Course Name: Soft Computing
Course Code: CS 125

Artificial Neural Network (ANN)

Delta Learning Rule Widrow–Hoff Rule

It is introduced by Bernard Widrow and Marcian Hoff, also called Least Mean Square LMS method, to minimize the error over all training patterns. It is kind of supervised learning algorithm with having continuous activation function.

Basic Concept − The base of this rule is gradient-descent approach, which continues forever. Delta rule updates the synaptic weights so as to minimize the net input to the output unit and the target value.

The weights represent information being used by the net to solve problem. The delta rule changes the weights of the neural connections so as to minimize the difference between the net input to the output unit and the target value.

Mathematical Formulation – To update the synaptic weights, delta rule is given by

$$\Delta w_i = \alpha . x_i . e_j$$

Here Δw_i = weight change for i^{th} pattern;

α = the positive and constant learning rate;

x_i = the input value from pre-synaptic neuron;

$e_j = (t - y_{in})$, the difference between the desired/target output and the actual output y_{in}

The above delta rule is for a single output unit only.

The updating of weight can be done in the following two cases –

Case-I – when **t ≠ y**, then

$$w(new) = w(old) + \Delta w$$

Case-II – when **t = y**, then

No change in weight

3.3 Adaptive Linear Neuron (Adaline)

3.3.1 Theory

The units with linear activation function are called linear units. A network with a single linear unit is called an *Adaline* (adaptive linear neuron). That is, in an Adaline, the input–output relationship is linear. Adaline uses bipolar activation for its input signals and its target output. The weights between the input and the output are adjustable. The bias in Adaline acts like an adjustable weight, whose connection is from a unit with activations being always 1. Adaline is a net which has only one output unit. The Adaline network may be trained using delta rule. The delta rule may also be called as *least mean square* (LMS) rule or Widrow-Hoff rule. This learning rule is found to minimize the mean-squared error between the activation and the target value.

3.3.2 Delta Rule for Single Output Unit

The Widrow-Hoff rule is very similar to perceptron learning rule. However, their origins are different. The perceptron learning rule originates from the Hebbian assumption while the delta rule is derived from the gradient-descent method (it can be generalized to more than one layer). Also, the perceptron learning rule stops after a finite number of learning steps, but the gradient-descent approach continues forever, converging only asymptotically to the solution. The delta rule updates the weights between the connections so as to minimize the difference between the net input to the output unit and the target value. The major aim is to minimize 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; x the vector of activation of input unit; y_{in} the net input to output unit, i.e., $Y = \sum_{i=1}^{n} x_i w_i$; t the target output. The delta rule in case of several output units for adjusting the weight from ith input unit to the jth output unit (for each pattern) is

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

3.3.5 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.

