Module II

Perception networks – Learning rule – Training and testing algorithm, AdaptiveLinear Neuron, Back propagation Network – Architecture, Training algorithm.

Perception networks

Theory

Perception networks come under single-layer feed-forward networks and are also called **simple perceptrons.**

The key points to be noted in a perceptron network are:

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

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

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

4. The response unit has an activation of l, 0 or -1. The binary step with fixed threshold $\theta$is used asactivation for associator. The output signals that are sent from the associator unit to the response unit areonly binary.

5. The output of the perceptron network is given by

$$y = f(y_{in})$$

where$f(y_{in})$ is activation function and is defined as

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

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

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

8. The weights on the connections from the units that send the nonzero signal will get adjusted suitably.

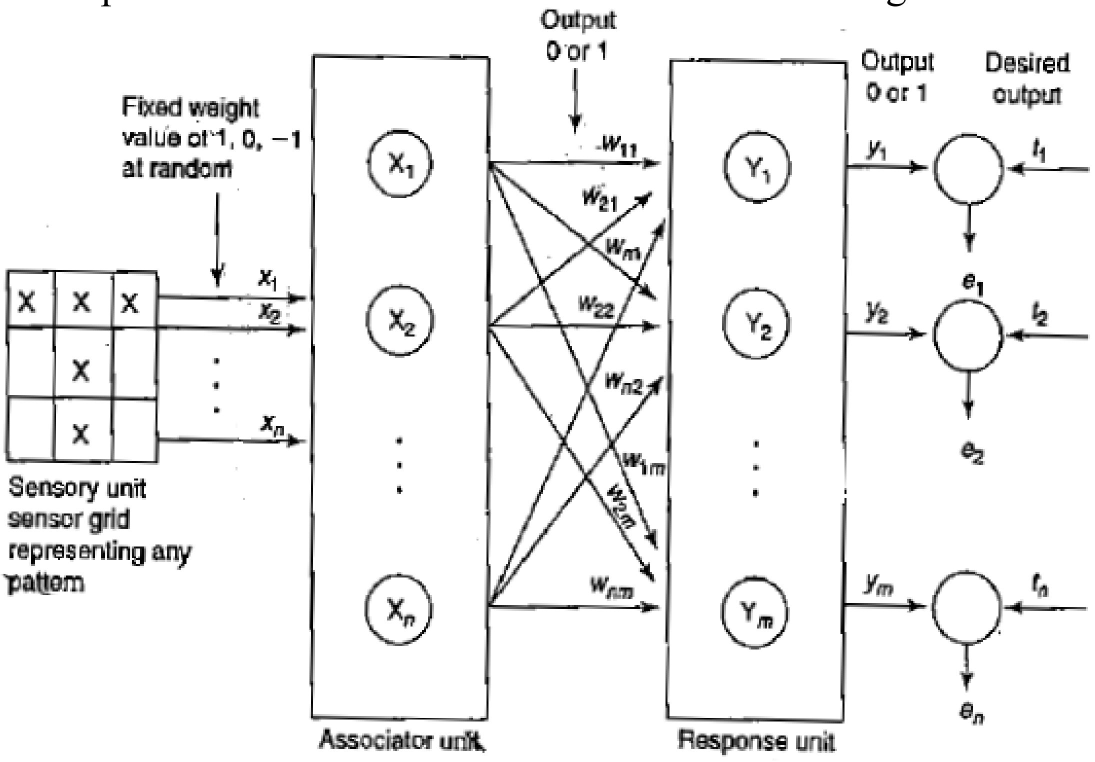
9. The weights will be adjusted on the basis of the learningrule if an error has occurred for a particulartraining pattern, i.e..,

$$w_i(\text{new}) = w_i(\text{old}) + \alpha t x_i$$
$$b(\text{new}) = b(\text{old}) + \alpha t$$

If no error occurs, there is no weight updation and hence the training process may be stopped. In the aboveequations, the target value *"t"* is+ 1 or-1 and *α*is the learningrate.

Original Perception network

A Perceptionnetwork with its three units is shown in Figure below:



Sensory unit

A sensory unit can be a two-dimensional matrix of 400 photo detectors upon which a lighted picture with geometric blackand white pattern impinges. These detectors provide a binary(0) electricalsignal if the input signal is found to exceed acertain value of threshold. Also, these detectors are connected randomly with the associator unit.

Associator unit

The associator unit is found to consist of a set ofsubcircuits called **feature predicates**. The feature predicates arehard-wired to detect the specific feature of a pattern and are equivalent to the *feature detectors.* For a particularfeature, each predicate is examined with a few or all of the responses of the sensory unit. It can be found thatthe results from the predicate units are also binary (0 or 1).

Response unit

The last unit, i.e. response unit, contains thepatternrecognizers or perceptrons. The weights presentin the input layers are all fixed, while the weights onthe response unit are trainable.

Learning rule

In case of the perceptron learning rule, the learning signal is the difference between the desiredandactual response of a neuron. The perceptron learning rule is explained as follows:

Consider a finite *"n"* number of input training vectors, with their associated targetvalues *x(n)*and t{n), where *"n"* ranges from 1 to*N*. The target is either+ 1 or -1. The output *"y"* is obtained on thebasis of the net input calculated and activation function being applied over the net input.

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

The weight updation in case of perceptron learning is as shown.

