CHAPTER 4

Perceptron Networks

What You Will Learn

- How the perceptron learning rule is better than the Hebb rule.
- Layer structure in the original perceptrons.
- Learning and training algorithms in the perceptron network .
- Architecture, algorithm and the application procedure of the perceptron net.
- Derivation of perceptron algorithm for several output classes .
- Applications of multilayer perceptrons.

4.1 Introduction

Frank Rosenblatt [1962], and Minsky and Papert [1988], developed large class of artificial neural networks called Perceptrons. The perceptron learning rule uses an iterative weight adjustment that is more powerful than the Hebb rule. The perceptrons use threshold output function and the McCulloch-Pitts model of a neuron. Their iterative learning converges to correct weights, i.e. the weights that produce the exact output value for the training input pattern. The original perceptron is found to have three layers, sensory, associator and response units as shown in Fig. 4.1.

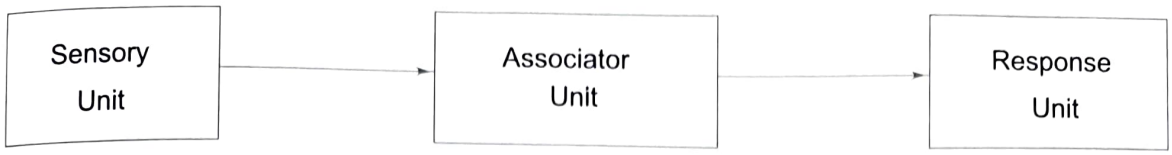


Fig. 4.1 | *Original Perceptron*

The sensory and association units have binary activations and an activation of + 1, 0 or – 1 is used for the response unit. All the units have their corresponding weighted interconnections. Training in perceptron will continue until no error occurs. This net solves the problem and is also used to learn the classification. The perceptrons are of two types: single layer and multi layer perceptrons. A detailed study about the perceptron networks is made in this chapter.

4.2 Single Layer Perceptron

A single layer perceptron is the simplest form of a neural network used for the classification of patterns that are linearly separable. Fundamentally, it consists of a single neuron with adjustable weights and bias. Rosenblatt found that if the patterns used to train the perceptron are drawn from two linearly separable classes, the perceptron algorithm converges and positions the decision surface in the form of a hyperplane between the two classes. The perceptron built around a single neuron is limited to performing pattern classification with only two classes. Also classes have to be linearly separable for the perception to work properly.

The basic concept of a single layer perceptron as used in pattern classification is that, it is concerned with only a single neuron. The linearity and the integrity learning makes the perceptron network very simple. Training in the perceptron continues till no error occurs.

4.2.1 Architecture

The architecture of the single layer perceptron is shown in Fig. 4.2.

As we have already studied, the perceptron has sensory, associator and response units. The input to the response unit will be the output from the associator unit, which is a binary vector. Since only the weight between the associator and the response unit is adjusted, the concept is limited to single layer network as discussed in Section 2.7.

In the architecture shown in Fig. 4.2, only the associator unit and the response unit is shown. The sensor unit is hidden, because only the weights between the associator and the response unit are adjusted. The input layer consists of input neurons from $X_1 \cdots X_i \cdots X_n$. There always exists a common bias of '1'. The input neurons are connected to the output neurons through weighted interconnections. This is a single layer network because it has only one layer of interconnections between the input and the output neurons. This network perceives the input signal received and performs the classification.

