Ok} = \frac{1}{\left(1+e^{-\lambda I_{Ok}}\right)}$$

or $$[O_O] = f[WI]$$

where, λ is sigmoidal gain and $[W]$ is weight matrix and is also known as connection matrix.

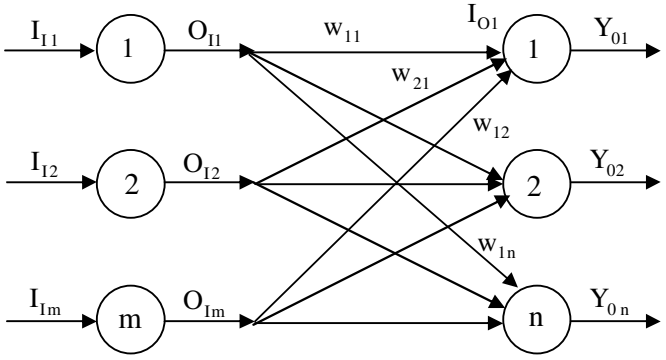


Fig. 8: Single layer feedforward neural network

The nonlinear activation function $f[WI]$ operates component wise on the activation values 'I' of each neuron, where each activation value is in turn a scalar product of the input with respect to weight.

The sigmoidal function is given as

$$f(I) = \frac{1}{\left(1+e^{-\lambda I}\right)}$$

and $$f'(I) = \lambda f(I)\left(I - f(I)\right)$$

Now let us extend the single layer model to multi layer model.

In multi layer perceptron the adapted perceptrons are arranged in layer. This model has three layers; an input layer, and output layer, and a layer in between the input and the output called the hidden layer. We use linear transfer function for the perceptrons in the input layer and sigmoidal or squashed-S functions for hidden layer and the output layer. The input layer does not perform a weighted sum or threshold. In this, we have modified the single layer perceptron by changing the nonlinearity from a step function to a sigmoidal function and added a hidden layer. This type of network recognizes more complex things. The input-output mapping of multilayer perceptron is given by

$$O = N_3\left[N_2\left[N_1[I]\right]\right]$$

where N_1, is the nonlinear mapping provided by input, N_2 is the nonlinear mapping provide by hidden layer and N_3 is the nonlinerar mapping provided by output layer. Multilayer perceptron provides many applications of neural networks, such as functional approximation, learning, generalization, etc. The multilayer network with one hidden layer is shown in Fig. 9. The activity of the output units are described by the activity of neurons in the hidden layer and the weight between the hidden and output layers and the neurons in the hidden layer is determined by the activities of the neurons in the input layer and the connecting weights between input and hidden units. In such networks neurons in the hidden layers are free to construct their own representations of the input.

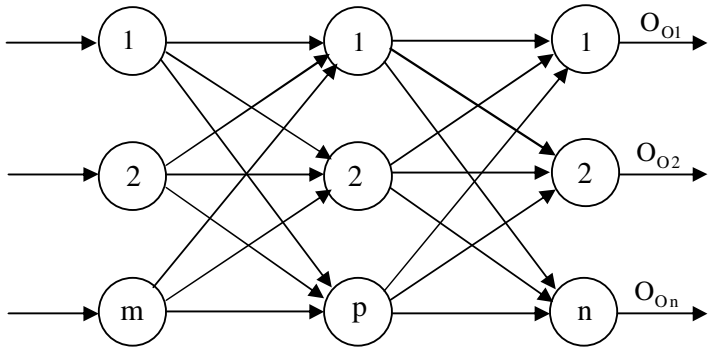


Fig. 9: Multilayer perceptron model

Now, we shall discuss MADALINE network, which is an example of multilayer perceptron. MADALINE is abbreviated for many ADALINE i.e. many ADALINE are connected to create such network. The MADALINE network helps countering the problem of non-linear separability. For example, the MADALINE network with two units can be applied to find a solution of the XOR problem. In its functioning the input bits x_1, x_2 are received by each unit of ADALINE and the bias input is assumed as 1 as its input. The calculated weighted sum of the inputs is passed on to the bipolar threshold units. The final output is obtained by computing the logical 'and' ing (bipolar) of the two threshold outputs. The computation of output is as given in Table 1.