If y≠ tthen

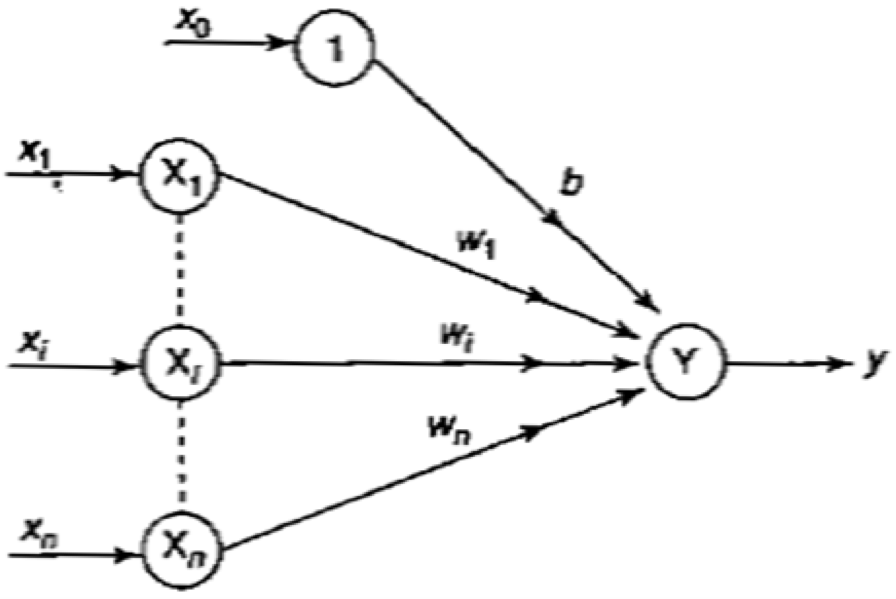
$$w(\text{new}) = w(\text{old}) + \alpha t x \quad (\alpha - \text{learning rate})$$

else

$$w(\text{new}) = w(\text{old})$$

Architecture

A simple perceptron network architecture is shown inFigure below:

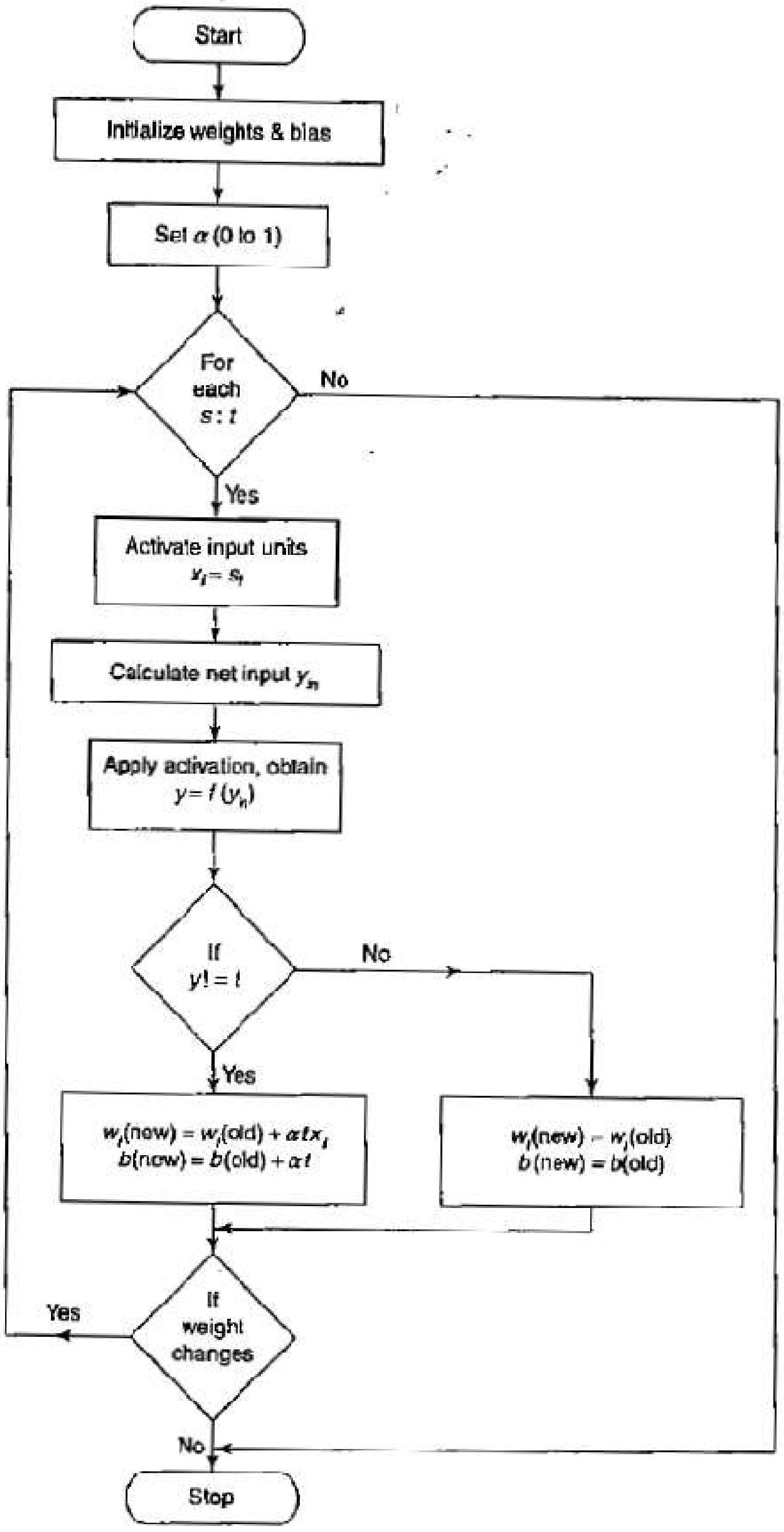


In Figure, there are *n*input neurons, 1 output neuron and a bias. The input-layer and outputlayerneurons are connected through a directed communication link, which is associated with weights.

