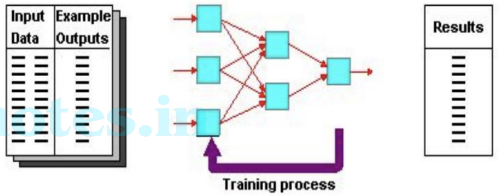
Module 2:

Supervised Learning Network

- Perceptron Networks–
  - Learning rule,
  - Training and testing algorithm.
- Adaptive Linear Neuron
  - Architecture,
  - Training and testing algorithm.
- Back propagation Network
  - Architecture,
  - Training and testing algorithm



Perceptron Networks

Supervised Learning

- supervised learning takes place under the supervision of a teacher.
- This learning process is dependent.
- During the training of ANN under supervised learning, the input vector is presented to the network, which will produce an output vector.
- This output vector is compared with the desired/target output vector.
- An error signal is generated if there is a difference between the actual output and the desired/target output vector.
- On the basis of this error signal, the weights would be adjusted until the actual output is matched with the desired output.

Layers in Neural Network

- The input layer:
  - Introduces input values into the network.
  - No activation function or other processing.
- The hidden layer(s):
  - Performs classification of features.
  - Two hidden layers are sufficient to solve any problem.
  - Features imply more layers may be better.
- The output layer:
  - Functionally is just like the hidden layers.
  - Outputs are passed on to the world outside the neural network

Single Layer , Feed Forward Network

- Single Layer network
  - In Single layer network i/p stage/layer and o/p layer are linked with each other.
  - i/p layer → transmits i/p and
  - o/p layer→ generates the o/p of the n/w
- Feed Forward Network
  - A network is said to be feed forward ,if no neuron in the o/p layer is an i/p to a node in the same /preceding layer
- Single layer feed forward n/w
  - Perceptron Network
  - Adaline Network
- Multi Layer Feed forward n/w
  - Back Propagation network

Perceptron Networks

- Also called Simple Perceptron's.
- Type of Single layer feed forward network, used to classify the given i/p data.
- Developed by Frank Rosenblatt (1962)by using McCulloch and Pitts model, perceptron is the basic operational unit of artificial neural networks.
- It employs supervised learning rule and is able to classify the data into two classes.
- A Perceptron is an algorithm for supervised learning of binary classifiers.
- This algorithm enables neurons to learn and processes elements in the training set one at a time.
- A target output is present and the i/p signals should be processed to meet the target output.(Comparing the calculated[Actual] and Target[Desired] o/p)
- It processes both binary and bipolar i/p data's and targets.

Perceptron Networks

- Perceptron Networks consist of 3 units
  - ❑ Sensory unit(input unit)
  - ❑ Associator Unit(hidden unit)
  - ❑ Response unit(output unit)
- Sensory units are connected to associator unit with a fixed weight having values 1, 0 or -1 which are assigned at random
- Binary activation functions are used in the sensory unit and associator unit
- The response unit has an activation of 1,0 or -1
- The binary step with fixed threshold θ is used as an activation for associator
- The output signal that are send from the associator to response are only binary

Perceptron Networks

- The output of perceptron network is given by y= f(Yin) ,where f(Yin) is the activation function and is defined as

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

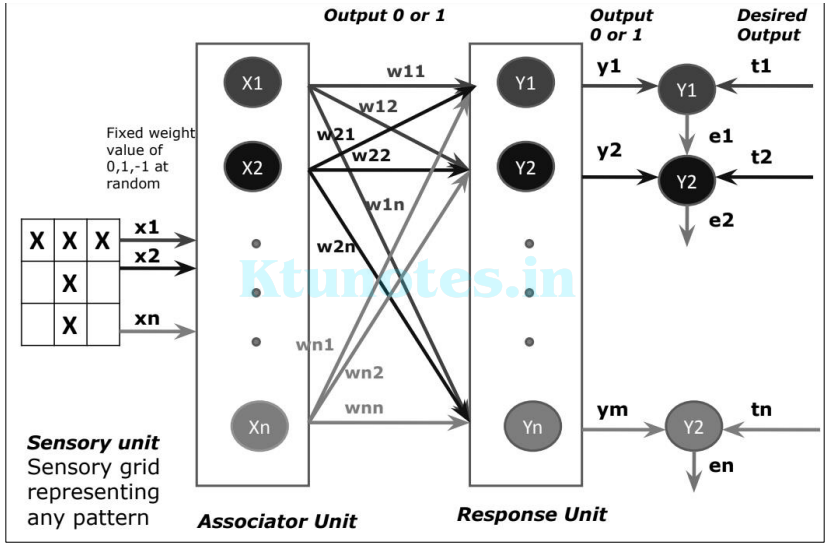
- The perceptron learning rule is used in the weight updation between the associator unit and the response unit
- For each training input, the net will calculate the response and it will determine whether or not an error has occurred
- The error calculations are based on the comparison of values for targets with those of calculated output