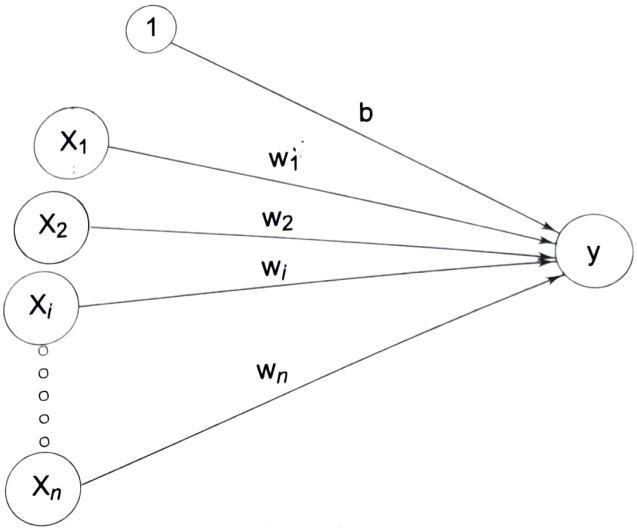


Fig. 4.2 *Architecture of Single Layer Perceptron*

4.2.2 Algorithm

To start the training process, initially the weights and the bias are set to zero. The initial weights of the network can be formulated from other techniques like Fuzzy systems, Genetic Algorithm etc. It is also essential to set the learning rate parameter, which ranges between 0 to 1. Then the input is presented, the net input is calculated by multiplying the weights with the inputs and adding the result with the bias entity. Once the net input is calculated, by applying the activation function the output of the network is also obtained. This output is compared with the target, where if any difference occurs, we go in for weight updation based on perceptron learning rule, else the network training is stopped. The algorithm can be used for both binary and bipolar input vectors. It uses a bipolar target with fixed threshold and adjustable bias.

The training algorithm is as follows:

Step 1: Initialize weights and bias (initially it can be zero). Set learning rate α(0 to 1).

Step 2: While stopping condition is false do Steps 3–7.

Step 3: For each training pair s:t do Steps 4–6.

Step 4: Set activations of input units.

$x_i = s_j$ for $i = 1$ to n

Step 5: Compute the output unit response.

$$Y_{in} = b + \sum_i X_i W_i$$

The activation function used is,

$$y = f(y_{in}) = \begin{cases} 1, & \text{if } y_{in} > \theta \\ 0, & \text{if } -\theta \le y_{-in} \le \theta \\ -1, & \text{if } y_{-in} < -\theta \end{cases}$$

Step 6: The weights and bias are updated if the target is not equal to the output response.

If $t \neq y$ and the value of x_i is not zero.

$$w_{i(new)} = w_{i(old)} + \alpha t x_i$$
$$b_{(new)} = b_{(old)} + \alpha t.$$

else

$$w_{i(new)} = w_{i(old)}$$
$$b_{(new)} = b_{(old)}.$$

Step 7: Test for stopping condition.

The stopping conditions may be the weight changes.

Note:

1. Only weights connecting active input units ($x_i \neq 0$) are updated.
2. Weights are updated only for patterns that do not produce the correct value of y.

4.2.3 Application Procedure

This procedure enables the user to test the network performance. The network should be trained with sufficient number of training data and using the testing data its performance can be tested. The application procedure used for testing perceptron network is as follows.

Step 1: The weights to be used here are taken from the training algorithm.

Step 2: For each input vector x to be classified do Steps 3 – 4.

Step 3: Input units activations are set.

Step 4: Calculate the response of output unit,

$$y_{-in} = \sum_i x_i w_i$$

$$y = f(y_{-in}) = \begin{cases} 1, & \text{if } y_{-in} > \theta \\ 0, & \text{if } -\theta \leq y_{-in} \leq \theta \\ -1, & \text{if } y_{-in} < -\theta \end{cases}$$

4.2.4 Perception Algorithm for Several Output Classes

The perceptron network for single output class is extended for several output classes. Here there exist more number of output neurons, but the weight updation in this case also is based on the perceptron learning rule. The algorithm is as follows:

Step 1: Initialize the weights and biases. Set the learning rate

Step 2: When stopping condition is false, perform Steps 3 – 7.

Step 3: For each input training pair, do Steps 4 – 6.

