

Course Name: Soft Computing
Course Code: CS 125

Artificial Neural Network (ANN)

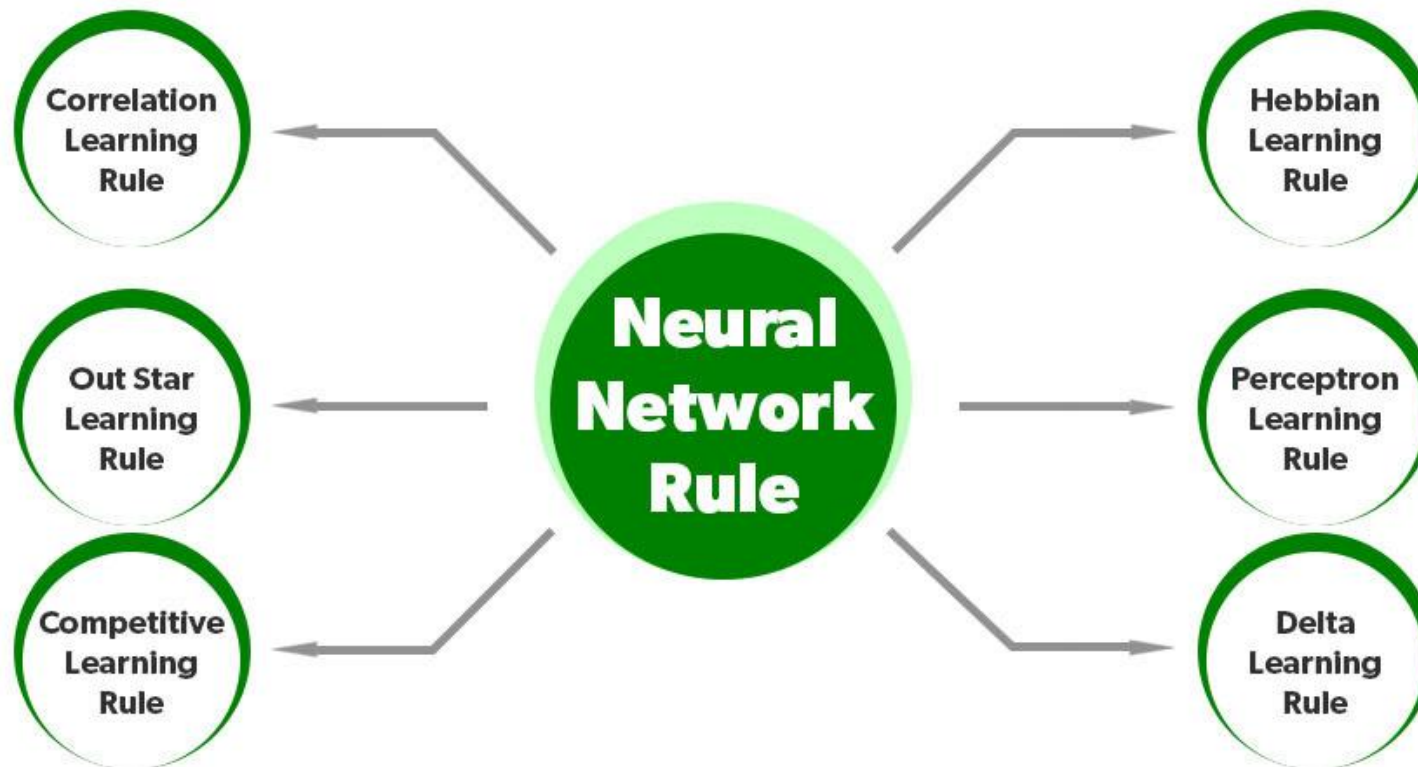
Learning Rules in ANN

2

- Learning rule enhances the Artificial Neural Network's (ANN) performance by applying this rule over the network.
- Thus learning rule updates the weights and bias levels of a network when certain conditions are met in the training process.
- It is a crucial part of the development of the Neural Network.

Types Of Learning Rules in ANN

3



Hebbian Learning Rule

4

- Donald Hebb developed it in 1949 as an unsupervised learning algorithm in the neural network. We can use it to improve the weights of nodes of a network. The following phenomenon occurs when
 - If two neighbor neurons are operating in the same phase at the same period of time, then the weight between these neurons should increase.
 - For neurons operating in the opposite phase, the weight between them should decrease.
 - If there is no signal correlation, the weight does not change, the sign of the weight between two nodes depends on the sign of the input between those nodes
 - When inputs of both the nodes are either positive or negative, it results in a strong positive weight.
 - If the input of one node is positive and negative for the other, a strong negative weight is present.

Hebbian Learning Rule Algorithm

5

- Set all weights to zero, $w_i = 0$ for $i=1$ to n , and bias to zero.
- For each input vector, $S(\text{input vector}) : t(\text{target output pair})$, repeat steps 3-5.
- Set activations for input units with the input vector $X_i = S_i$ for $i = 1$ to n .
- Set the corresponding output value to the output neuron, i.e. $y = t$.
- Update weight and bias by applying Hebb rule for all $i = 1$ to n :

$$w_i (\text{new}) = w_i (\text{old}) + x_i y$$

$$b (\text{new}) = b (\text{old}) + y$$

Hebbian Learning Rule

6

- Implementing AND Gate :

INPUT				TARGET	
	x_1	x_2	b		y
X_1	-1	-1	1	Y_1	-1
X_2	-1	1	1	Y_2	-1
X_3	1	-1	1	Y_3	-1
X_4	1	1	1	Y_4	1

Hebbian Learning Rule

7

There are 4 training samples, so there will be 4 iterations. Also, the activation function used here is Bipolar Sigmoidal Function so the range is $[-1,1]$.

- **Step 1 :**

Set weight and bias to zero, $w = [0\ 0\ 0]^T$ and $b = 0$.

- **Step 2 :**

Set input vector $X_i = S_i$ for $i = 1$ to 4.

$$X_1 = [-1\ -1\ 1]^T$$

$$X_2 = [-1\ 1\ 1]^T$$

$$X_3 = [1\ -1\ 1]^T$$

$$X_4 = [1\ 1\ 1]^T$$

- **Step 3 :**

Output value is set to $y = t$.

Hebbian Learning Rule

8

- **Step 4 :**

Modifying weights using Hebbian Rule:

- First iteration –

$$w(\text{new}) = w(\text{old}) + x_1 y_1 = [0 \ 0 \ 0]^T + [-1 \ -1 \ 1]^T \cdot [-1] = [1 \ 1 \ -1]^T$$

For the second iteration, the final weight of the first one will be used and so on.