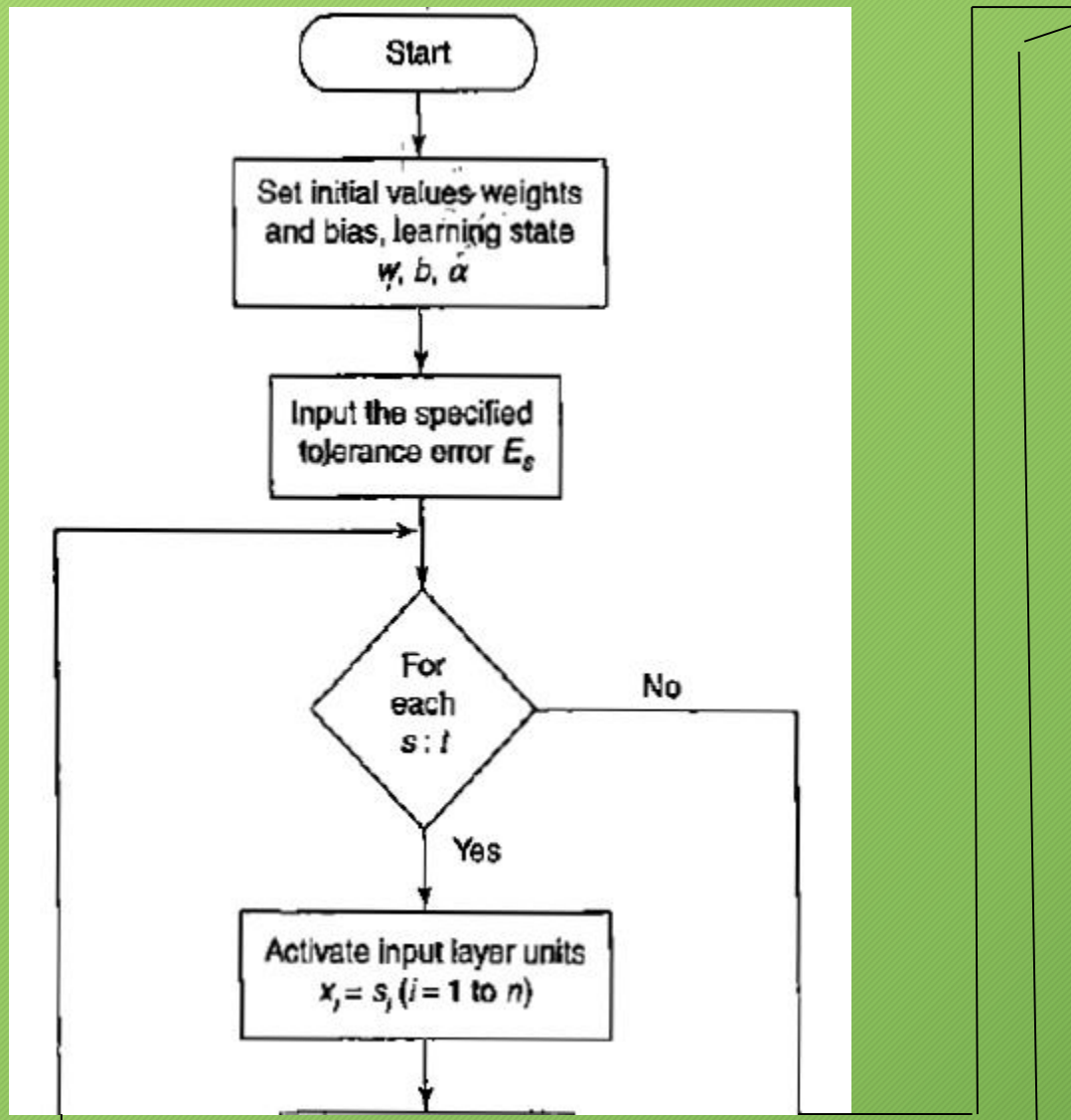
$$y_{in} = b + \sum_{i=1}^{n} x_i w_i$$

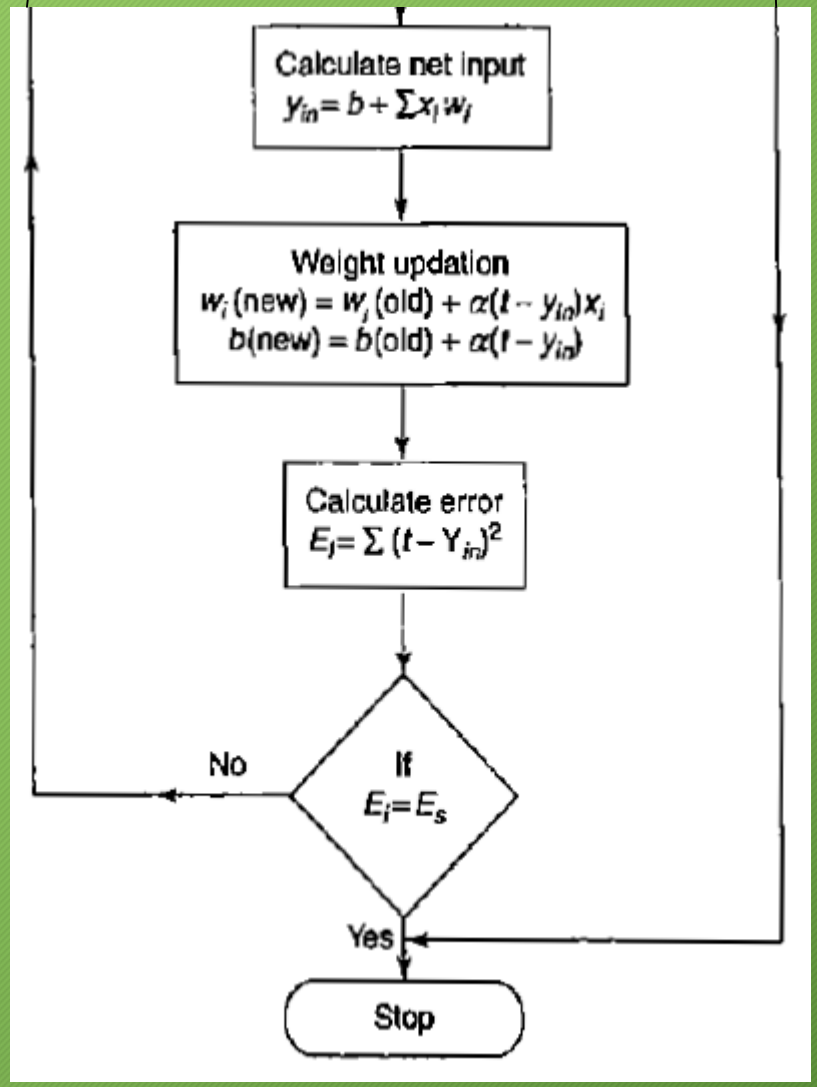
Step 5: Update the weights and bias for $i = 1$ to n:

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

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 test for stopping condition of a network.

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





Refer Page 59 of Book

3.3.6 Testing Algorithm

It is essential to perform the testing of a network that has been trained. When training is completed, the Adaline can be used to classify input patterns. A step function is used to test the performance of the network. The testing procedure for the Adaline network is as follows:

Step 0: Initialize the weights. (The weights are obtained from the training algorithm.)

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:

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

6. Implement OR function with bipolar inputs and targets using Adaline network.

Solution: The truth table for OR function with bipolar inputs and targets is shown in Table 10.

Table 10

x_1	x_2	1	t
1	1	1	1
1	−1	1	1
−1	1	1	1
−1	−1	1	−1

Initially all the weights and links are assumed to be small random values, say 0.1, and the learning rate is also set to 0.1. Also here the least mean square error may be set. The weights are calculated until the least mean square error is obtained.

The initial weights are taken to be $w_1 = w_2 = b = 0.1$ and the learning rate $\alpha = 0.1$. For the first input sample, $x_1 = 1, x_2 = 1, t = 1$, we calculate the net input as

$$y_{in} = b + \sum_{i=1}^{n} x_i w_i = b + \sum_{i=1}^{2} x_i w_i$$
$$= b + x_1 w_1 + x_2 w_2$$
$$= 0.1 + 1 \times 0.1 + 1 \times 0.1 = 0.3$$

Now compute $(t - y_{in}) = (1 - 0.3) = 0.7$. Updating the weights we obtain,

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

where $\alpha(t - y_{in})x_i$ is called as weight change Δw_i. The new weights are obtained as

$$w_1(\text{new}) = w_1(\text{old}) + \Delta w_1 = 0.1 + 0.1 \times 0.7 \times 1$$
$$= 0.1 + 0.07 = 0.17$$
$$w_2(\text{new}) = w_2(\text{old}) + \Delta w_2 = 0.1$$
$$+ 0.1 \times 0.7 \times 1 = 0.17$$

$$b(\text{new}) = b(\text{old}) + \Delta b = 0.1 + 0.1 \times 0.7 = 0.17$$

where

$$\Delta w_1 = \alpha(t - y_{in})x_1$$
$$\Delta w_2 = \alpha(t - y_{in})x_2$$
$$\Delta b = \alpha(t - y_{in})$$

Now we calculate the error:

$$E = (t - y_{in})^2 = (0.7)^2 = 0.49$$

The final weights after presenting first input sample are

$$w = [0.17 \quad 0.17 \quad 0.17]$$

and error $E = 0.49$.

These calculations are performed for all the input samples and the error is calculated. One epoch is completed when all the input patterns are presented. Summing up all the errors obtained for each input sample during one epoch will give the total mean square error of that epoch. The network training is continued until this error is minimized to a very small value.

Adopting the method above, the network training is done for OR function using Adaline network and is tabulated below in Table 11 for $\alpha = 0.1$.

The total mean square error after each epoch is given as in Table 12.

Thus from Table 12, it can be noticed that as training goes on, the error value gets minimized. Hence, further training can be continued for further minimization of error. The network architecture of Adaline network for OR function is shown in Figure 8.