Thegoal of the perceptron net is to classify the inputpattern *as* a member or not a member to a particularclass.

Flowchart for Training Process

The flowchart for the perceptron network training is shown in Figure.



Start
Initialize weights & bias
Set α (0 to 1)
For each $s:t$
No
Yes
Activate input units $x_i = s_i$
Calculate net input y_{in}
Apply activation, obtain $y = f(y_{in})$
If $y != t$
No
Yes
$w_i(\text{new}) = w_i(\text{old}) + \alpha t x_i$
$b(\text{new}) = b(\text{old}) + \alpha t$
$w_i(\text{new}) = w_i(\text{old})$
$b(\text{new}) = b(\text{old})$
If weight changes
Yes
No
Stop

The flowchart depicted here presents the flow of the training process.As depicted in the flowchart, first the basic initialization required for the training process is performed.

The entire loop of the training process continues until the training input pair is presented to the network. The training (weight updation) is done on the basis of the comparison between the calculated and desired output. The loop is terminated if there is no change in weight.

Training and testing algorithm

Perceptron Training Algorithm for Single Output Classes

The perceptron algorithm can be used for either binary or bipolar input vectors, having bipolar targets,threshold being fixed and variable bias.

In the algorithm below, initiallythe inputs are assigned. Then the net input is calculated. The output of the network is obtained by applying the activation function over the calculated net input.

On performing comparison over the calculated and the desired output, the weight updation process is carried out. The entire network is trained based on thementioned stopping criterion.

The algorithm of a perceptron network is as follows:

Step 0: Initialize the weights and the bias. Also initialize the learning rate $\alpha (O < \alpha \leq 1)$. For simplicity *α is* set to 1.

Step 1: Perform Steps 2-6 until the final stopping condition *is* false.

Step 2: Perform Steps 3-5 for each training pair indicated by *s:t.*

Step 3: The input layer containing input units is applied with identity activation functions:

$$x_i = s_i$$

Step 4: Calculate the output of the network. To do so, first obtain the net input:

$$y_{in} = b + \sum_{i=1}^{n} x_i w_i$$

where"n" is the number of input neurons in the input layer. Then apply activations over the netinput calculated to obtain the output:

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

Step 5: Weight and bias adjustment: Compare the value of the actual (calculated) output and desired(target) output.

$$\begin{aligned} &\text{If } y \neq t, \text{ then} \\ &\qquad w_i(\text{new}) = w_i(\text{old}) + \alpha t x_i \\ &\qquad b(\text{new}) = b(\text{old}) + \alpha t \\ &\text{else, we have} \\ &\qquad w_i(\text{new}) = w_i(\text{old}) \\ &\qquad b(\text{new}) = b(\text{old}) \end{aligned}$$

Step 6: 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 2.

Perceptron Training Algorithm for Multiple Output Classes

For multiple output classes, the perceptron training algorithm is as follows:

Step 0: Initialize the weights, biases and learning rate suitably.
Step 1: Check for stopping condition; if it is false, perform Steps 2-6.
Step 2: Perform Steps 3--5 for each bipolar or binary training vector pair *s:t*.
Step 3: Set activation (identity) of each input unit *i*= 1 ton:

$$x_i=s_i$$

Step 4: Calculate output response of each output unit j=1 to m; First the net input is calculated as:

$$y_{inj} = b_j + \sum_{i=1}^{n} x_i w_{ij}$$

Then activations are applied over the net input to calculate the output response:

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

Step 5: Make adjustment in weights and bias for *j* =1 to *m* and *i*= 1to *n*.

If $t_j \neq y_j$, then

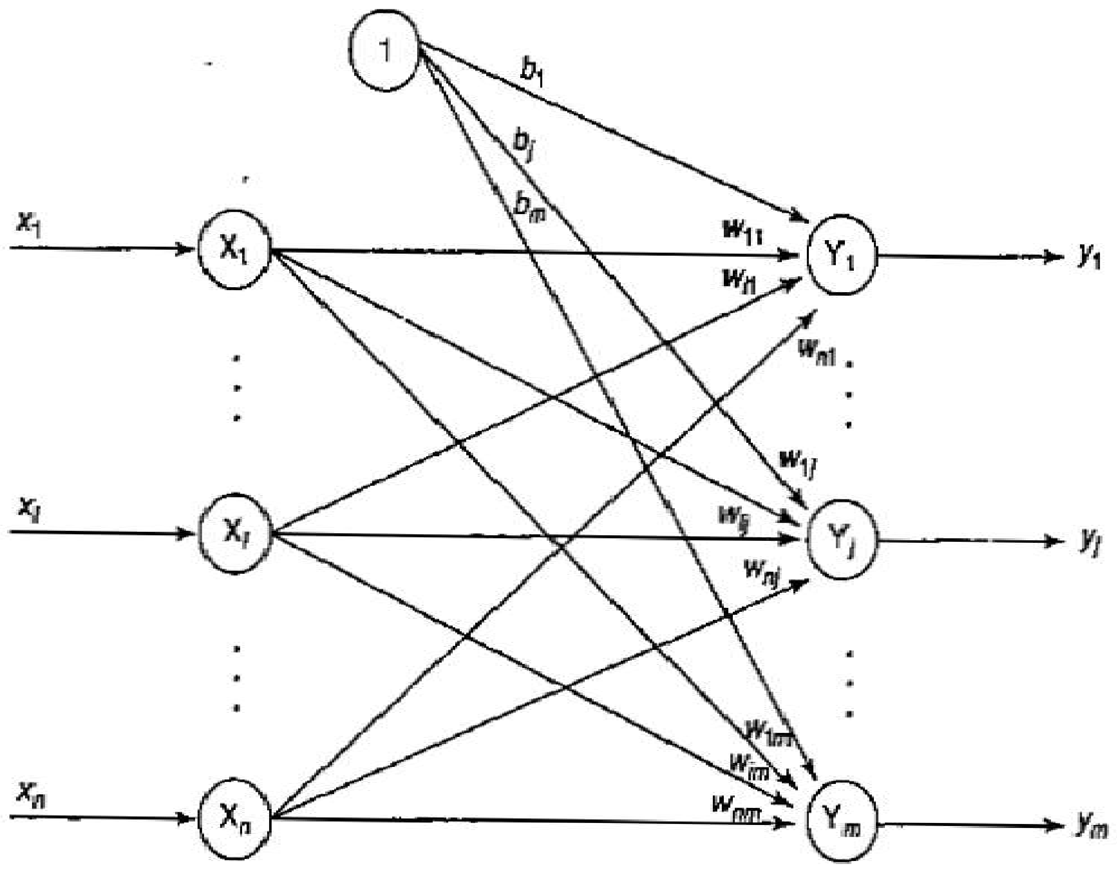
$$w_{ij}(\text{new}) = w_{ij}(\text{old}) + \alpha t_j x_i$$
$$b_j(\text{new}) = b_j(\text{old}) + \alpha t_j$$