Table 1

Threshold Outputs			Output
x_1	x_2		
+1	+1	Even parity	1
–1	–1		1
+1	–1	Odd parity	–1
–1	+1		–1

The corresponding network is shown in Fig. 10.

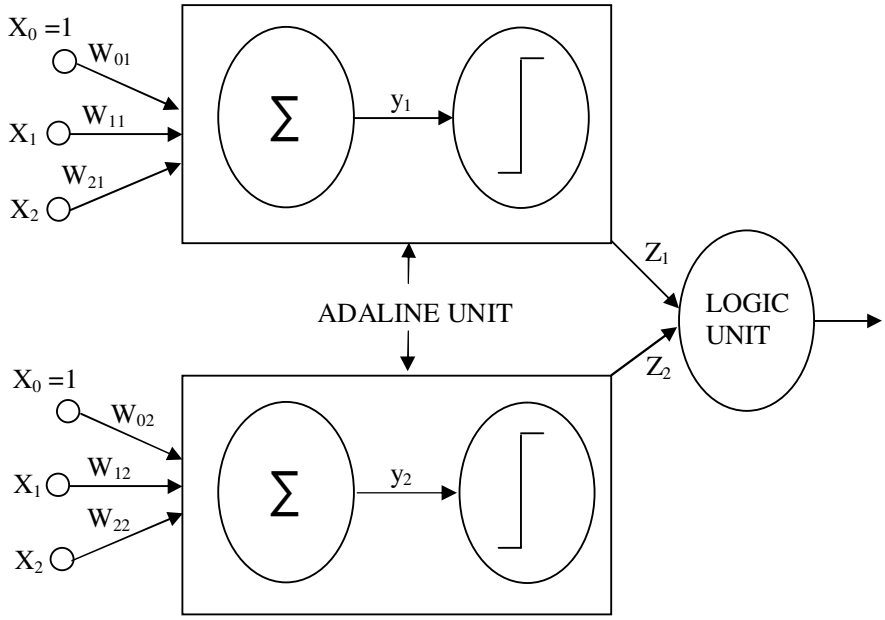


Fig. 10: A MADALINE network

Now, try the following exercises.

E1) Consider the weights w_{11}, w_{12}, w_{21} and w_{22} on connection from the input neurons to the hidden layer neurons and v_1, v_2 be the weights on the connections from the hidden layer neurons to the output neurons with following values. $w_{11} = 0.15$, $w_{12} = 0.3$, $w_{21} = 0.15$, $w_{22} = 0.3$, $v_1 = -0.3$ and $v_2 = 0.3$. Also consider the input (0, 0) generates the output (0, 0), (1,1) generates (1,1) and (1, 0) or (0, 1) generate (0, 1) as the hidden layer output.

i) Write the output of the neuron set at the hidden layer.

ii) Check whether the vectors in this set are linear separable or not. Give reason.

iii) Assume that input (0, 0) and (1, 1) gives 0, and (0, 1) generates 1 as output at outer layer, then draw the diagraph of this network.

iv) Obtain the activation of layers in the network.

E2) Show the decision boundaries of MADALINE to solve XOR problem.

E3) Consider the following table for the connections between the input neurons and the hidden layer neurons.

Input neuron	Hidden layer neurons	Connection weights
1	1	1
1	2	0.1
1	3	–1
2	1	1
2	2	–1
2	3	–1
3	1	0.2
3	2	0.3
3	3	0.6

The connections weights from Hidden layer neurons to the output neurons are 0.6, 0.3 and 0.6 for first, second and third neurons respectively corresponding threshold value for output layer is 0.5 and for hidden layer 1.8, 0.05 and – 0.2 for first, second and third neuron respectively,

i) Draw the diagraph of the network.

ii) Write the results of activation and interpret.

Now, in the following section we shall discuss the backpropagation learning.

6.3 BACKPROPAGATION LEARNING

Consider the network with input, hidden and output neurons, where the corresponding activities are denoted by writing subscript I, H, O for input, hidden and output neurons respectively. Also consider the weighted synapses from i^{th} hidden neurons to j^{th} input neuron, denoted by V_{ij}. The corresponding diagram is represented in Fig. 11.