- Second iteration –

$$w(\text{new}) = [1 \ 1 \ -1]^T + [-1 \ 1 \ 1]^T \cdot [-1] = [2 \ 0 \ -2]^T$$

- Third iteration –

$$w(\text{new}) = [2 \ 0 \ -2]^T + [1 \ -1 \ 1]^T \cdot [-1] = [1 \ 1 \ -3]^T$$

- Fourth iteration –

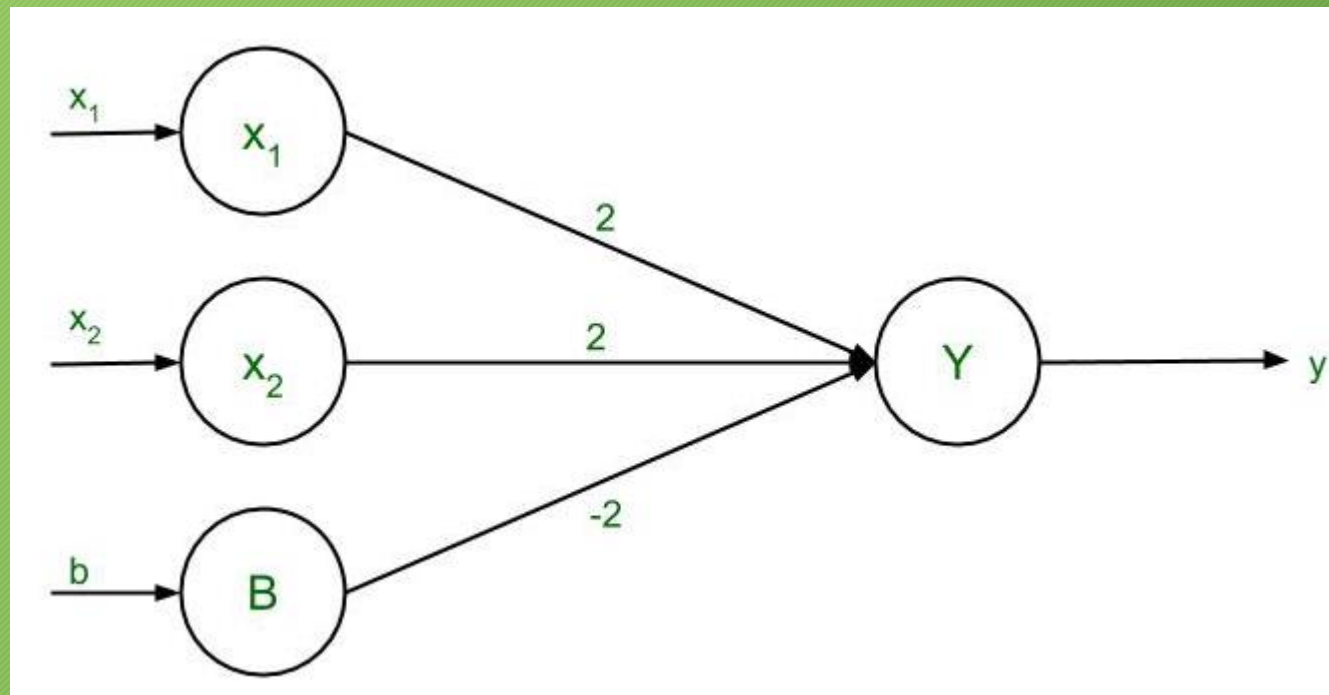
$$w(\text{new}) = [1 \ 1 \ -3]^T + [1 \ 1 \ 1]^T \cdot [1] = [2 \ 2 \ -2]^T$$

So, the final weight matrix is $[2 \ 2 \ -2]^T$

Hebbian Learning Rule

9

- Testing the network :



Hebbian Learning Rule

10

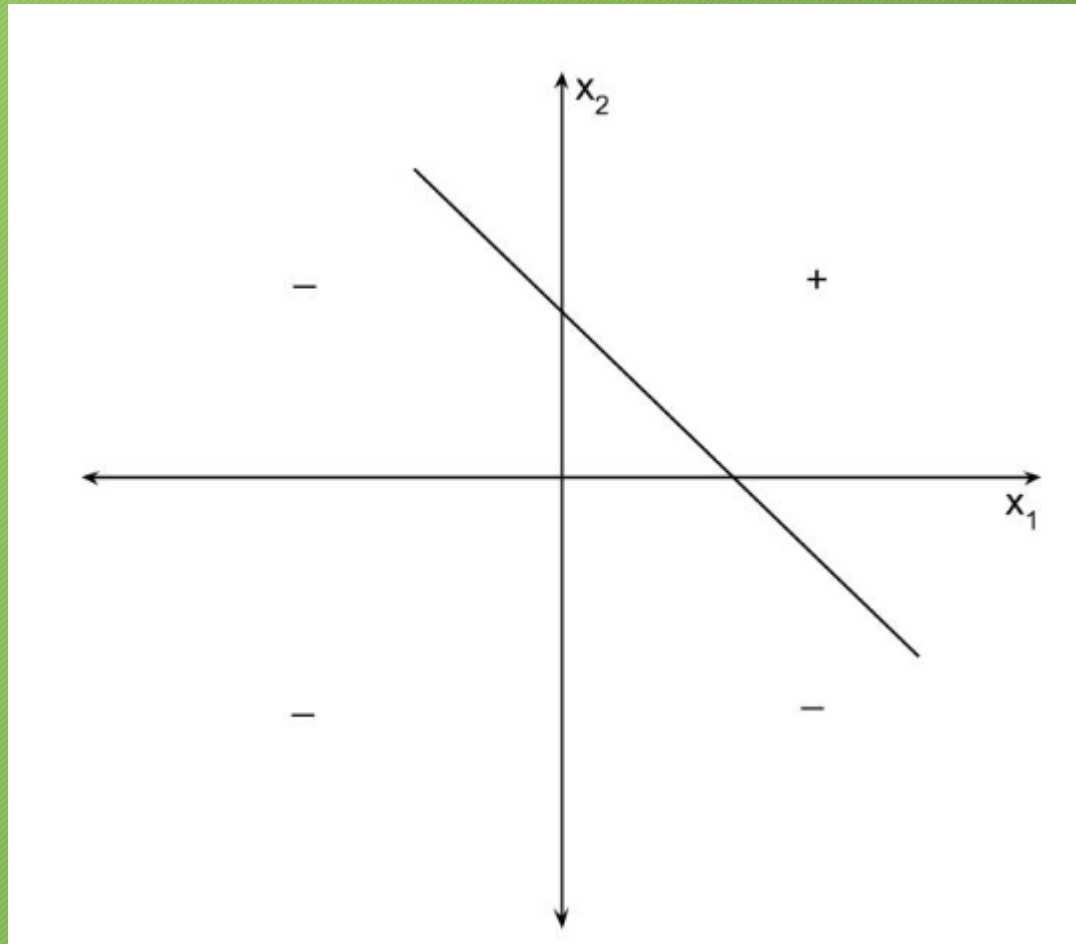
- For $x_1 = -1$, $x_2 = -1$, $b = 1$, $Y = (-1)(2) + (-1)(2) + (1)(-2) = -6$
- For $x_1 = -1$, $x_2 = 1$, $b = 1$, $Y = (-1)(2) + (1)(2) + (1)(-2) = -2$
- For $x_1 = 1$, $x_2 = -1$, $b = 1$, $Y = (1)(2) + (-1)(2) + (1)(-2) = -2$
- For $x_1 = 1$, $x_2 = 1$, $b = 1$, $Y = (1)(2) + (1)(2) + (1)(-2) = 2$
- The results are all compatible with the original table.

Decision Boundary :

- $2x_1 + 2x_2 - 2b = y$
- Replacing y with 0, $2x_1 + 2x_2 - 2b = 0$
- Since bias, $b = 1$, so $2x_1 + 2x_2 - 2(1) = 0$
- $2(x_1 + x_2) = 2$
- The final equation, $x_2 = -x_1 + 1$

Hebbian Learning Rule

11



Perceptron Learning Rule

12

- This rule is an error correcting the supervised learning algorithm of single layer feedforward networks with linear activation function, introduced by Rosenblatt.
- **Basic Concept** – As being supervised in nature, to calculate the error, there would be a comparison between the desired/target output and the actual output. If there is any difference found, then a change must be made to the weights of connection.