Perceptron Networks

- The weight **s** on the connection from the unit that send the non zero signal will get adjusted suitably
- Weight will be adjusted on the bias of the learning rule if an error has occurred for a particular training pattern

$$w_i(new) = w_i(old) + \alpha t x_i$$
$$b(new) = b(old) + \alpha t$$
where, $t-$ target value$(+1 or -1)$
$\alpha-$ learning rate

- $\alpha-$Learning Rate is used to control the amount of weight adjustment at each step of training. takes values between 0 to 1
- In general, these learning rules begin with an initial guess at the weight values and then successive adjustments are made on the basis of the evaluation of an objective function. Eventually, the learning rules reach a near-optimal or optimal solution in a finite number of steps



Perceptron Networks

- Sensory unit
  - can be a 2D matrix of 400 photo detectors
  - These detectors provide a binary electrical signal if the input signal is found to exceed a certain value of threshold
  - Also these detectors are connected randomly with the associator unit
- Associator unit
  - consist of a set of sub circuits called feature predicates
  - For a particular feature each predicate is examined with a few or all of the responses of sensory unit
  - The result from these predicate unit is also binary
- Response Unit
  - Contain pattern recognizer or perceptron
  - The weight present in the input layer are all fixed
  - While weight in the response layer are trainable

Perceptron Networks

- perceptron learning rule is used in the weight updation between the associator unit and response unit.
- For each training i/p ,the net will calculate the Response and will determine whether or not an error has occurred.
- Error calculation is based on the comparison of values of targets with those of the calculated o/p's.
- The parameter, α is called the learning rate, which as the name suggests can be used to tune how quickly or slowly w is updated.
- if the learning rate is too small, the learning algorithm may fail to converge in a reasonable amount of time. If the learning rate is too large, the learning algorithm may fail to settle on a good decision boundary at all! Thus we can see that the learning rate, α, is an important parameter in the learning algorithm, and is vital to the success of the Perceptron.

Perceptron learning rule

Perceptron Networks Learning Rule

- In perceptron network, the learning rule is the difference between the desired and actual response of a neuron.
- **Learning rule explained as follows**
- Consider a finite 'n' number of input training vectors, with their associated target values 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 the basis of the net input calculated and activation function being applied over the net input

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

Perceptron Networks Learning Rule

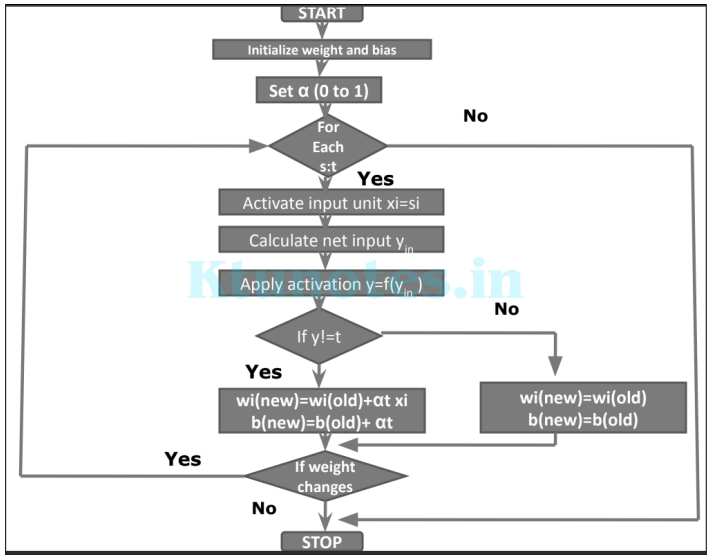
- Weight updation in case of perceptron learning is as shown
- If y ≠ t then