else, we have

$$w_{ij}(\text{new}) = w_{ij}(\text{old})$$
$$b_j(\text{new}) = b_j(\text{old})$$

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

The above algorithm issuited for the architecture shown in Figure below.



Perceptron Network Testing Algorithm

The testing algorithm is asfollows:

Step 0: The initialweights to be used here are taken from the training algorithms.

Step 1: For each input vector X to be classified, perform Steps 2-3.

Step 2: Set activations of the input unit.

Step 3: Obtain the response of output unit.

$$y_{in} = \sum_{i=1}^{n} x_i w_i$$

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

Adaptive Linear Neuron (Adaline)

Theory

The units with linear activation function are called linear units. A network with a single linear unit is calledan **Adaline (adaptive linear neuron)**. That is, in an Adaline, the input-output relationship is linear.

Adalineusesbipolar activation for its input signals and its target output. The weights between the input and theoutput are adjustable. The bias in Adaline acts like an adjustable weight, whose connection is from a unitwith activations being always 1. Adaline is a net which has only one output unit. The Adaline network maybe trained using delta rule.

The delta rule may also be called as least mean square (LMS) rule or Widrow-Hoffrule. This learning rule is found to minimize the mean-squared error between the activation and the targetvalue.

Delta Rule for Single Output Unit

The Widrow-Hoff rule is very similar to perceptron learning rule. The delta rule updates the weights between the connections so as tominimize the difference between the net input to the output unit and the target value. The major aim is tominimize the error over all training patterns. This is done by reducing the error for each pattern, one at a time.

The delta rule for adjusting the weight of ith pattern {i = 1 to n*)* is

$$\Delta w_i = \alpha(t - y_{in})x_i$$

where

Δw_i is the weight change

α the learning rate

xthe vector of activation of input unit

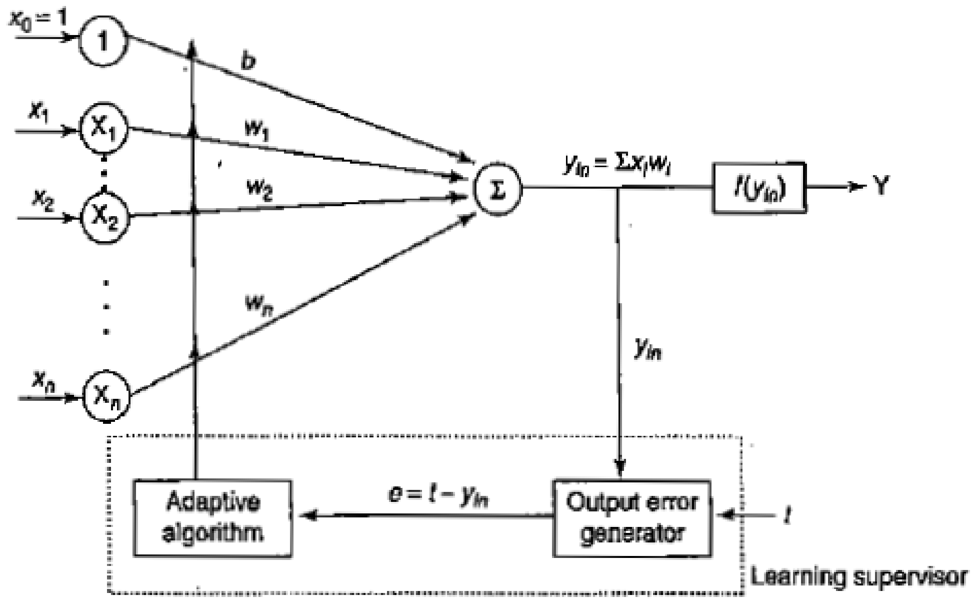
$y_{in}$the net inputto output unit

The delta rule in case of several output units foradjusting the weight from ithinput unit to the jth output unit (for each pattern) is

$$\Delta w_{ij} = \alpha(t_j - y_{inj})x_i$$

Architecture

Adaline is a single-unitneuron, which receives input from several units and also from oneunit called bias. An Adaline model is shown in Figure below:



The basic Adaline model consists of trainableweights.