Perceptron Learning Rule

13

- **Mathematical Formulation** – To explain its mathematical formulation, suppose we have ‘n’ number of finite input vectors, $\times n$, along with its desired/target output vector t_n , where $n=1$ to N . Now the output ‘y’ can be calculated, as explained earlier on the basis of the net input, and activation function being applied over that net input can be expressed as follows –

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

Where θ is threshold.

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

Case I – when $t \neq y$, then

$$w(new) = w(old) + tx$$

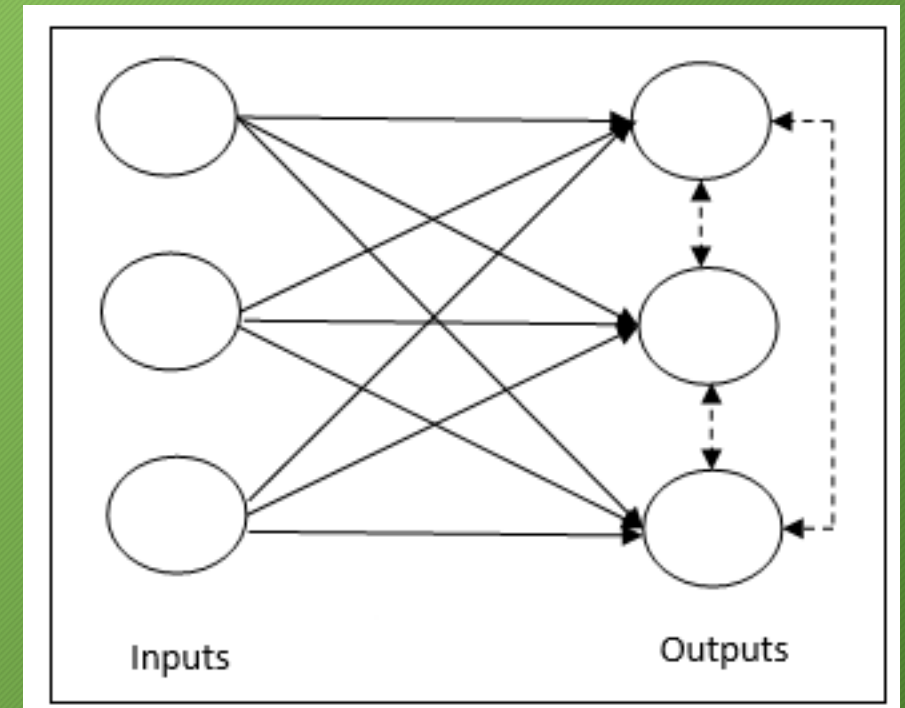
Case II – when $t = y$, then

No change in weight

Competitive Learning Rule Winner–takes–all

14

- It is concerned with unsupervised training in which the output nodes try to compete with each other to represent the input pattern. To understand this learning rule, we must understand the competitive network which is given as follows –
- **Basic Concept of Competitive Network** – This network is just like a single layer feedforward network with feedback connection between outputs. The connections between outputs are inhibitory type, shown by dotted lines, which means the competitors never support themselves.



Competitive Learning Rule Winner–takes–all

15

- **Basic Concept of Competitive Learning Rule** – As said earlier, there will be a competition among the output nodes. Hence, the main concept is that during training, the output unit with the highest activation to a given input pattern, will be declared the winner. This rule is also called Winner-takes-all because only the winning neuron is updated and the rest of the neurons are left unchanged.

- **Mathematical formulation** – Following are the three important factors for mathematical formulation of this learning rule –
- **Condition to be a winner** – Suppose if a neuron y_k wants to be the winner then there would be the following condition –

$$y_k = \begin{cases} 1 & \text{if } v_k > v_j \text{ for all } j, j \neq k \\ 0 & \text{otherwise} \end{cases}$$

It means that if any neuron, y_k , wants to win, then its induced local field *the output of summation unit*, say v_k , must be the largest among the other neurons in the network.

Competitive Learning Rule Winner–takes–all

16

- **Condition of sum total of weight** – Another constraint over the competitive learning rule is, the sum total of weights to a particular output neuron is going to be 1. For example, if we consider neuron **k** then –.

$$\sum_j w_{kj} = 1 \quad \text{for all } k$$

- **Change of weight for winner** – If a neuron does not respond to the input pattern, then no learning takes place in that neuron. However, if a particular neuron wins, then the corresponding weights are adjusted as follows

$$\Delta w_{kj} = \begin{cases} -\alpha(x_j - w_{kj}), & \text{if neuron } k \text{ wins} \\ 0, & \text{if neuron } k \text{ losses} \end{cases}$$

Here α is the learning rate.

This clearly shows that we are favoring the winning neuron by adjusting its weight and if there is a neuron loss, then we need not bother to re-adjust its weight.

Outstar Learning Rule

17

- This rule, introduced by Grossberg, is concerned with supervised learning because the desired outputs are known. It is also called Grossberg learning.
- **Basic Concept** – This rule is applied over the neurons arranged in a layer. It is specially designed to produce a desired output **d** of the layer of **p** neurons.
- **Mathematical Formulation** – The weight adjustments in this rule are computed as follows

$$\Delta w_j = \alpha (d - w_j)$$

Here **d** is the desired neuron output and α is the learning rate.

Delta Learning Rule Widrow–Hoff Rule

18

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

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

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 $t \neq y$, then

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

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 i th 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 i th input unit to the j th output unit (for each pattern) is

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

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.