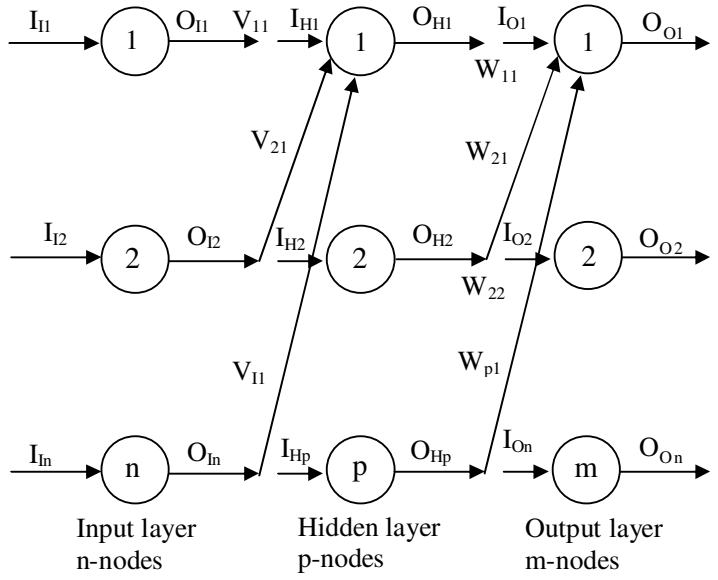


Fig. 11: Multilayer feedforward backpropagation network model

Let us assume the activation function for the output of the input layer to the input of input layer linear for this

$$\underset{n\times 1}{\{O\}_I} = \underset{n\times 1}{\{I\}_I}$$

input to the hidden neuron is the weighted sum of the outputs of the input neurons to get I_{Hp} (i.e. Input to the p^{th} hidden neuron) as

Therefore $$I_{Hp} = \sum_{i=1}^{n} V_{ip} O_{Ii}$$

where $$n = 1, 2, 3, \cdots, p$$

Denoting weight matrix or connectivity matrix between input neurons and hidden neurons as $\underset{l\times m}{[V]}$, we can get an input to the hidden neuron as

In the matrix notation, $\underset{p\times 1}{\{I\}_H} = \underset{p\times n}{[V]^T} \underset{n\times 1}{\{O\}_I}$

Let us again assume the activation function for the output of the p^{th} hidden neuron as sigmoidal function or squashed-S function. Then the output O_{Hp} is given by

$$O_{Hp} = \frac{1}{\left(1+e^{-\lambda\left(I_{Hp}-\theta_{Hp}\right)}\right)}$$

θ_{Hp} is the threshold of the p^{th} neuron. The computation of hidden layer is shown in Fig. 12.

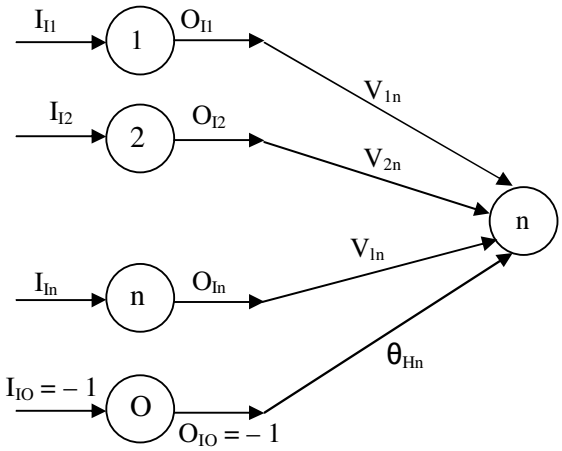


Fig. 12: Threshold in hidden layer

Here, the output of each component to the hidden neuron is given by

$$[O]_H = \left\{ \begin{array}{c} - \\ - \\ \frac{1}{\left(1+e^{-\lambda\left(I_{Hn}-\theta_{Hn}\right)}\right)} \\ - \\ - \end{array} \right\}$$

The input to the i^{th} output neuron

$$I_{Oi} = \sum_{j=1}^{i} W_{ji} O_{Hj} \qquad i = 1,2,3,\cdots m$$

The matrix notation is

$$\underset{m\times 1}{[I]_O} = \underset{m\times p}{[W]^T} \underset{p\times 1}{[O]_H}$$

Considering the output of the q^{th} output neuron O_{Oi} is given by

$$O_{Oi} = \frac{1}{\left(1+e^{-\lambda\left(I_{Oi}-\theta_{Oi}\right)}\right)} \text{ (considering the sigmoidal function)}$$

where θ_{Oq} is the threshold of the q^{th} neuron. This threshold is shown in Fig. 13.