Step 4: Set activation for the input units,

$x_i = s_i$ for $i = 1$ to n

Step 5: Compute the activation output of each output unit $y_{-inj} = b_j + \sum_i x_i w_i$ for $j = 1$ to m.

$$y_j = f(y_{-inj}) = \begin{cases} 1, & \text{if } y_{-inj} > 0 \\ 0, & \text{if } -\theta \le y_{-inj} \le \theta \\ -1, & \text{if } y_{-inj} < -\theta \end{cases}$$

Step 6: The weights and bias are to be updated for $j = 1$ to m and $i = 1$ to n.

If $y_j \neq t_j$ and $x_i \neq 0$, then

$$w_{ij(new)} = w_{ij(old)} + \alpha t_j x_i$$
$$b_j(new) = b_j(old) + \alpha t_j.$$

else if $y_j = t_j$

$$w_{ij(new)} = w_{ij(old)}$$
$$b_j(new) = b_j(old).$$

That is, the biases and weights remain unchanged.

Step 7: Test for stopping condition.

The stopping condition may be the weight changes.

Solved Examples

Example 4.1 Develop a perceptron for the AND function with bipolar inputs and targets.

Solution The training pattern for AND function can be,

Input			Target
X_1	X_2	b	t
1	1	1	1
−1	1	1	−1
1	−1	1	−1
−1	−1	1	−1

Step 1: Initial weights $w_1 = w_2 = 0$ and $b = 0$, $\alpha = 1$, $\theta = 0$.

Step 2: Begin computation.

Step 3: For input pair (1, 1): 1, do Steps 4–6

Step 4: Set activations of input units

$$x_i = (1, 1).$$

Step 5: Calculate the net input.

$$y_{-in} = b + \sum x_i w_i = 0 + 1 \times 0 + 1 \times 0 = 0$$

Applying the activation,

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

Therefore $y = 0$.

Step 6: $t = 1$ and $y = 0$

Since $t \neq y$, the new weights are,

$$w_{i(new)} = w_{i(old)} + \alpha t x_i$$

$$w_{1(new)} = w_{i(old)} + \alpha t x_1 = 0 + 1 \times 1 \times 1 = 1$$

$$w_{2(n)} = w_{2(o)} + \alpha t x_2 = 0 + 1 \times 1 \times 1 = 1$$

$$b_{(new)} = b_{(old)} + \alpha t$$

$$b_{(n)} = b_{(0)} + \alpha t = 0 + 1 \times 1 = 1$$

The new weights and bias are [1 1 1].

The algorithmic steps are repeated for all the input vectors with their initial weights as the previously calculated weights.

By presenting all the input vectors, the updated weights are shown in table below:

Input			**Net**	**Output**	**Target**	**Weight Changes**			**Weights**		
x_1	x_2	B	y_{in}	y	t	Δw_1	Δw_2	Δb	w_1	w_2	B
								(0	0	0)	
1	1	1	0	0	1	1	1	1	1	1	1
−1	1	1	1	1	−1	1	−1	−1	2	0	0
1	−1	1	2	1	−1	−1	1	−1	1	1	−1
−1	−1	1	−3	−1	−1	0	0	0	1	1	−1

This completes one epoch of the training.

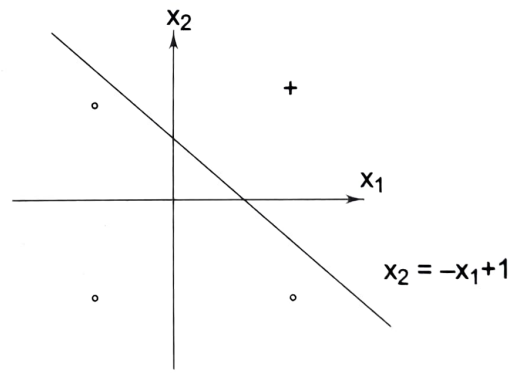
The final weights after the first epoch is completed are, $w_1 = 1$, $w_2 = 1$, $b = -1$

We know that $b + x_1 w_1 + x_2 w_2 = 0$

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

$$x_2 = -x_1 \frac{1}{1} - \frac{(-1)}{1}$$

$x_2 = -x_1 + 1$ is the separating line equation.