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

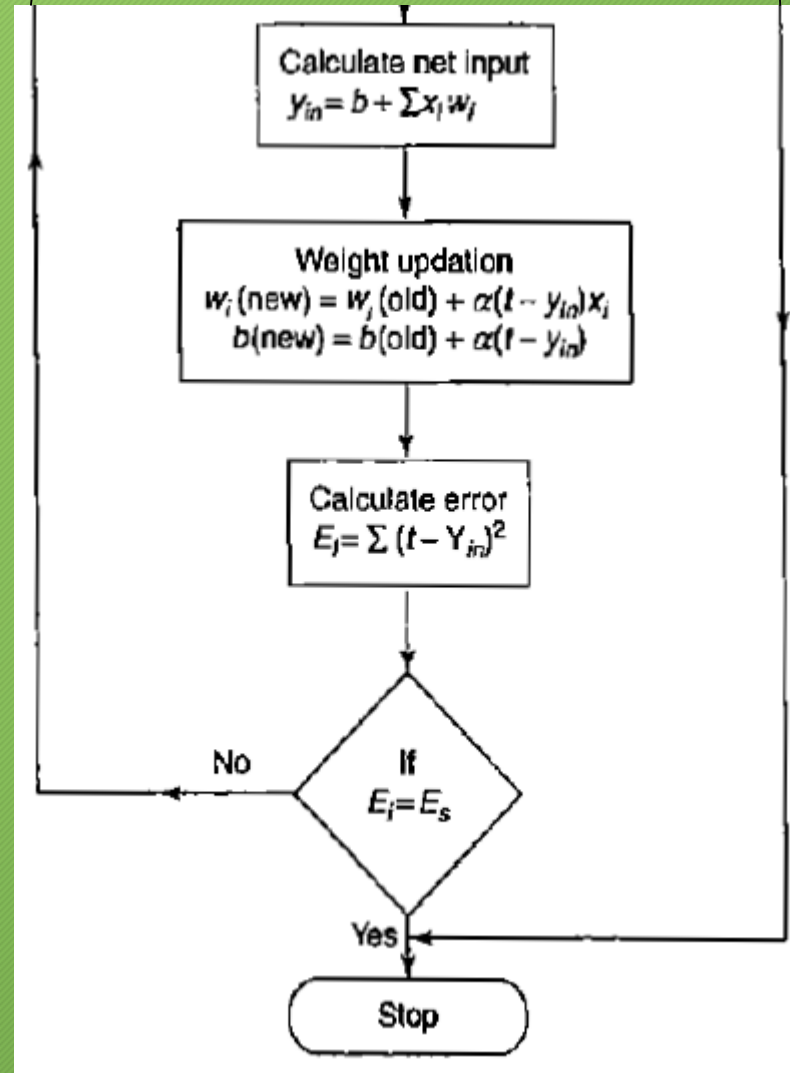
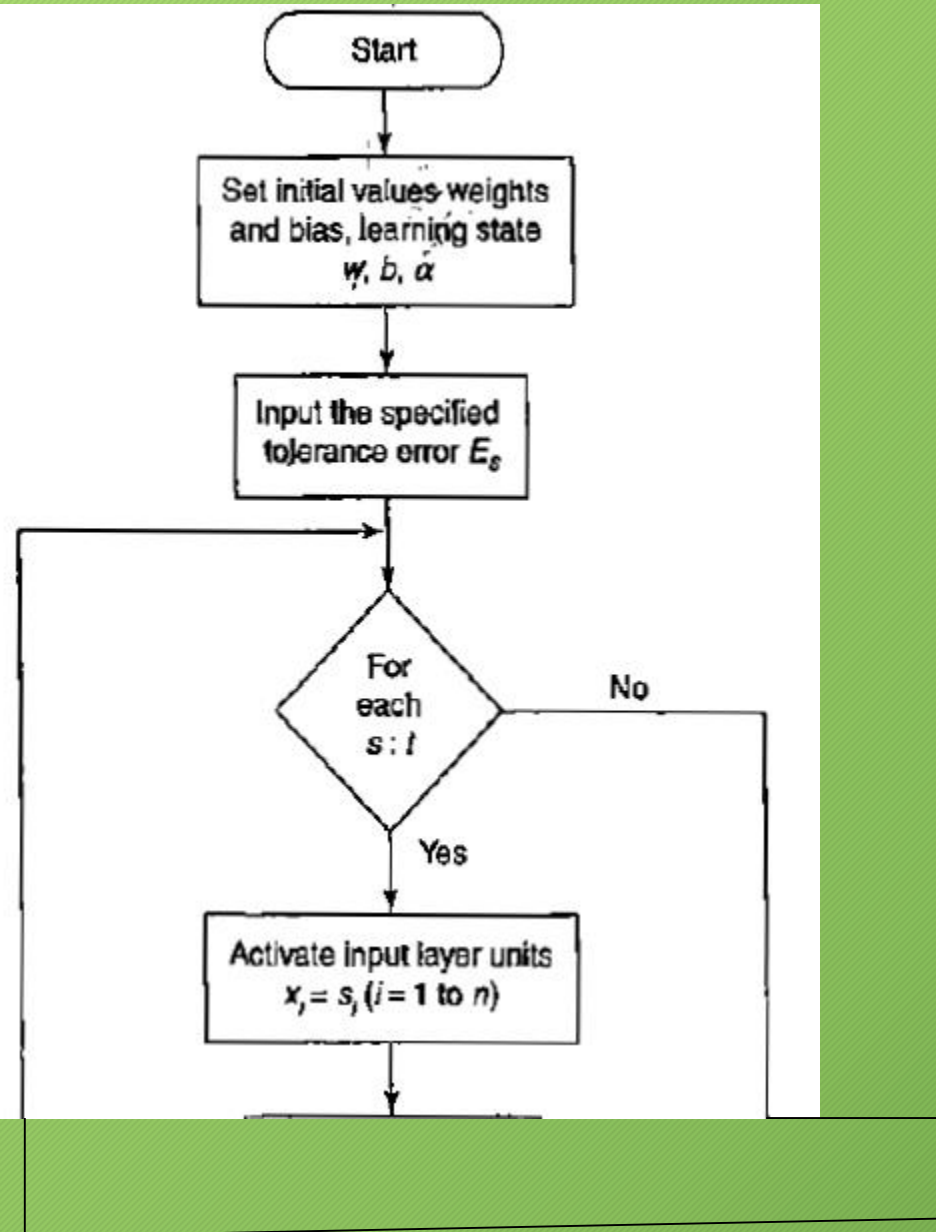
Step 5: Update the weights and bias for $i = 1$ to n :

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

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

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.



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	t
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

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

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

$$\begin{aligned}
 w_1(\text{new}) &= w_1(\text{old}) + \Delta w_1 = 0.1 + 0.1 \times 0.7 \times 1 \\
 &= 0.1 + 0.07 = 0.17
 \end{aligned}$$

$$\begin{aligned}
 w_2(\text{new}) &= w_2(\text{old}) + \Delta w_2 = 0.1 \\
 &\quad + 0.1 \times 0.7 \times 1 = 0.17
 \end{aligned}$$

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

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 t	Net input y_{in}	$(t - y_{in})$	Weight changes			Weights			Error $(t - y_{in})^2$
x_1	x_2	1				Δw_1	Δw_2	Δb	w_1 (0.1)	w_2 0.1	b (0.1)	
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

Back-Propagation Network (BPN)

28

- The back-propagation learning algorithm is applied to multilayer feed-forward networks consisting of processing elements with continuous differential activation functions. Backpropagation is a supervised learning algorithm, for training Multi-layer Perceptrons (Artificial Neural Networks).
- The network associated with back-propagation learning algorithm are also called **back-propagation networks (BPN)**.
- The aim of neural network is to train the net to achieve a balance between the net's ability to respond (memorization) and its ability to give reasonable responses to input that is similar but not identical to the one that is used in training algorithm (generalization).
- The training of BPN is done in three stages-the feed-forward of the input training pattern, the calculation and back-propagation of the error, and updation of weights.
- The testing of BPN involves the computation of feed-forward phase only.
- There can be more than one hidden layer but one hidden layer is sufficient.
- Even though the training is very slow, once the network is trained it can produce its outputs very rapidly.

Architecture of Back-Propagation Network

29

- 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, which are the connections from the units whose activation is always 1.
- The bias terms also acts as weights. Figure 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) or bipolar (-1, + 1). The activation function could be any function which increases monotonically and is also differentiable.