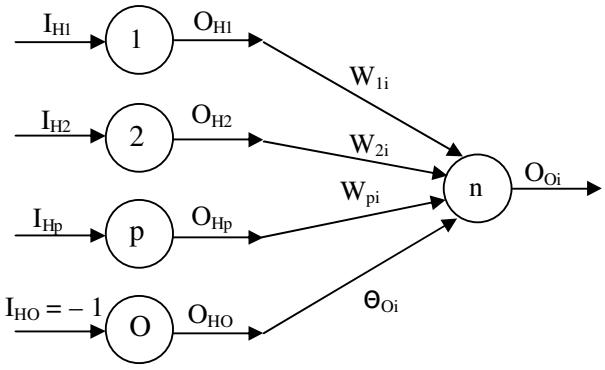


Fig. 13: Threshold in output layer

Similarly, the outputs of output neurons are given by

$$[O]_H = \begin{Bmatrix} - \\ - \\ \dfrac{1}{\left(1+e^{-\lambda\left(I_{Oi}-\theta'_{Oi}\right)}\right)} \\ - \\ - \end{Bmatrix}$$

Now let us calculate the error.

The Euclidean norm of error E_1^0 for the first training pattern is given by

$$\sum_{i=1}^{n} E_{1i} = \frac{1}{2} \sum_{i=1}^{n} \left(O_T - O_C\right)^2$$

where E_{1i} is the error in the i^{th} neuron for the first training patterns, and O_T is the target output and O_C is the calculated output.

Using the error, let us now write the modified values of weight vectors using steepes descent method.

$$[V]^{t+1} = [V]^t + [\Delta V]^{t+1}$$
$$[W]^{t+1} = [W]^t + [\Delta W]^{t+1}$$

where

$$[\Delta w]^{t+1} = \alpha[\Delta w]^t + \eta[y],$$
$$p\times m \qquad p\times m \qquad p\times m$$
$$[\Delta v]^{t+1} = \alpha[\Delta v]^t + \eta[x], \quad \text{here } \eta \text{ is learning rate.}$$
$$n\times p \qquad n\times p \qquad n\times p$$
$$[y] = [o]_H \quad \left[d_y\right]'$$
$$p\times m \quad p\times 1 \quad 1\times m'$$
$$\left[d_y\right]_{m\times 1} = \begin{bmatrix} - \\ - \\ \left(O_T - O_{Ok}\right) O_{Ok}\left(1-O_{Ok}\right) \\ - \end{bmatrix}$$

$$\underset{1\times p}{[X]} = \underset{1\times 1}{[o]_I} \quad \underset{1\times p}{[d_x]} = \underset{1\times 1}{[I]_I} \quad \underset{1\times p}{[d_x]'}$$

$$[d_x]_{p\times 1} = \begin{bmatrix} - \\ - \\ e_i [O_{Hi}](1-O_{Hi}) \\ - \\ - \end{bmatrix}$$

and

$$[e]_{p\times 1} = [w]_{p\times m} \; [d_y]_{m\times 1}$$

The weights and threshold may be updated as

$$[W]^{t+1} = [W]^t + [\Delta W]^{t+1}$$
$$[V]^{t+1} = [V]^t + [\Delta V]^{t+1}$$
$$[\theta]_O^{t+1} = [\theta]_O^t + [\Delta\theta]_O^{t+1}$$
$$[\theta]_H^{t+1} = [\theta]_H^t + [\Delta\theta]_H^{t+1}$$

Now, after defining the backpropagation network let us take the backpropagation algorithm for this. Let us consider the three-layer network with input layer having 'n' nodes, hidden layer having 'p' nodes, and an output layer with 'm' nodes. We consider sigmoidal functions for activation functions for the hidden and output layers and linear activation function for input layer. Now let us write the backpropagation algorithm stepwise.

Step 1: Initialize the weights V and W usually from –1 to 1.

Step 2: For each training pair, assume there are 'n' inputs given by $\underset{n\times 1}{[I]_I}$ and 'm' output $\underset{m\times 1}{[O]_O}$ in a normalized form.

Step 3: Set the number of neurons in the hidden layer to lie between $1 < p < 21$.