$$w_i(new) = w_i(old) + \alpha t x_i$$

- Else

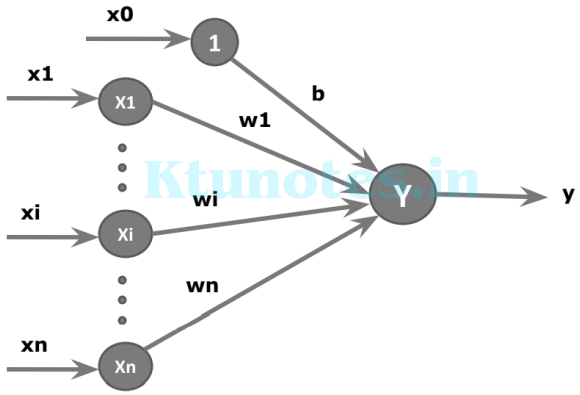
$$w(new) = w(old)$$

perceptron network- Flowchart



Perceptron Training Algorithm For Single Output Class

Perceptron Training Algorithm For Single Output Class



Perceptron Training Algorithm For Single Output Class

Step 0: Initialize the weights and bias. Also, initialize the learning rate, $\alpha (0 < \alpha \leq 1)$.

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 unit is applied with identity activation functions:

$$x_i = s_i$$

Step 4: Calculate the output of the network.

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

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

Training algorithm –Multiple Output class

Perceptron Training Algorithm For Single Output Class

Step 5: Weight and bias adjustment:

If $y \neq t$ then ,

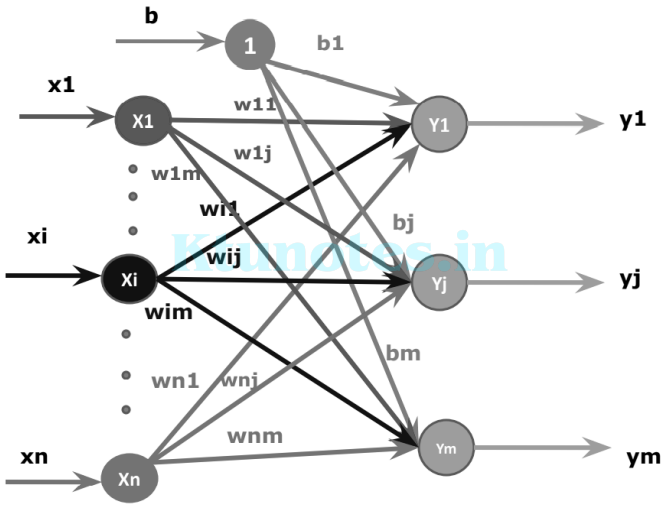
$$w_i(new) = w_i(old) + \alpha t x_i$$
$$b(new) = b(old) + \alpha t$$

else, we have

$$w(new) = w(old)$$
$$b(new) = b(old)$$

Step 6:

Train the network until there is no weight change. Otherwise, start again from Step 2.



Training algorithm –Multiple Output class

Step 0:

Initialize the weights and bias. Also, initialize the learning rate,$\alpha(0 < \alpha \leq 1)$.

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 unit is applied with identity activation functions:

$$x_i = s_i$$

Step 4: Calculate the output of the network. $y_{inj} = b_j + \sum_{i=1}^{n} x_i w_{ij}$

Perceptron –Testing Algorithm

Training algorithm –Multiple Output class

Step 5:

Make adjustment in weight and bias for $j = 1$ *to* m and $i = 1$ *to* n

If $t_i \neq y_j$ then , $w_{ij}(new) = w_{ij}(old) + \alpha t_j x_i$

$b_j(new) = b_j(old) + \alpha t_j$

else, we have $w_{ij}(new) = w_{ij}(old)$

$b_j(new) = b_j(old)$

Step 6:

Train the network until there is no weight change. Otherwise, start again from Step 2.