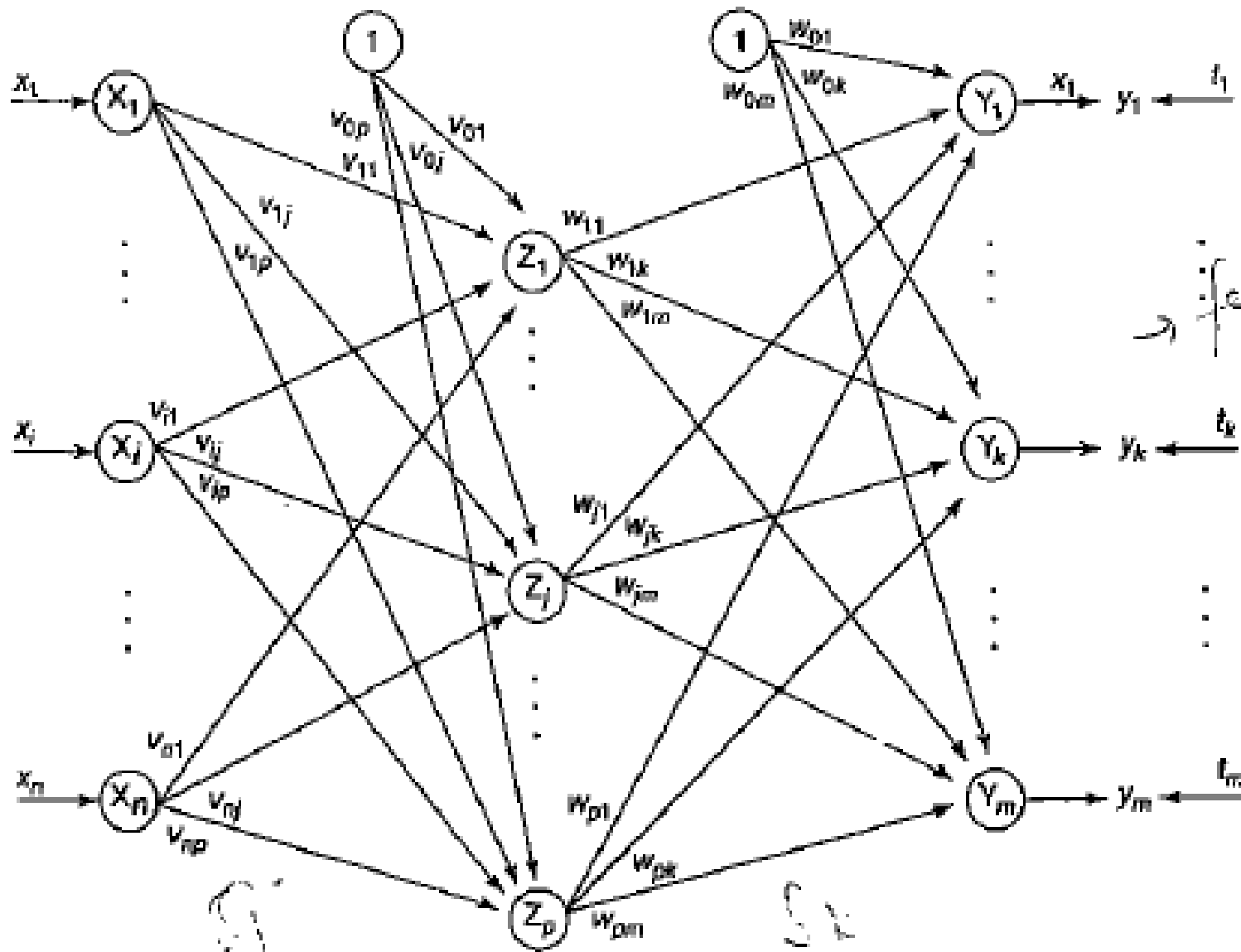


Figure 3-9 Architecture of a back-propagation network.

Solve numerical related to BPN from book already sent to you

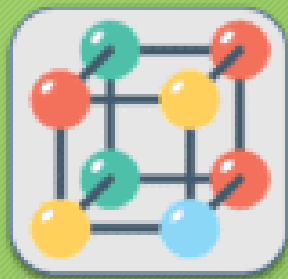
Why We Need Backpropagation?

31

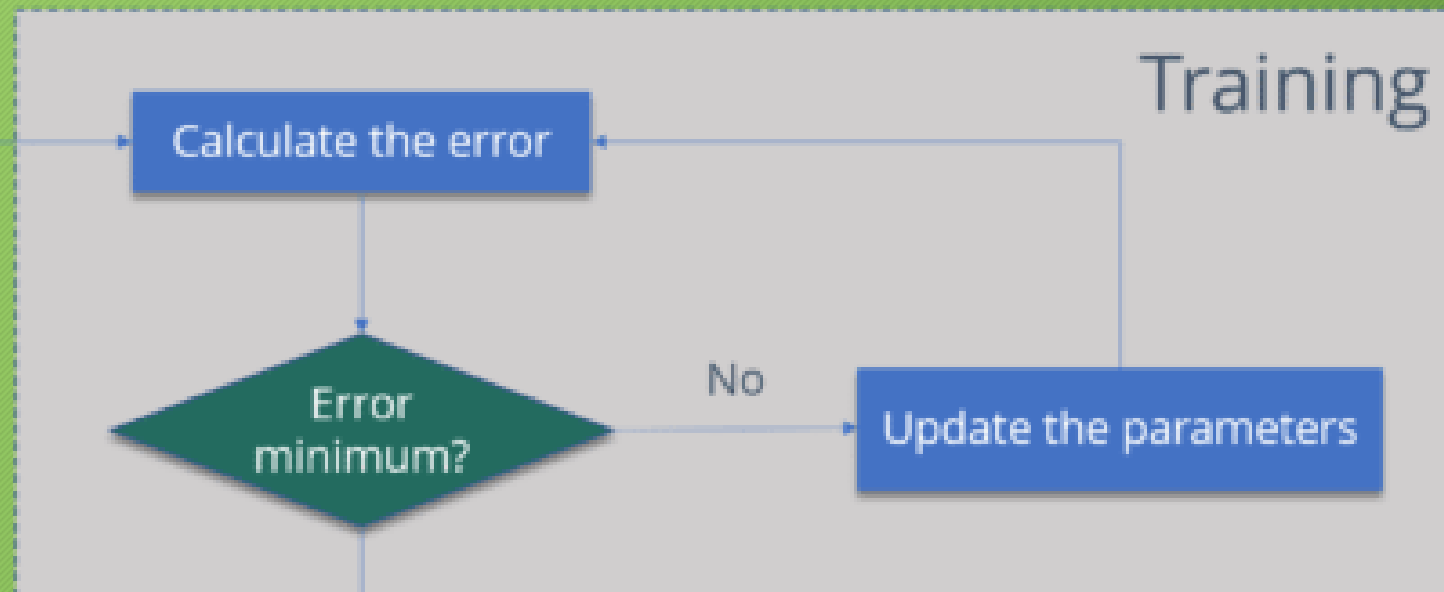
- While designing a Neural Network, in the beginning, we initialize weights with some random values or any variable for that fact.
- Now obviously, we are not *superhuman*. So, it's not necessary that whatever weight values we have selected will be correct, or it fits our model the best.
- Okay, fine, we have selected some weight values in the beginning, but our model output is way different than our actual output i.e. the error value is huge.
- Now, how will you reduce the error?
- Basically, what we need to do, we need to somehow explain the model to change the parameters (weights), such that error becomes minimum.
- Let's put it in another way, we need to train our model.
- One way to train our model is called as Backpropagation. Consider the diagram below:

Back-Propagation Network

32



Model



Model is ready to make prediction

Back-Propagation Network

33

Let me summarize the steps

- **Calculate the error** – How far is your model output from the actual output.
- **Minimum Error** – Check whether the error is minimized or not.
- **Update the parameters** – If the error is huge then, update the parameters (weights and biases). After that again check the error. Repeat the process until the error becomes minimum.
- **Model is ready to make a prediction** – Once the error becomes minimum, you can feed some inputs to your model and it will produce the output.

What is Backpropagation?