Step 4: Set $\lambda = 1$ and threshold value 0.

Step 5: Compute the output of the input layer.

Step 6: Compute the inputs and outputs to the hidden layer.

Step 7: Compute the inputs and outputs to the output layer.

Step 8: Calculate the error.

Step 9: Find

$$[V]^{t+1} = [V]^t + [\Delta V]^{t+1}$$
$$[W]^{t+1} = [W]^t + [\Delta W]^{t+1}$$

Step 10: Repeat steps 5 to 9 until the convergence in the error rate is less than the tolerance value.

Let us cite the following example to understand the algorithm.

Example 1: Consider the three training sets given in the following Table 2,

Table 2

Inputs		Output
I_1	I_1	O
0.3	– 0.2	0.2
0.4	0.6	0.3
0.6	– 0.2	0.1

with the initial vectors $[W]^0 = \begin{bmatrix} 0.2 \\ -0.5 \end{bmatrix}$ and $[V]^0 = \begin{bmatrix} 0.1 & 0.4 \\ -0.2 & 0.2 \end{bmatrix}$.

Solution: Let us find the improved weights. The architecture of this model is given in Fig. 14.

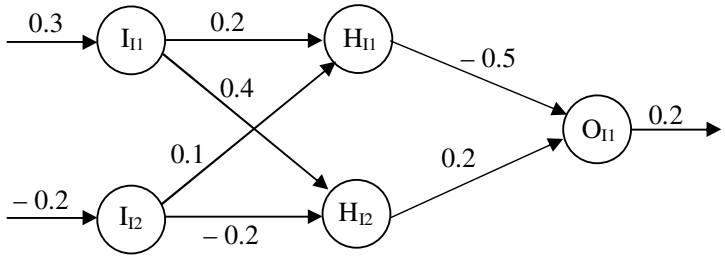


Fig. 14: Multilayer architecture

$$[I]_H = [V]^T \quad [O]_I$$

$$= \begin{bmatrix} 0.2 & 0.1 \\ 0.4 & -0.2 \end{bmatrix} \begin{bmatrix} 0.3 \\ -0.2 \end{bmatrix} = \begin{bmatrix} 0.04 \\ 0.16 \end{bmatrix}$$

$$[O_C]_H = \begin{bmatrix} \frac{1}{1+e^{-0.04}} \\ \frac{1}{1+1e^{-0.16}} \end{bmatrix} = \begin{bmatrix} 0.51 \\ 0.54 \end{bmatrix}$$

$$[I]_O = [w]^t [O]_H = [-0.5 \quad 0.2] \begin{bmatrix} 0.51 \\ 0.52 \end{bmatrix}$$

$$= -0.15$$

$$[O_C]_O = \frac{1}{1+e^{0.151}} = 0.462$$

$$\text{error} = [O_{TO} - O_{CO}]^2$$

$$= [0.2 - 0.462]^2 = 0.069$$

$$d = (O_{TO} - O_{CO1})(O_{CO1})$$

$$= (0.2 - 0.462)(0.462)(1 - 0.462) = -0.065$$

$$[O_C]_H [d] = \begin{bmatrix} -0.033 \\ -0.035 \end{bmatrix}$$

Here assume that $\alpha = 1$ and $\eta = 0.5$

$$[\Delta w]^1 = \alpha [\Delta w]^0 + \eta [y]$$

$$= \begin{bmatrix} -0.517 \\ 0.182 \end{bmatrix}$$

$$e = [w][d] = \begin{bmatrix} 0.032 \\ -0.013 \end{bmatrix}$$

$$d^* = \begin{bmatrix} -0.001 \\ 0.0004 \end{bmatrix}$$

$$X = [O]_i \left[d^*\right]^1 = \begin{bmatrix} 0.0003 & 0.0001 \\ 0.0002 & -0.00009 \end{bmatrix}$$

$$[V]^1 = \begin{bmatrix} 0.1998 & 0.40007 \\ 0.1001 & -0.20004 \end{bmatrix}$$

and $[W]^1 = \begin{bmatrix} -1.01 \\ 0.38 \end{bmatrix}$

Using these modified $[V]^1$ and $[w]^1$, error is calculated and then can be processed for next training set.

Now, try the following exercises.