The decision boundary for AND function trained by perceptron network is given as,

In a similar way, the perceptron network can be developed for logic functions OR, NOT, AND NOT etc.

Example 4.2 Develop a perceptron for the AND function with binary inputs and bipolar targets without bias up to 2 epochs. (Take first with (0,0) and next without (0,0)).

Solution Initializing the weights to be, $w_1 = w_2 = 0$ and the bias is neglected here (because the problem is stated without bias). Hence $\alpha = 1$ and threshold $\theta = 0$.

(a) *With (0,0) and without bias.*

The net input is, $Y_{in} = \sum x_i w_i$

$$y = f(y_{\_in}) = \begin{cases} 1, & \text{if } y_{\_in} > 0 \\ 0, & \text{if } -0 \le y_{\_in} \le 0 \\ -1, & \text{if } y_{\_in} < -0 \end{cases}$$

The weight change,

$$\Delta w_i = \propto t x_i \text{ and}$$

New weight is,

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

Epoch 1:

Input		Net	Output	Target	Weight Changes		Weights	
x_1	x_2	y_{in}	y	t	Δw_1	Δw_2	W_1 (0	W_2 0)
1	1	0	0	1	1	1	1	1
1	0	1	1	−1	−1	0	0	1
0	1	1	1	−1	0	−1	0	0
0	0	0	0	−1	0	0	0	0

The separating line for 1st and 2nd input are, $x_1 + x_2 = 0$ and $x_2 = 0$ respectively.

Epoch 2:

The initial weights used are the final weights from the pervious iteration.

Input		Net	Output	Target	Weight Changes		Weights	
x_1	x_2	y_{in}	y	t	Δw_1	Δw_2	W_1 (0	W_2 0)
1	1	0	0	1	1	1	1	1
1	0	1	1	−1	−1	0	0	1
0	1	1	1	−1	0	−1	0	0
0	0	0	0	1	0	0	0	0

Without bias for the given inputs, the final weights obtained are same as that for with bias and the equation of separating line also remains same. Thus the equations remain same and are given by,

for 1st input : $x_1 + x_2 = 0$

for 2nd input : $x_2 = 0$

(b) *Without bias and (0,0)*

Epoch 1:

Input		Net	Output	Target	Weight Changes		Weights	
x_1	x_2	y_{in}	y	t	Δw_1	Δw_2	W_1 (0	W_2 0)
1	1	0	0	1	1	1	1	1
1	0	1	1	−1	−1	0	0	1
0	1	1	1	−1	0	−1	0	0

In this case, also the final weights are (0,0) and the separating line are $x_2 = -x_1$ and $x_2 = 0$.

Epoch 2:

The final weights from Epoch 1 are used here as initial weights.

Input		Net	Output	Target	Weight Changes		Weights	
x_1	x_2	y_{in}	y	t	Δw_1	Δw_2	W_1	W_2
							(0	0)
1	1	0	0	1	1	1	1	1
1	0	1	1	– 1	– 1	0	0	1
0	1	1	1	– 1	0	– 1	0	0

Here also, the weights are same as that of previous epoch.

The separating line here also, without bias

1st input

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

$$x_2 = -x_1$$

2nd input

$$x_2 = -x_1 \frac{w_1}{w_2} = -x_1 \cdot \frac{0}{1} = 0$$

$$x_2 = -0$$

Thus from all this, it is clear that without bias the convergence does not occur. Even after neglecting (0, 0), the convergence does not occur.

Example 4.3 Using the perceptron learning rule, find the weights required to perform the following classifications. Vectors (1 1 1 1), (– 1 1 – 1 – 1) and (1 – 1 – 1 1) are members of class (having target value 1); vectors (1 1 1 – 1) and (1 – 1 – 1 1) are not members of class (having target value – 1). Use learning rate of 1 and starting weights of 0. Using each of the training and vectors as input, test the response of the net.

Solution The initial weights are assumed to be zero and the learning rate as 1.