Inputs are either of the two values (+ 1 or -1) and the weights have signs (positive or negative).

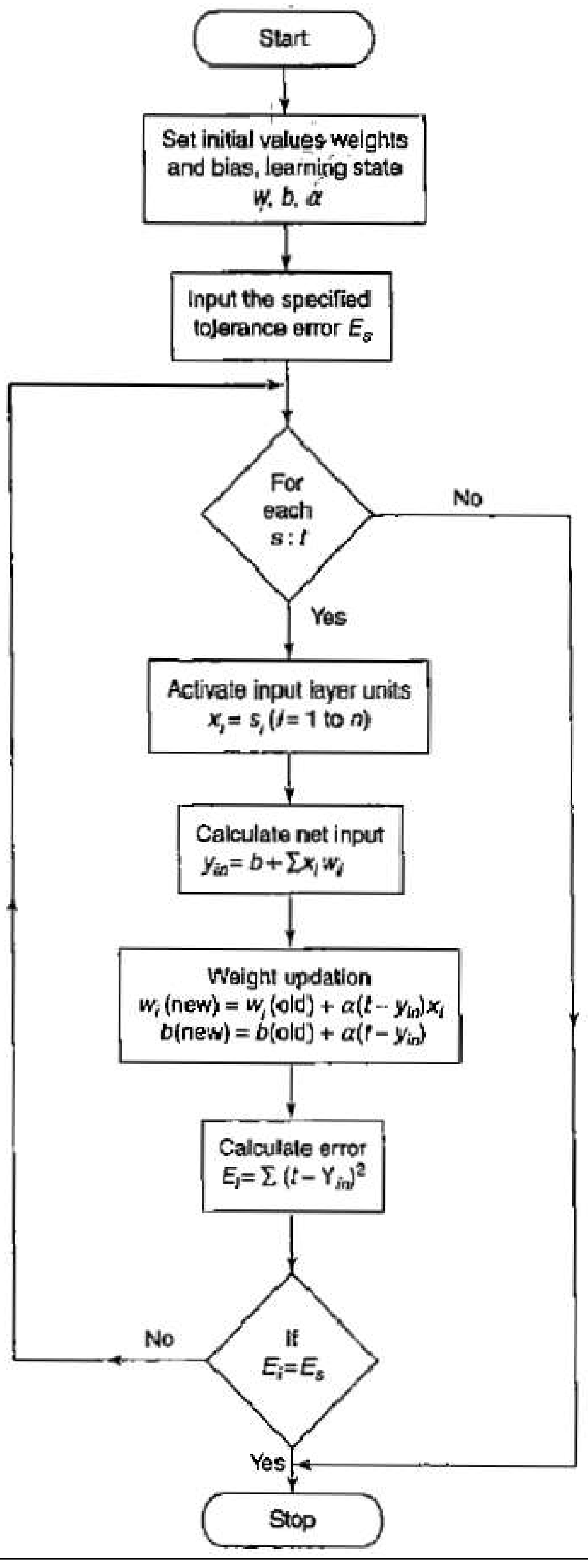
Initially, random weights are assigned.

The net input calculated is applied to a quantizer transfer functionthat restores the output to +1 or -1. The Adaline model compares the actualoutput with the target output and on the basis of the training algorithm, the weights are adjusted.

Flowchart for Training Process

The flowchart for the training process is shown in Figure below:This gives a pictorial representation of thenetwork training.

The conditions necessary for weight adjustments have to be checked carefully. The weightsand other required parameters are initialized. Then the net input is calculated, output is obtained and comparedwith the desired output for calculation of error. On the basis of the error Factor, weights are adjusted.


Start
Set initial values weights and bias, learning state w, b, α
Input the specified tolerance error E_s
For each $s : t$
No
Yes
Activate input layer units $x_i = s_i$ (i = 1 to n)
Calculate net input $y_{in} = b + \sum x_i w_i$
Weight updation
w_i(new) = w_i(old) + $\alpha(t - y_{in})x_i$
b(new) = b(old) + $\alpha(t - y_{in})$
Calculate error $E_i = \sum (t - Y_{in})^2$
If $E_i = E_s$
No
Yes
Stop

Training Algorithm

The Adaline network training algorithm is as follows:

Step 0: Weights and bias are set to some random values but not zero. Set the learning rate parameter α.
Step 1: Perform Steps 2-6 when stopping condition is false.
Step 2: Perform Steps 3-5 for each bipolar training pair *s: t.*
Step 3: Set activations for input units *i*= 1 to *n.*

$$x_i = s_i$$

Step 4: Calculate the net input to the output unit.

$$y_{in} = b + \sum_{i=1}^{n} x_i w_i$$

Step 5: Update the weights and bias fori= 1ton:

$$w_i(\text{new}) = w_i(\text{old}) + \alpha\,(t - y_{in})\,x_i$$
$$b(\text{new}) = b\,(\text{old}) + \alpha\,(t - y_{in})$$

Step 6: If the highest weight change that occurred during training is smaller than a specified tolerance then stop the training process, else continue. This is the rest for stopping condition of anetwork.

The range of learning rate can be between 0.1 and 1.0.

Testing Algorithm

It is essential to perform the resting of a network that has been trained. When training is completed, theAdaline can be used to classify input patterns.