E4) Find the modified weights for the training set having input $I_1 = 0.3$, $I_2 = -0.5$ and output = 0.1 with $[V]^0 = \begin{bmatrix} 0.1 & 0.4 \\ -0.2 & 0.2 \end{bmatrix}$ and $[W]^0 = \begin{bmatrix} 0.2 \\ -0.5 \end{bmatrix}$.

E5) How many Layers are there in Multi layer Neural Network?

E6) What is cycle to modify the weight values?

Now, let us summarize the unit.

6.4 SUMMARY

In this unit, we have covered the following points.

i) The neural network architectures are broadly classified as single layer feed forward networks and multilayer feed forward networks. If only input and output layers are present, then the network is single layer network. In turn, if in addition to the input and output layer one or more intermediate layer exists, then the network is multilayer network.

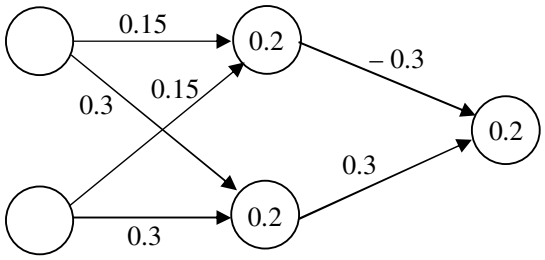
ii) Backpropagation is a systematic method of training multilayer neural networks. It is built on high mathematical foundation and has very good application potential.

6.5 SOLUTIONS/ANSWERS

E1) i) Output set = {(0, 0), (1, 1), (0, 1)}

ii) These three vectors given in (i) are separable with (0, 0) and (1, 1) on one side of the separating line, while (0, 1) is on the other side.

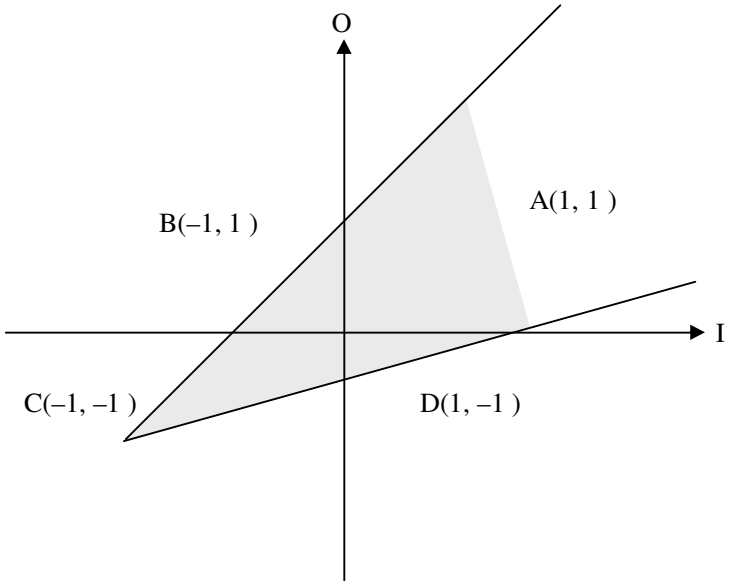
iii)



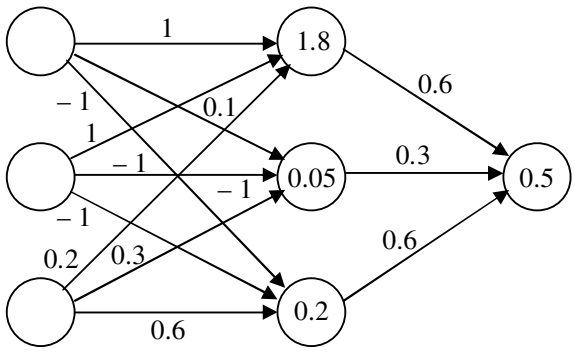
iv)

Input	Activation (Hidden layer)	Output (Hidden layer)	Output neuron activation	Output of the network
(0, 0)	(0, 0)	(0, 0)	0	0
(1, 1)	(0.3, 0.6)	(1, 1)	0	0
(0, 1)	(0.15, 0.3)	(0, 1)	0.3	1
(1, 0)	(0.15, 0.3)	(0, 1)	0.3	1

E2)



E3) i)



ii) Let us consider the vertexes of a cube be O, A, B, C, D, E, F and G be (0, 0, 0,), (0, 0, 1), (0, 1, 0), (0, 1, 1,), (1, 0, 0), (1, 0, 1), (1, 1, 0) and (1, 1, 1).