The updation is done according to perceptron learning rule,

If $y \neq t$, weight change, $\Delta w = \propto t x_i$ & $\Delta b = \propto t$.

New weights are, $w_{(new)} = w_{(old)} + \Delta w$

$b_{(new)} = b_{(old)} + \Delta b$

If t = y, no weight change

By using the above, the below tabulation is formed, where, $y_{-in} = b + \sum_i x_i w_i$

and $y = f(y_{in})$ is the activation applied.

Input					Net	Output	Target	Weight Changes					Weights				
x_1	x_2	x_3	x_4	1	y_{in}	y	t	Δw_1	Δw_2	Δw_3	Δw_4	Δb	w_1	w_2	w_3	w_4	b
													(0	0	0	0	0)
Epoch 1:																	
1	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1
1	1	1	−1	1	3	1	−1	−1	−1	−1	1	−1	0	0	0	2	0
−1	1	−1	−1	1	0	0	1	−1	1	−1	−1	1	−1	1	−1	−1	1
1	−1	−1	1	1	−1	−1	−1	0	0	0	0	0	−1	1	−1	−1	1
												initial →	0	0	0	2	0
Epoch 2:																	
1	1	1	1	1	2	1	1	0	0	0	0	0	0	0	0	2	0
1	1	1	−1	1	−2	−1	−1	0	0	0	0	0	0	0	0	2	0
													−1	1	−1	−1	1
	Initial →																
−1	1	−1	−1	1	5	1	1	0	0	0	0	0	−1	1	−1	−1	1
1	−1	−1	1	1	−1	−1	−1	0	0	0	0	0	−1	1	−1	−1	1

The final weights from Epoch 1 are used as the initial weights for Epoch 2. Thus the output is equal to target by training for suitable weights.

Testing the response of the net

The final weights are,

For the 1st set of input, $w_1 = 0$, $w_2 = 0$, $w_3 = 0$, $w_4 = 2$, $b = 0$, and

For the 2nd set of input, $w_1 = -1$, $w_2 = 1$ $w_3 = -1$, $w_4 = -1$, $b = 1$

The net input is, $y_{in} = b + \sum x_i w_i$

For the 1st set of inputs,

(i) (1 1 1 1 1)

$$y_{-in1} = 0 + 0 \times 1 + 0 \times 1 + 0 \times 1 + 2 \times 1 = 2 > 0$$

Applying activation, $y_1 = f(y_{-in1}) = 1.$

(ii) (1 1 1 −1 1)

$$y_{-in1} = 0 + 0 \times 1 + 0 \times 1 + 0 \times 1 + 2 \times -1 = -2 < 0$$

Applying activation, $y_2 = f(y_{-in2}) = -1.$

For 2nd set of inputs,

(i) (−1 1 −1 −1 1)

$$y_{-in1} = 1 + -1 \times -1 + 1 \times 1 + -1 \times -1 + -1 \times -1 + 1 \times 1 = 5 > 0$$

Applying activation, $y_1 = f(y_{-in1}) = 1$

(ii) (1 −1 −1 1 1)

$$y_{-in2} = 1 + -1 \times 1 + 1 \times -1 + -1 \times -1 + -1 \times 1 + -1 \times 1 = -1 < 0$$

Applying activations, $y_2 = f(y_{-in2}) = -1.$

Hence the calculated test output values matches with the target output values for the corresponding input vectors.

Example 4.4 For the following noisy versions of training patterns, identify the response of network by saggregating it into correct, incorrect or indefinite.

(0 –1 1), (0 1 –1), (0 0 1), (0 0 –1), (0 1 0), (1 0 1),

(1 0 –1), (1 –1 0), (1 0 0), (1 1 0), (0 –1 0), (1 1 1)

Solution The concept used for this problem is

If $x_1w_1 + x_2w_2 + x_3w_3 > 0$, then the response is correct.

If $x_1w_1 + x_2w_2 + x_3w_3 < 0$, then the response is incorrect.