34

- The Backpropagation algorithm looks for the minimum value of the error function in weight space using a technique called the delta rule or gradient descent. The weights that minimize the error function is then considered to be a solution to the learning problem.
- Let's understand how it works with an example:
- You have a dataset, which has labels.
- Consider the below table:

Input	Desired Output
0	0
1	2
2	4

Now the output of your model when 'W' value is 3:

Input	Desired Output	Model output (W=3)
0	0	0
1	2	3
2	4	6

Notice the difference between the actual output and the desired output:

Input	Desired Output	Model output (W=3)	Absolute Error	Square Error
0	0	0	0	0
1	2	3	1	1
2	4	6	2	4

Let's change the value of 'W'. Notice the error when 'W' = '4'

Input	Desired Output	Model output (W=3)	Absolute Error	Square Error	Model output (W=4)	Square Error
0	0	0	0	0	0	0
1	2	3	1	1	4	4
2	4	6	2	4	8	16

Now if you notice, when we increase the value of 'W' the error has increased. So, obviously there is no point in increasing the value of 'W' further. But, what happens if I decrease the value of 'W'? Consider the table below:

Input	Desired Output	Model output (W=3)	Absolute Error	Square Error	Model output (W=2)	Square Error
0	0	0	0	0	0	0
1	2	3	2	4	3	0
2	4	6	2	4	4	0

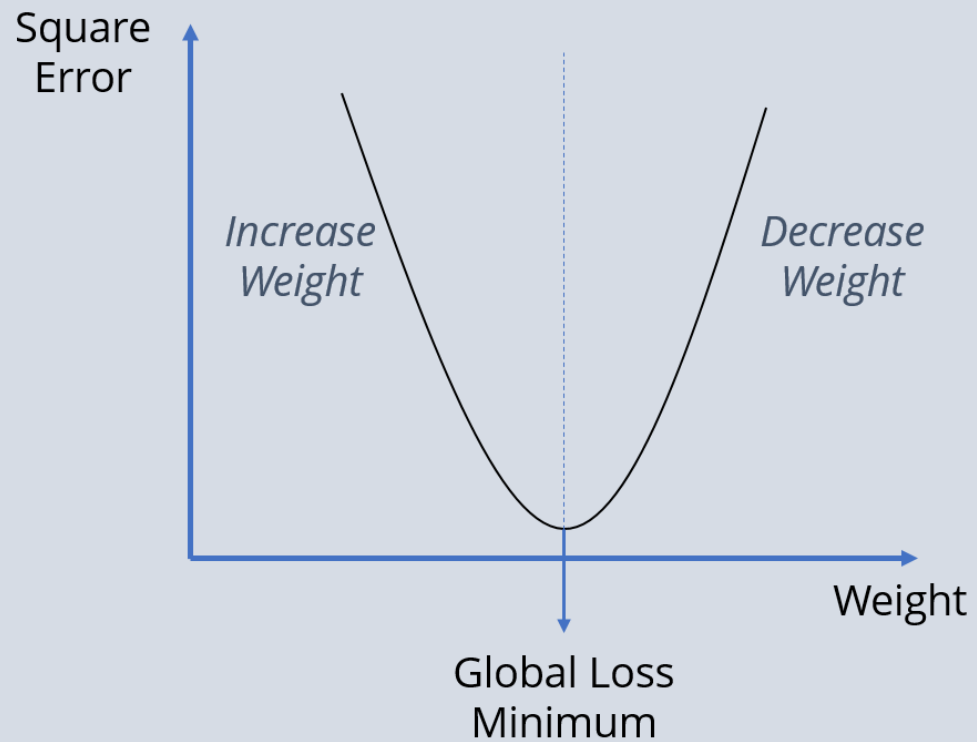
Back-Propagation Network

37

Now, what we did here:

- We first initialized some random value to 'W' and propagated forward.
- Then, we noticed that there is some error. To reduce that error, we propagated backwards and increased the value of 'W'.
- After that, also we noticed that the error has increased. We came to know that, we can't increase the 'W' value.
- So, we again propagated backwards and we decreased 'W' value.
- Now, we noticed that the error has reduced.

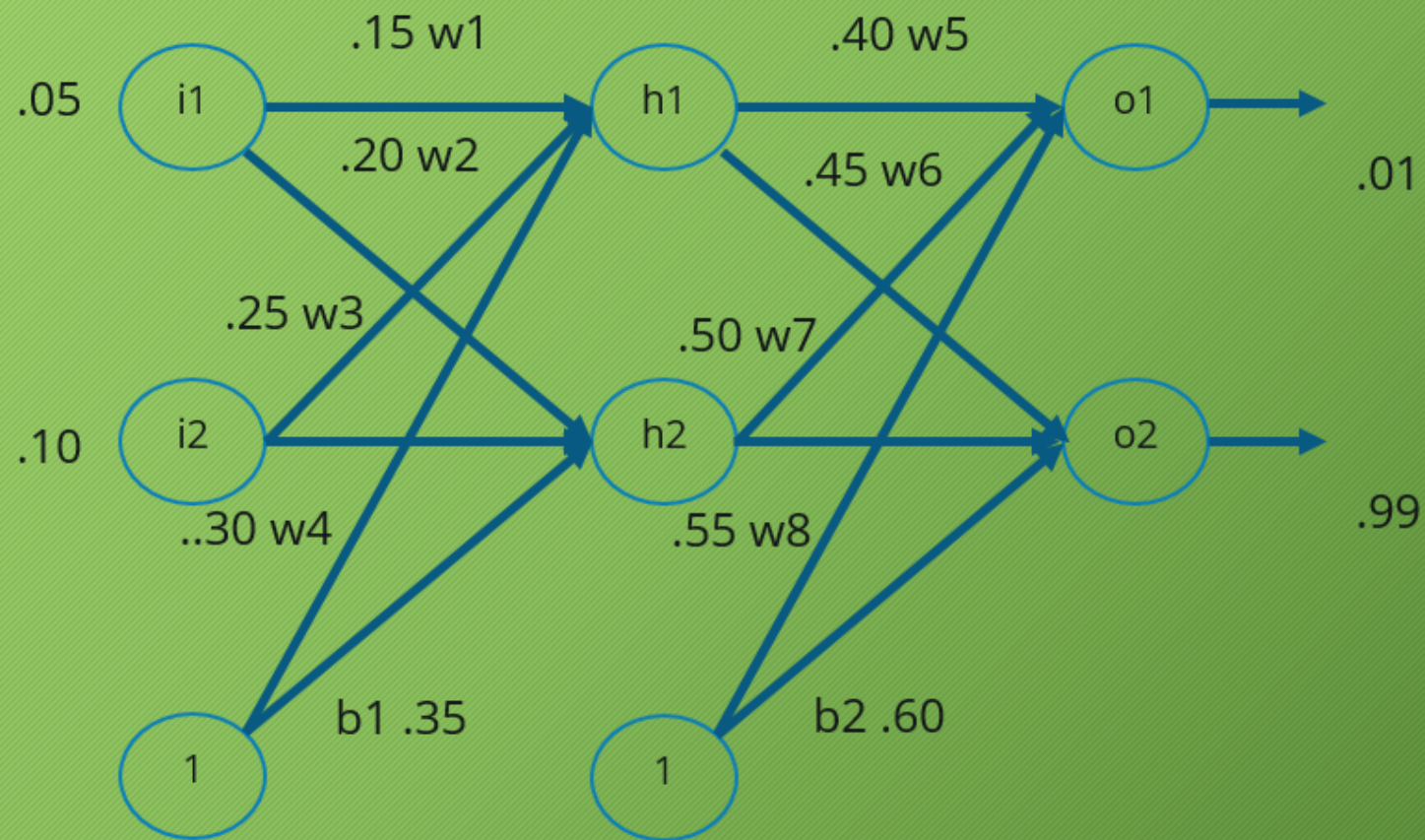
So, we are trying to get the value of weight such that the error becomes minimum. Basically, we need to figure out whether we need to increase or decrease the weight value. Once we know that, we keep on updating the weight value in that direction until error becomes minimum. You might reach a point, where if you further update the weight, the error will increase. At that time you need to stop, and that is your final weight value.