A step function is used to test the performance of the network.The resting procedure for the Adaline network is as follows:

Step 0: Initialize the weights.
Step 1: Perform Steps 2-4 for each bipolar input vector *x.*
Step 2: Set the activations of the input units to *x.*
Step 3: Calculate the net input to the output unit:

$$y_{in} = b + \sum x_i w_i$$

Step 4: Apply the activation function over the net input calculated:

$$y = \begin{cases} 1 & \text{if } y_{in} \geq 0 \\ -1 & \text{if } y_{in} < 0 \end{cases}$$

Back propagation Network

Theory

The backpropagation learning algorithm is one of the most important developments in neural networks. The networks associatedwith back-propagation learning algorithm are called back propagation networks(BPNs).

For a given setof training input-output pair, this algorithm provides a procedure for changing the weights in a BPN toclassify the given input patterns correctly.

The basic concept for this weight update algorithm is simply thegradient-descent method. This is amethod where the error is propagated back to the hidden unit.

The aim of the neural networkistotrain thenet to achieve a balance between the net's ability to respond and its ability to give reasonableresponses to the inputthat is similar but not identical to the one that is used in training.

The back-propagation algorithm is different from other networks in respect to the process by which weights are calculated during the learning period of the network.

The general difficulty with the multilayerperceptrons is calculating the weights of the hidden layers in an efficient way that would result in a very smallor zero output error.

When the hidden layers are increased the network training becomes more complex. Toupdate weights, the error must be calculated. The error, which is the difference between the actual (calculated)and the desired (target) output, is easily measured at the output layer.

The training of the BPN is done in three stages:

- the feed-forward of the input training pattern
- thecalculation and back-propagation of the error
- updation of weights.

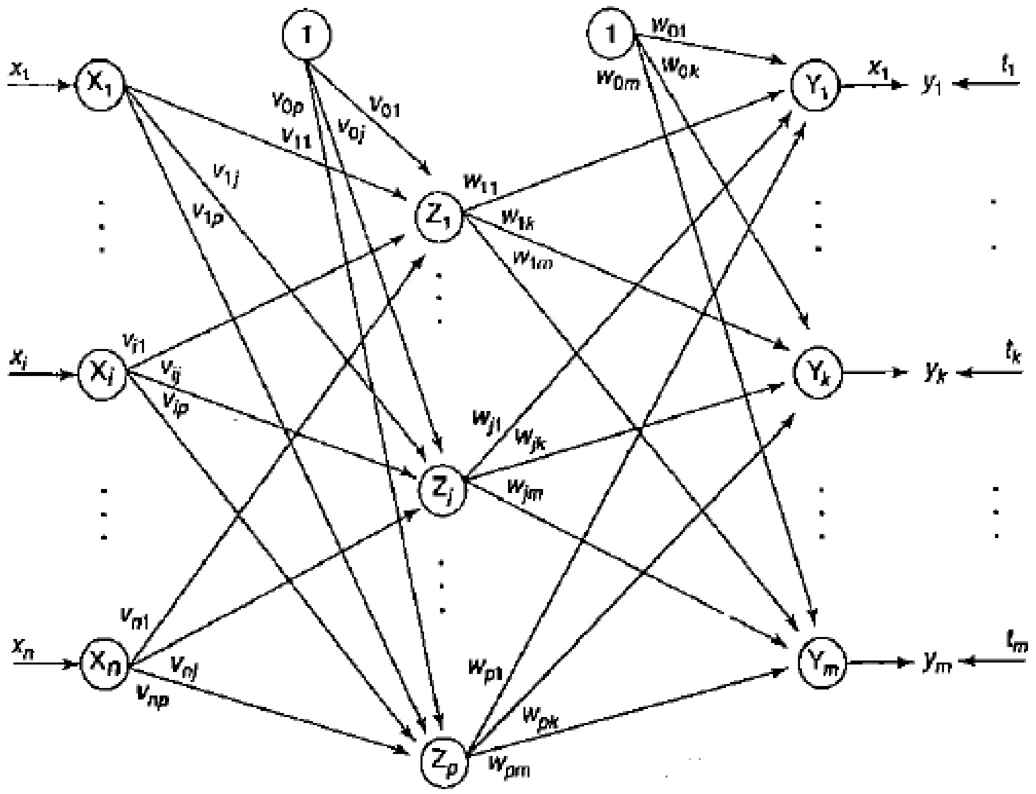
Architecture

A back-propagation neural network is a multilayer, feed forward neural network consisting of an input layer,a hidden layer and an output layer.

The neurons present in the hidden and output layers have biases, whichare the connections from the units whose activation is always 1. The bias terms also acts as weights.

The Figure below shows the architecture of a BPN, depicting only the direction of information flow for the feed-forward phase.During the back propagation phase of learning signals are sent in the reverse direction.

The inputs sent to the BPN and the output obtained from the net could be either binary (0, 1) orbipolar (-1, + 1). The activation function could be any function which increases monotonically and is alsodifferentiable.



