Perceptron Network

- The perceptron learning rule uses an interactive 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 key point to be noted in a perceptron network are:

1. The perceptron network consists of three units, namely, input unit, associator unit (hidden unit), response unit (output unit).

2. The input units are connected to associator units with fixed weights having values 1, 0 or -1, which are assigned at random.

3. The binary activation function is used in input unit and associator unit.

4. The response unit has an activation of 1, 0 or -1. The binary step with fixed threshold θ is used as activation for associator. The output signals that are sent from the associator unit to the response unit are only binary.

6. The perceptron learning rule is used in the weight updation between the associator unit and the response unit. For each training input, the net will calculate the response and it will determine whether or not an error has occurred.

7. The error calculation is based on the comparison of the values of targets with those of the calculated outputs.

Training Algorithm for Single Output Classes

Step 1: Initialize weights and bias (initially it can be zero). Set learning rate α (0 to 1). For simplicity α is set 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. $x_i=s_i$ for i=1 to n.

Step 5: Compute the output unit response. $Y_{in} = b + \sum_i X_i W_i$

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

Step 6: The weight and bias are updated if the target is not equal to the output response. If $y \neq t$, then

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

$$\text{else} \quad w_{i(new)} = w_{i(old)} \text{ and } b_{(new)} = b_{(old)}$$

Step 7: Train the network until there is no weight change. This is the stopping condition for the network. If this condition is not met, then start again from Step 3.

Training Algorithm for Multiple Output Classes

Step 0: Initialize weights, biases and learning rate suitably.

Step 1: While stopping condition is false do Steps 2-6.

Step 2: Perform Steps 3-5 for each bipolar or binary training pair s:t.

Step 3: Set activations of each input units. $x_i=s_i$ for i=1 to n.

Step 4: Compute the output unit response. $y_{inj} = b_j + \sum_i x_i w_{ij}$

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

Step 5: Make adjustment in weights and bias for j=1 to m and i=1 to n.. If $y \neq t$, then

$$w_{ij(new)} = w_{ij(old)} + \alpha t_j x_i \text{ and } b_{j(new)} = b_{j(old)} + \alpha t_j$$

$$\text{else} \quad w_{ij(new)} = w_{ij(old)} \text{ and } b_{j(new)} = b_{j(old)}$$

Step 6: Test for the stopping condition, i. e., if tehre is no change in weights then stop the training process, else start again from Step 2.