- We need to reach the ‘Global Loss Minimum’.
- This is nothing but Backpropagation.
- Let’s now understand the math behind Backpropagation.

How Backpropagation Works?

39



Back-Propagation Network

40

The above network contains the following:

- two inputs
- two hidden neurons
- two output neurons
- two biases

Below are the steps involved in Backpropagation:

- Step – 1: Forward Propagation
- Step – 2: Backward Propagation
- Step – 3: Putting all the values together and calculating the updated weight value

Step – 1: Forward Propagation: We will start by propagating forward.

Net Input For h1:

$$\text{net h1} = w1*i1 + w2*i2 + b1*1$$

$$\text{net h1} = 0.15*0.05 + 0.2*0.1 + 0.35*1 = 0.3775$$

Output Of h1:

$$\text{out h1} = 1/1+e^{-\text{net h1}}$$

$$1/1+e^{.3775} = 0.593269992$$

Output Of h2:

$$\text{out h2} = 0.596884378$$

We will repeat this process for the output layer neurons, using the output from the hidden layer neurons as inputs.

Output For o1:

$$\text{net } o1 = w5 * \text{out } h1 + w6 * \text{out } h2 + b2 * 1$$

$$0.4 * 0.593269992 + 0.45 * 0.596884378 + 0.6 * 1 = 1.105905967$$

$$\text{Out } o1 = 1 / (1 + e^{-\text{net } o1})$$

$$1 / (1 + e^{-1.105905967}) = 0.75136507$$

Output For o2:

$$\text{Out } o2 = 0.772928465$$

Now, let's see what is the value of the error:

Error For o1:

$$E_{o1} = \sum 1/2 (\text{target} - \text{output})^2$$

$$\frac{1}{2} (0.01 - 0.75136507)^2 = 0.274811083$$

Error For o2:

$$E_{o2} = 0.023560026$$

Total Error:

$$E_{\text{total}} = E_{o1} + E_{o2}$$

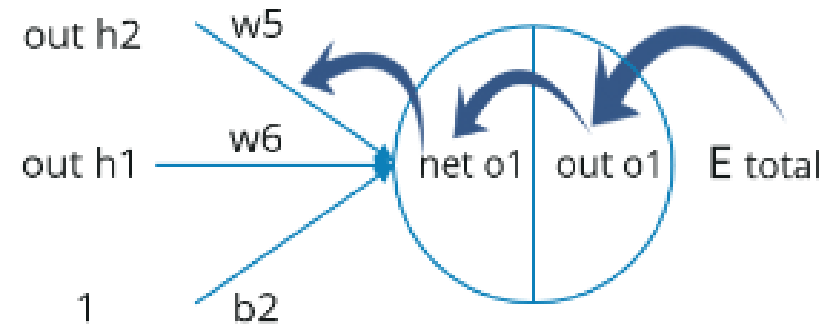
$$0.274811083 + 0.023560026 = 0.298371109$$

Step – 2: Backward Propagation

Now, we will propagate backwards. This way we will try to reduce the error by changing the values of weights and biases.

Consider W5, we will calculate the rate of change of error w.r.t change in weight W5.

$$\frac{\delta E_{total}}{\delta w_5} = \frac{\delta E_{total}}{\delta out\ o1} * \frac{\delta out\ o1}{\delta net\ o1} * \frac{\delta net\ o1}{\delta w_5}$$



Since we are propagating backwards, first thing we need to do is, calculate the change in total errors w.r.t the output O1 and O2.

$$E_{total} = 1/2(target\ o1 - out\ o1)^2 + 1/2(target\ o2 - out\ o2)^2$$

$$\frac{\delta E_{total}}{\delta out\ o1} = -(target\ o1 - out\ o1) = -(0.01 - 0.75136507) = 0.74136507$$

Now, we will propagate further backwards and calculate the change in output O1 w.r.t to its total net input.

$$\text{out } o1 = 1/1 + e^{-neto1}$$

$$\frac{\delta out\ o1}{\delta net\ o1} = \text{out } o1 (1 - \text{out } o1) = 0.75136507 (1 - 0.75136507) = 0.186815602$$

Let's see now how much does the total net input of O1 changes w.r.t W5?

$$\text{net } o1 = w5 * \text{out } h1 + w6 * \text{out } h2 + b2 * 1$$

$$\frac{\delta net\ o1}{\delta w5} = 1 * \text{out } h1\ w5^{(1-1)} + 0 + 0 = 0.593269992$$

Step - 3: Putting all the values together and calculating the updated weight value

Now, let's put all the values together:

$$\frac{\delta E_{total}}{\delta w5} = \frac{\delta E_{total}}{\delta out\ o1} * \frac{\delta out\ o1}{\delta net\ o1} * \frac{\delta net\ o1}{\delta w5}$$

0.082167041

Let's calculate the updated value of W5:

$$w5^+ = w5 - \eta \frac{\delta E_{total}}{\delta w5}$$

$$w5^+ = 0.4 - 0.5 * 0.082167041$$

Updated w5

0.35891648

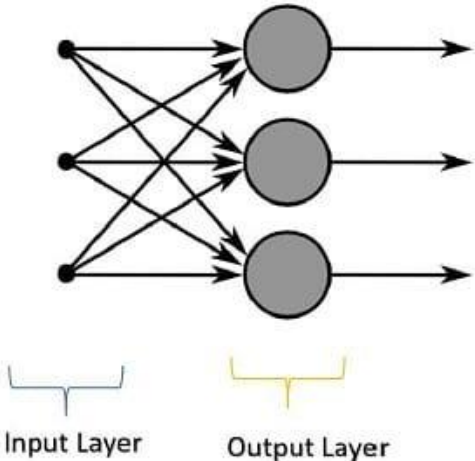
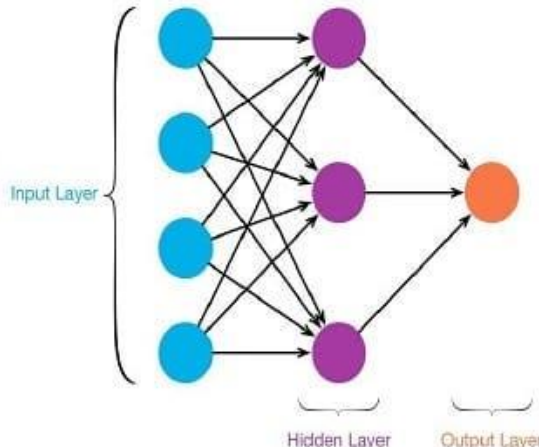
Back-Propagation Network

46

- Similarly, we can calculate the other weight values as well.
- After that we will again propagate forward and calculate the output. Again, we will calculate the error.
- If the error is minimum we will stop right there, else we will again propagate backwards and update the weight values.
- This process will keep on repeating until error becomes minimum.