Training algorithm

Flowchart for Training Process

The flowchart for the training process using a BPN is shown in Figure below. The terminologies used in theflowchart and in the training algorithm are as follows:

x = input training vector $(x_1, \ldots, x_i, \ldots, x_n)$
t = target output vector $(t_1, \ldots, t_k, \ldots, t_m)$
α = learning rate parameter
x_i = input unit i. (Since the input layer uses identity activation function, the input and output signals here are same.)
v_{0j} = bias on jth hidden unit
w_{0k} = bias on kth output unit
z_j = hidden unit j. The net input to z_j is

$$z_{inj} = v_{0j} + \sum_{i=1}^{n} x_i v_{ij}$$

andthe output is

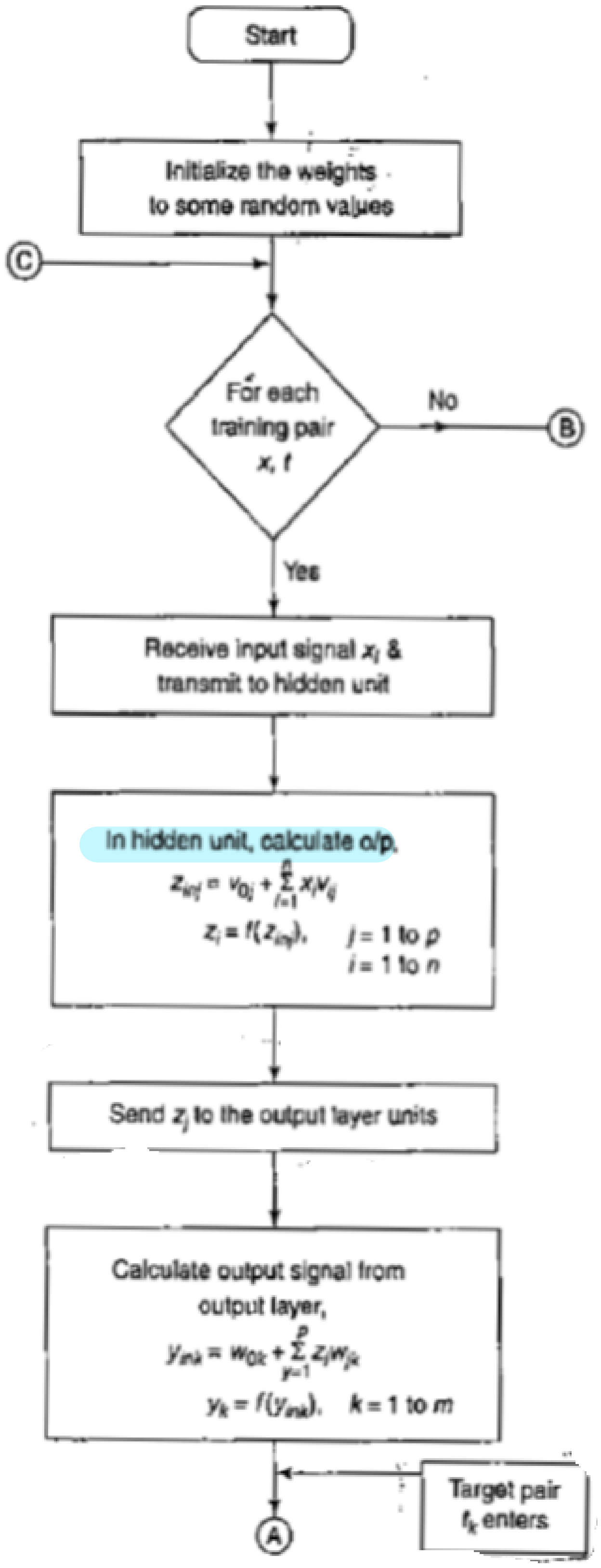
$$z_j = f(z_{inj})$$

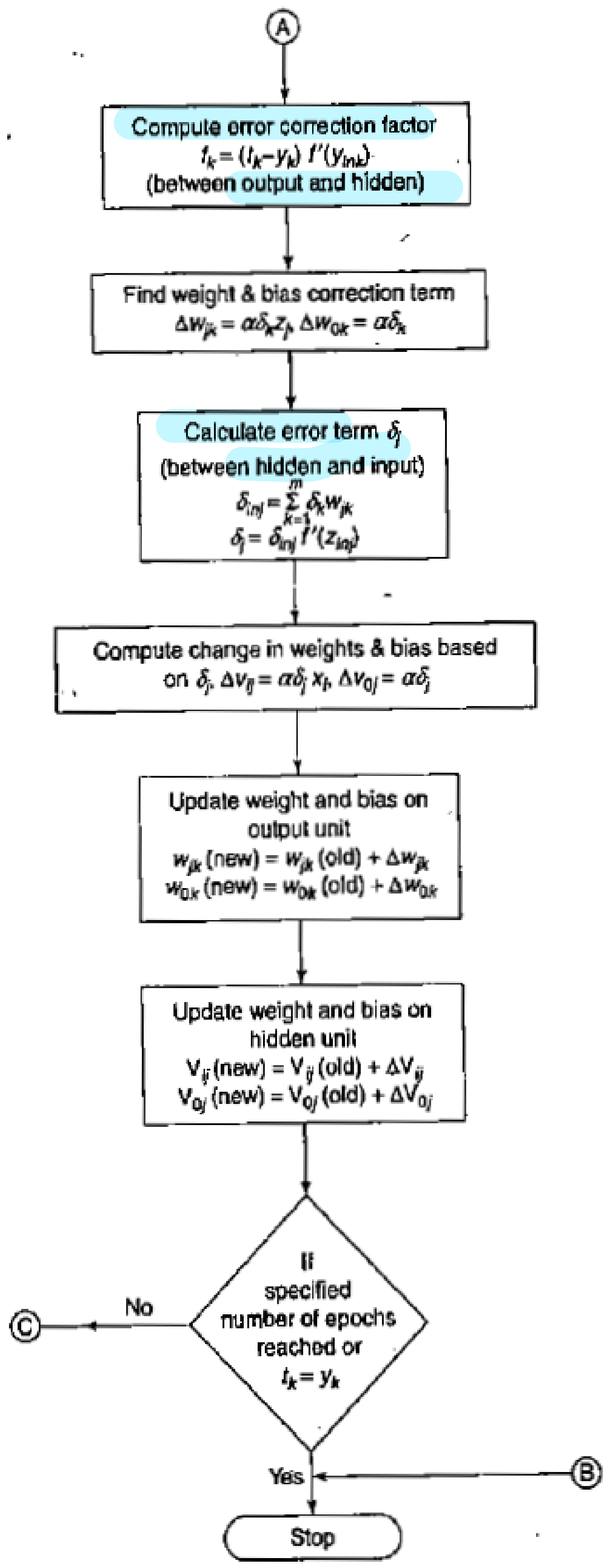
y_k= output unit k. The net input to$y_k$is