If $x_1w_1 + x_2w_2 + x_3w_3 = 0$, then the response is indefinite or undetermined.

Say if the weights taken from the bipolar step function are, $w_1 = 0$, $w_2 = -2$, $w_3 = 2$

For (0 – 1 1), $x_1 = 0$, $x_2 = -1$, $x_3 = 1$,

$x_1w_1 + x_2w_2 + x_3w_3 = 0 + 2 + 2 = 4 > 0$, so the response is correct.

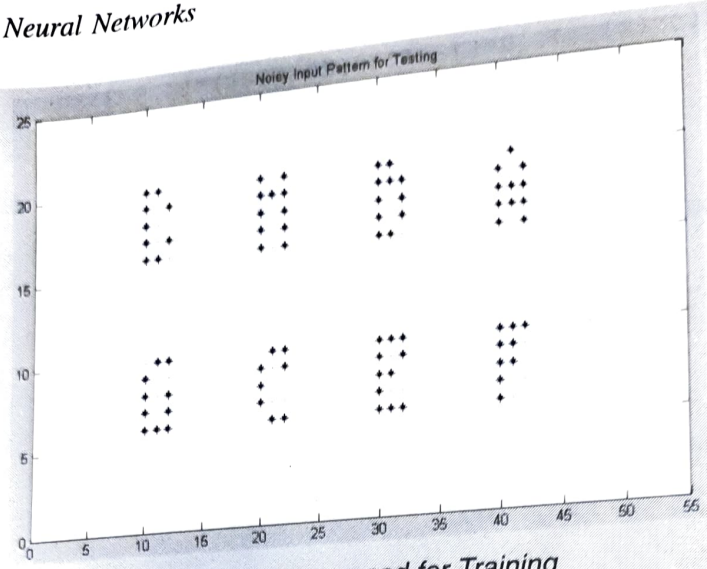
Vector				Response
(x_1	x_2	x_3)	$x_1w_1 + x_2w_2 + x_3w_3$	
(0	– 1	0)	2	Correct
(0	– 1	1)	4	Correct
(0	1	– 1)	– 4	Incorrect
(0	0	1)	2	Correct
(0	0	– 1)	– 2	Incorrect
(0	1	0)	– 2	Incorrect
(1	0	1)	2	Correct
(1	0	– 1)	– 2	Incorrect
(1	– 1	0)	2	Correct
(1	0	0)	– 0	Undetemined
(1	1	0)	– 2	Incorrect
(1	1	1)	0	Undetermined

Example 4.5 Write a MATLAB program for perceptron net for an AND function with bipolar inputs and targets.

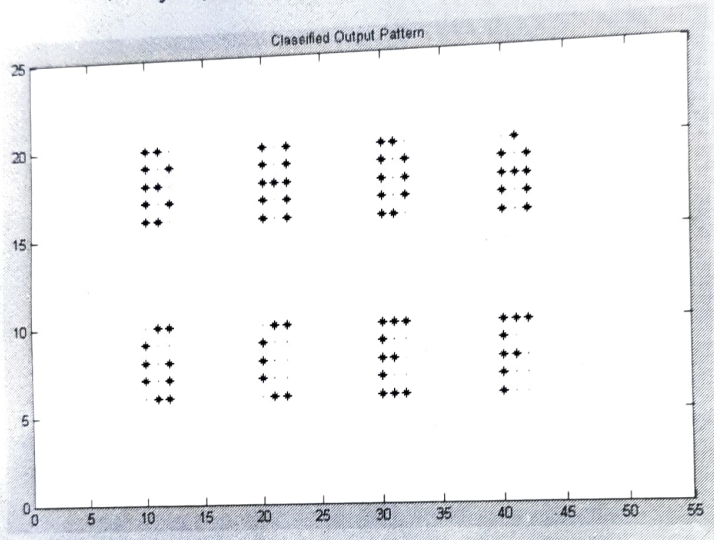
Solution The truth table for the AND function is given as

X_1	X_2	Y
– 1	– 1	– 1
– 1	1	– 1
1	– 1	– 1
1	1	1

The MATLAB program for the above table is given as follows.