Conclusion:

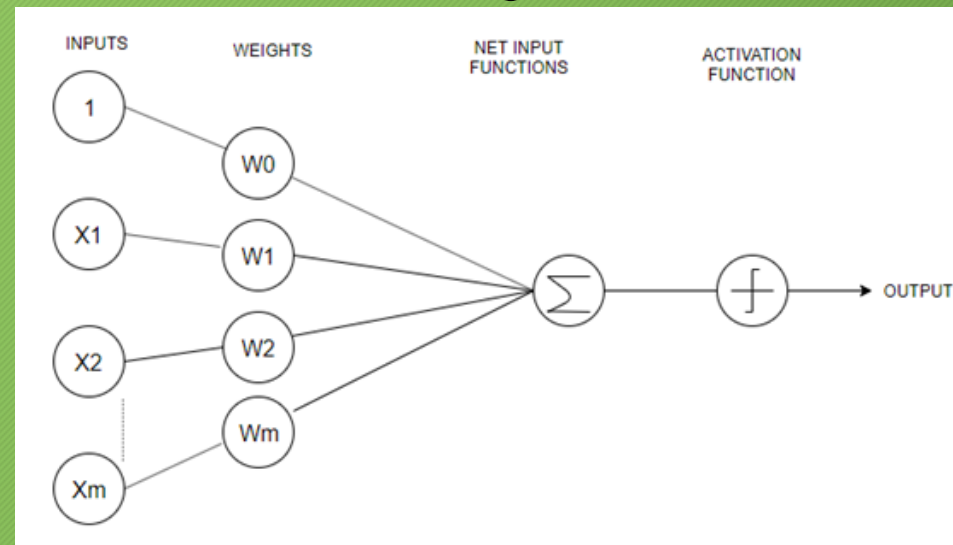
- Well, if I have to conclude Backpropagation, the best option is to write pseudo code for the same.

Single Layer Feed-Forward Neural Network	Multi Layer Feed-Forward Neural Network
Layer is formed by taking processing element & combining it with other processing element.	It is formed by interconnection of several layers.
Input & output are linked with each other.	There are multiple layers between input & output layers which are known as hidden layers.
Inputs are connected to the processing nodes with various weights resulting series of output one per node.	Input layers receives input & buffers input signal, output layer generates output.
Zero hidden layers are present.	Zero to several hidden layers are in a network.
Not efficient in certain areas.	More the hidden layers, more the complexity of networks, but efficient output is produced.
	

Terminologies related to ANN

48

Perceptron: A perceptron also called an artificial neuron is a neural network unit that does certain computations to detect features. It is a single-layer neural network used as a linear classifier while working with a set of input data. Since perceptron uses classified data points which are already labeled, it is a supervised learning algorithm. This algorithm is used to enable neurons to learn and process elements in the training set one at a time.



Terminologies related to ANN

49

There are two types of perceptrons:

- 1. Single-Layer Perceptrons:** Single-layer perceptrons can learn only linearly separable patterns.
- 2. Multilayer Perceptrons:** Multilayer perceptrons, also known as feedforward neural networks having two or more layers have a higher processing power.

Terminologies related to ANN

50

- **Loss functions:** The loss function is used as a measure of accuracy to identify whether our neural network has learned the patterns accurately or not with the help of the training data. This is completed by comparing the training data with the testing data. Therefore, the loss function is considered as a primary measure for the performance of the neural network. In Deep Learning, a good performing neural network will have a low value of the loss function at all times when training happens.

Terminologies related to ANN

51

What is the role of the Activation functions in Neural Networks?

1. The idea behind the activation function is to introduce nonlinearity into the neural network so that it can learn more complex functions.
2. Without the Activation function, the neural network behaves as a linear classifier, learning the function which is a linear combination of its input data.
3. The activation function converts the inputs into outputs.
4. The activation function is responsible for deciding whether a neuron should be activated i.e, fired or not.
5. To make the decision, firstly it calculates the weighted sum and further adds bias with it.

Terminologies related to ANN

52

What is the role of the Activation functions in Neural Networks?

1. The idea behind the activation function is to introduce nonlinearity into the neural network so that it can learn more complex functions.
2. Without the Activation function, the neural network behaves as a linear classifier, learning the function which is a linear combination of its input data.
3. The activation function converts the inputs into outputs.
4. The activation function is responsible for deciding whether a neuron should be activated i.e, fired or not.
5. To make the decision, firstly it calculates the weighted sum and further adds bias with it.

Terminologies related to ANN

53

Cost Function

- While building deep learning models, our whole objective is to minimize the cost function.
- A cost function explains how well the neural network is performing for its given training data and the expected output.
- It may depend on the neural network parameters such as weights and biases. As a whole, it provides the performance of a neural network.

Terminologies related to ANN

54

Backpropagation

The backpropagation algorithm is used to train multilayer perceptrons. It propagates the error information from the end of the network to all the weights inside the network. It allows the efficient computation of the gradient or derivatives. Backpropagation can be divided into the following steps:

- It can forward the propagation of training data through the network to generate output.
- It uses target value and output value to compute error derivatives by concerning the output activations.
- It can backpropagate to calculate the derivatives of the error concerning output activations in the previous layer and continue for all the hidden layers.
- It uses the previously computed derivatives for output and all hidden layers to calculate the error derivative concerning weights.
- It updates the weights and repeats until the cost function is minimized.

Terminologies related to ANN

55

Why is ReLU the most commonly used Activation Function?

ReLU (Rectified Linear Unit) is the most commonly used activation function in neural networks due to the following reasons:

- 1. No vanishing gradient:** The derivative of the RELU activation function is either 0 or 1, so it could be not in the range of $[0,1]$. As a result, the product of several derivatives would also be either 0 or 1, because of this property, the vanishing gradient problem doesn't occur during backpropagation.
- 2. Faster training:** Networks with RELU tend to show better convergence performance. Therefore, we have a much lower run time.
- 3. Sparsity:** For all negative inputs, a RELU generates an output of 0. This means that fewer neurons of the network are firing. So we have sparse and efficient activations in the neural network.

Terminologies related to ANN

56

Epoch, iteration, and batch are different types that are used for processing the datasets and algorithms for gradient descent. All these three methods, i.e., epoch, iteration, and batch size are basically ways of working on the gradient descent depending on the size of the data set.

- **Epoch:** It represents one iteration over the entire training dataset (everything put into the training model).
- **Batch:** This refers to when we are not able to pass the entire dataset into the neural network at once due to the problem of high computations, so we divide the dataset into several batches.
- **Iteration:** Let's have 10,000 images as our training dataset and we choose a batch size of 200. then an epoch should run $(10000/200)$ iterations i.e, 50 iterations.

Terminologies related to ANN

57

Weight and Bias initialization

- Neural network initialization means initialized the values of the parameters i.e, weights and biases. Biases can be initialized to zero but we can't initialize weights with zero.
- Weight initialization is one of the crucial factors in neural networks since bad weight initialization can prevent a neural network from learning the patterns.
- On the contrary, a good weight initialization helps in giving a quicker convergence to the global minimum. As a rule of thumb, the rule for initializing the weights is to be close to zero without being too small.
- If we initialize the set of weights in the neural network as zero, then all the neurons at each layer will start producing the same output and the same gradients during backpropagation.
- As a result, the neural network cannot learn anything at all because there is no source of asymmetry between different neurons. Therefore, we add randomness while initializing the weight in neural networks.