$$y_{ink} = w_{0k} + \sum_{j=1}^{p} z_j w_{jk}$$

andthe output is

$$y_k = f(y_{ink})$$


Start
Initialize the weights to some random values
C
For each training pair x, t
No
B
Yes
Receive input signal x_i & transmit to hidden unit
In hidden unit, calculate o/p,
$z_{inj} = v_{0j} + \sum_{i=1}^{n} x_i v_{ij}$
$z_j = f(z_{inj}), \quad j = 1 \text{ to } p$
$i = 1 \text{ to } n$
Send z_j to the output layer units
Calculate output signal from output layer,
$y_{ink} = w_{0k} + \sum_{j=1}^{p} z_j w_{jk}$
$y_k = f(y_{ink}), \quad k = 1 \text{ to } m$
Target pair t_k enters
A


A
Compute error correction factor
$\delta_k = (t_k - y_k)\, f'(y_{ink})$
(between output and hidden)
Find weight & bias correction term
$\Delta w_{jk} = \alpha\delta_k z_j,\ \Delta w_{0k} = \alpha\delta_k$
Calculate error term δ_j
(between hidden and input)
$\delta_{inj} = \sum_{k=1}^{m} \delta_k w_{jk}$
$\delta_j = \delta_{inj}\, f'(z_{inj})$
Compute change in weights & bias based on δ_j, $\Delta v_{ij} = \alpha\delta_j x_i$, $\Delta v_{0j} = \alpha\delta_j$
Update weight and bias on output unit
w_{jk} (new) = w_{jk} (old) + Δw_{jk}
w_{0k} (new) = w_{0k} (old) + Δw_{0k}
Update weight and bias on hidden unit
V_{ij} (new) = V_{ij} (old) + ΔV_{ij}
V_{0j} (new) = V_{0j} (old) + ΔV_{0j}
If specified number of epochs reached or $t_k = y_k$
No
C
Yes
B
Stop

Training Algorithm

Step 0: Initialize weights and learning rate.
Step 1: Perform Steps 2-9 when stopping condition is false.
Step 2: Perform Steps 3-8 for training pair.

Feed-Forward Phase(Phase: I)

Step 3: Each input unit receives input signal *x;* and sends it to the hidden unit (i = l to n).

Step 4: Each hidden unit z_j*(j* = 1 top) sums its Weighted input signals to calculate net input:

$$z_{inj} = v_{0j} + \sum_{i=1}^{n} x_i v_{ij}$$

Calculate output of the hidden unit by applying its activation functions over z_{inj}

$$z_j = f(z_{inj})$$

and send the output signal from the hidden unit to the input of output layer units.

Step 5: For each output unit y_k*(k* = 1 to m),calculate the net input:

$$y_{ink} = w_{0k} + \sum_{j=1}^{p} z_j w_{jk}$$

and apply the activation function to compute output signal

$$y_k = f(y_{ink})$$

Back propagation of error (Phase ll)

Step 6: Each output unity$_k$*(k* 1 to m) receives a target pattern corresponding to the input trainingpattern and computes theerrorcorrection term:

$$\delta_k = (t_k - y_k) f'(y_{ink})$$

On the basis of the calculated errorcorrection term, update the change in weights and bias:

$$\Delta w_{jk} = \alpha \delta_k z_j; \quad \Delta w_{0k} = \alpha \delta_k$$

Step 7: Each hidden unit (z_j,j= 1 top) sums its delta inputs from the output units:

$$\delta_{inj} = \sum_{k=1}^{m} \delta_k w_{jk}$$

On the basis of the calculated δ*j*, update the changein weights and bias:

$$\Delta v_{ij} = \alpha \delta_j x_i; \quad \Delta v_{0j} = \alpha \delta_j$$

Weight and bias updation (PhaseIII):

Step 8: Each output unit (y_k, k = 1 tom) updates the bias and weights:

$$w_{jk}(\text{new}) = w_{jk}(\text{old}) + \Delta w_{jk}$$
$$w_{0k}(\text{new}) = w_{0k}(\text{old}) + \Delta w_{0k}$$

Each hidden unit (z_j,j= 1 top) updates its bias and weights:

$$v_{ij}(\text{new}) = v_{ij}(\text{old}) + \Delta v_{ij}$$
$$v_{0j}(\text{new}) = v_{0j}(\text{old}) + \Delta v_{0j}$$

Step 9: Check for the stopping condition. The stopping condition may be certain number of epochsreached or when the actual output equals the target output.

The above algorithm uses the incremental approach for updation of weights, i.e., the weights are beingchanged immediately after a training pattern is presented. There is another way of training called **batch-mode training**, where the weights are changed only after all the training patterns are presented. The effectiveness of two approaches depends on the problem, but batch-mode training requires additional local storage for eachconnection to maintain the immediate weight changes. When a BPN is used as a classifier, it is equivalent tothe optimal Bayesian discriminant function for asymptotically large sets of statistically independent trainingpatterns.