Table 12

Epoch	Total mean square error
Epoch 1	3.02
Epoch 2	1.938
Epoch 3	1.5506
Epoch 4	1.417
Epoch 5	1.377

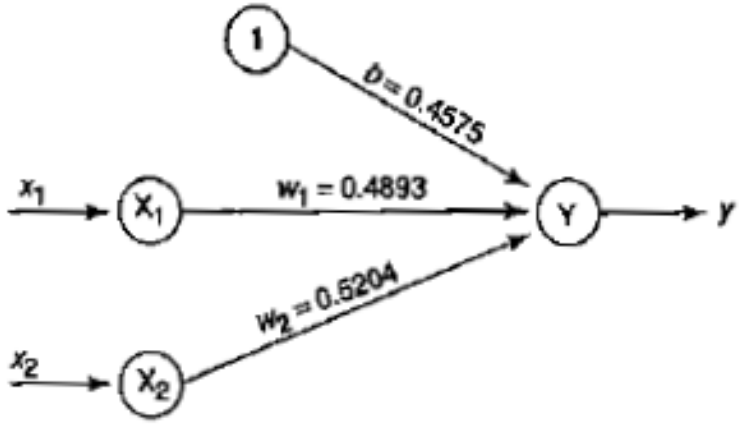


Figure 8 Network architecture of Adaline.

Table 11

Inputs			Target	Net input		Weight changes			Weights			Error
x_1	x_2	1	t	y_{in}	$(t - y_{in})$	Δw_1	Δw_2	Δb	w_1 (0.1	w_2 0.1	b 0.1)	$(t - y_{in})^2$
EPOCH-1												
1	1	1	1	0.3	0.7	0.07	0.07	0.07	0.17	0.17	0.17	0.49
1	−1	1	1	0.17	0.83	0.083	−0.083	0.083	0.253	0.087	0.253	0.69
−1	1	1	1	0.087	0.913	−0.0913	0.0913	0.0913	0.1617	0.1783	0.3443	0.83
−1	−1	1	−1	0.0043	−1.0043	0.1004	0.1004	−0.1004	0.2621	0.2787	0.2439	1.01
EPOCH-2												
1	1	1	1	0.7847	0.2153	0.0215	0.0215	0.0215	0.2837	0.3003	0.2654	0.046
1	−1	1	1	0.2488	0.7512	0.7512	−0.0751	0.0751	0.3588	0.2251	0.3405	0.564
−1	1	1	1	0.2069	0.7931	−0.7931	0.0793	0.0793	0.2795	0.3044	0.4198	0.629
−1	−1	1	−1	−0.1641	−0.8359	0.0836	0.0836	−0.0836	0.3631	0.388	0.336	0.699
EPOCH-3												
1	1	1	1	1.0873	−0.0873	−0.087	−0.087	−0.087	0.3543	0.3793	0.3275	0.0076
1	−1	1	1	0.3025	+0.6975	0.0697	−0.0697	0.0697	0.4241	0.3096	0.3973	0.487
−1	1	1	1	0.2827	0.7173	−0.0717	0.0717	0.0717	0.3523	0.3813	0.469	0.515
−1	−1	1	−1	−0.2647	−0.7353	0.0735	0.0735	−0.0735	0.4259	0.4548	0.3954	0.541
EPOCH-4												
1	1	1	1	1.2761	−0.2761	−0.0276	−0.0276	−0.0276	0.3983	0.4272	0.3678	0.076
1	−1	1	1	0.3389	0.6611	0.0661	−0.0661	0.0661	0.4644	0.3611	0.4339	0.437
−1	1	1	1	0.3307	0.6693	−0.0669	0.0669	0.0699	0.3974	0.428	0.5009	0.448
−1	−1	1	−1	−0.3246	−0.6754	0.0675	0.0675	−0.0675	0.465	0.4956	0.4333	0.456
EPOCH-5												
1	1	1	1	1.3939	−0.3939	−0.0394	−0.0394	−0.0394	0.4256	0.4562	0.393	0.155
1	−1	1	1	0.3634	0.6366	0.0637	−0.0637	0.0637	0.4893	0.3925	0.457	0.405
−1	1	1	1	0.3609	0.6391	−0.0639	0.0639	0.0639	0.4253	0.4654	0.5215	0.408
−1	−1	1	−1	−0.3603	−0.6397	0.064	0.064	−0.064	0.4893	0.5204	0.4575	0.409