Noisy Input Pattern used for Training



Classified Output Pattern

4.3 Brief Introduction to Multilayer Perceptron Networks

Multilayer perceptron networks is an important class of neural networks. The network consists of a set of sensory units that constitute the input layer and one or more hidden layer of computation modes. The input signal passes through the network in the forward direction. The network of this type is called multilayer perceptron (MLP).

The multilayer perceptrons are used with supervised learning and have led to the successful back-propagation algorithm. The disadvantage of the single layer perceptron is that it cannot be extended to multi-layered version. In MLP networks there exists a non-linear activation function. The widely used non-linear activation function is logistic sigmoid function. The MLP network also has various layers of hidden neurons. The hidden neurons make the MLP network active for highly complex tasks. The layers of the network are connected by synaptic weights. The MLP thus has a high computational efficiency.

A disadvantage of MLP may also be the presence of non-linearity and complex connections of the network which leads to highly complex theoretical analysis. Also the existence of hidden neurons makes the learning process tedious.

The MLP networks are usually fully connected networks. There are various multilayer preceptron networks which includes Back propagation network, Radial basis function network etc. These are dealt in detail in the forthcoming chapters.

Summary

The perceptron training algorithm discussed above can be implemented on a digital computer or on any electronic hardware and the network becomes, in a sense self-adjusting. Rosenblatt's proof has been a major milestone and has a great importance in the field of neural network. The single layer perceptron network was discussed with its architecture algorithm, examples etc. An overview of multi-layer perceptrons is also given in this chapter. It should be noted that there is no proof that the perceptron training algorithm is faster than simply trying all possible weight adjustments, but in some cases, the perceptron network provides a superior result.

Review Questions

4.1 What are the three layers in the original perceptron?

4.2 Briefly discuss on the learning rule of a perception network.

4.3 Explain the working of the perceptron net.

4.4 What is the activation function used in the perception network?

4.5 Explain the architecture of the perceptron net used for pattern classification.

4.6 State the perceptron learning rule convergence theorem.

4.7 Explain the algorithm used for training the perceptron net.

4.8 What are the stopping conditions used to stop the progress of the training algorithm?

4.9 How is perceptron net used in the aspect of linear separability?

4.10 Compare perceptron and Hebb net.

4.11 State the application algorithm used in perceptron net.

4.12 Write the perceptron training algorithm for several output classes.

4.13 Give a brief note on multilayer perceptrons.

Exercise Problems

4.14 Form a perceptron net for OR function with binary input and output. Compare it with the results obtained using bipolar input and targets.

4.15 Generate AND NOT function with bipolar data using perceptron learning rule with learning rates.

(i) $\alpha = 1$ (ii) $\alpha = 0.5$. Compare the variations noted

4.16 Using perceptron net, find the weights required to perform the following pattern classifications. Vectors (– 1, 1, – 1, – 1) and (1, 1 , – 1, – 1) are members of class (therefore have target value 1); Vectors (– 1, 1, 1, – 1) and (1, – 1, 1, 1) are not members of class (have target value –1). Using the training vectors as input, test the response of the net using application algorithm.

4.17 Compare the operation of the AND function with bipolar inputs and targets using Hebb net and perceptron net.

4.18 Classify the pattern shown below using perceptron training algorithm. Also test the response

Pattern (a) : L

and

Pattern (b): M

4.19 Write a MATLAB program for perceptron net to generate AND function with binary inputs and targets.

4.20 Write a MATLAB program using perceptron net for OR function with binary input and bipolar target.

4.21 Write a MATLAB program for palindrome recognition function using a perceptron network.

$$h(x_1, \cdots, x_n) = \begin{cases} 1, & \text{if } x_1 = \quad \& \, x_2 - x_{N-1} \\ 0, & \text{otherwise} \end{cases}$$

and $x_1, \ldots \& \, x_{N/2} = x_{N/2+1}$