Vertex/ Coordinates	Hidden Layer Neuron	Weighted Sum	Comment	Activation	Contribution to output	Sum
O: 0, 0, 0	1	0+0+0 = 0	< 1.8	0	0	
	2	0+0+0 = 0	< 0.05	0	0	
	3	0+0+0 = 0	> –0.2	1	0.6	0.6*
A: 0, 0, 1	1	0+0+0.2 = 0.2	< 1.8	0	0	
	2	0+0+0.3 = 0.3	> 0.05	1	0.3	
	3	0+0+0.6 = 0.6	> –0.2	1	0.6	0.9*
B: 0, 1, 0	1	0+1+0 = 1	< 1.8	0	0	
	2	0–1+0 = –1	< 0.05	0	0	
	3	0–1+0 = –1	< –0.2	0	0	0
C: 0, 1, 1	1	0+1+0.2 = 1.2	< 1.8	0	0	
	2	0+0.1+0.2 = 0.2	> 0.05	1	0.3	
	3	0–1+0.6 = – 0.4	< – 0.2	0	0	0.3
D: 1, 0, 0	1	1+0+0 = 1	< 1.8	0	0	
	2	0.1+0+0 = 0.1	> 0.05	1	0.3	
	3	–1+0+0=–1	< –0.2	0	0	0.3
E: 1, 0, 1	1	1–0+0.2= 1.2	< 1.8	0	0	
	2	0.1+0+0.2 = 0.4	> 0.05	1	0.3	
	3	–1+0+0.6 = –0.4	< –0.2	0	0	0.3
F: 1, 1, 0	1	1+1+0 = 2	> 1.8	1	0.6	
	2	0.1–1+0 = – 0.9	< 0.05	0	0	
	3	–1–1+0 = –2	< – 0.2	0	0	0.6*
G: 1, 1, 1	1	1+1+0.2 = 2.2	> 1.8	1	0.6	
	2	0.1–1+0.3 = – 0.8	< 0.05	0	0	
	3	–1–1+0.6 = – 1.4	< – 0.2	0	0	0.6*

* The output neuron fires, as this value is greater than 0.5 (the threshold value); the function value is + 1.

E4) The following values are obtained:

$$[V]^0 = \begin{bmatrix} 0.1 & 0.4 \\ -0.2 & 0.2 \end{bmatrix}$$

$$[W]^0 = \begin{bmatrix} 0.2 \\ -0.5 \end{bmatrix}$$

$$[O]_I = \begin{bmatrix} 0.3 \\ -0.5 \end{bmatrix}$$

$$[I]_H = \begin{bmatrix} 0.13 \\ 0.02 \end{bmatrix}$$

$$[O_C]_H = \begin{bmatrix} 0.53 \\ 0.50 \end{bmatrix}$$

$$[I] = [-0.146]$$

$[O_C]_O = 0.464$

error = 0.132

assuming $[\alpha = 0.5, \eta = 0.5]$

d = – 0.090

$$[Y] = \begin{bmatrix} -0.048 \\ -0.0456 \end{bmatrix}$$

$$[\Delta W]^1 = \begin{bmatrix} 0.076 \\ -0.273 \end{bmatrix}$$

$$e = \begin{bmatrix} -0.018 \\ 0.045 \end{bmatrix}$$

$$d^* = \begin{bmatrix} -0.001 \\ -0.002 \end{bmatrix}$$

$$X = \begin{bmatrix} -0.0003 & -0.0006 \\ 0.0005 & 0.0012 \end{bmatrix}$$

$$[V]^1 = \begin{bmatrix} 0.0498 & 0.1997 \\ -0.0997 & 0.1005 \end{bmatrix}$$

$$[W]^1 = \begin{bmatrix} 0.276 \\ -0.773 \end{bmatrix}$$

6.6 PRACTICAL ASSIGNMENT

Session 7

Write a program in 'C' language to implement the backpropagation algorithm. Show step by step output to input, hidden and output neurons as well as errors. How the weights W and V modified? Use data given in Example 1.

6.7 REFERENCES

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