Testing algorithm

Step 0: Initial weights is equal to the final weights obtained during training.

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.

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

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

Testing algorithm

- The testing algorithm test the performance of the network.
- The perceptron network can be used for linear separability concept.
- Separating line may be based on the Threshold and must be a non-negative value.
- The condition for separating the response from region +ve to region 0 is

$$w_1x_1 + w_2x_2 + b > \theta$$

- The condition for separating the region of zero to region of negative is

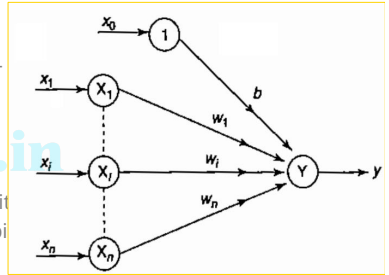
$$w_1x_1 + w_2x_2 + b < -\theta$$

- The condition above are stated for the single layer perceptron network with two i/p neurons and one o/p neurons and one bias.

Single classification perceptron network.

Single classification perceptron network Architecture

- The output obtained from the associator unit is a binary vector and hence that the output is taken as input signal to the response unit
- Here the weight between the sensory unit and associator unit are fixed
- Weight between the associator unit and response unit output unit can be adjusted
- As a result the discussion of the network is limited to a single portion ,Thus associator unit behaves like input unit
- There are n input neuron and one output neuron and a bi
- The input layer and output layer are connected through the direct communication link which is associated with weight
- The goal of perceptron network is to classify the input pattern as the member or not a member of a particular class



Perceptron Network-Problems

Perceptron Network-Problems

- The input patterns are presented to the network one by one.
- When all the four input patterns are presented ,then one EPOCH is completed
- The initial weights ,bias and threshold are set to zero.w1=w2=0,b=0
- The learning rate α is set to 1.
- Learning occurs only when a sample has y != t – Otherwise, it's similar to Hebb Learning rule
- Two loops, a completion of the inner loop (each sample is used once) is called an epoch
- Stop condition
  - When no weight is changed in the current epoch, or
  - When pre determined number of epochs is reached

Implement AND function using perceptron networks with Bipolar inputs and targets.

Truth Table		
A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1
Read as A AND B gives Q		

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

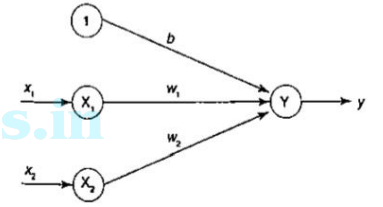


Figure Perceptron network for AND function.

Implement AND function using perceptron networks with Bipolar inputs and targets.

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

For the first input pattern, $x_1 = 1, x_2 = 1$ and $t = 1$, with weights and bias, $w_1 = 0$, $w_2 = 0$ and $b = 0$:

Calculate the net input $y_{in} = b + x_1 w_1 + x_2 w_2$

$= 0 + 1 \times 0 + 1 \times 0 = 0$

The output y is computed by applying activations over the net input calculated:

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

Here we have taken $\theta = 0$. Hence, when, $y_{in} = 0$, $y = 0$.

Check whether $t = y$. Here, $t = 1$ and $y = 0$, so $t \neq y$, hence weight updation takes place:

Implement AND function using perceptron networks with Bipolar inputs and targets.

If $y \neq t$ then *wi(new)=wi(old)+α t xi*

b(new)=b(old)+ αt

$w_1(\text{new}) = w_1(\text{old}) + \alpha t x_1 = 0 + 1 \times 1 \times 1 = 1$

$w_2(\text{new}) = w_2(\text{old}) + \alpha t x_2 = 0 + 1 \times 1 \times 1 = 1$

$b(\text{new}) = b(\text{old}) + \alpha t = 0 + 1 \times 1 = 1$

Here, the change in weights are

$\Delta w_1 = \alpha t x_1;$

$\Delta w_2 = \alpha t x_2;$

$\Delta b = \alpha t$

The weights w1=1,w2=1 and b=1 are the final weights and bias after the first input is processed.
The same processes is repeated for all other input patterns.
The process can be stopped when all the targets becomes equals to the calculated output.
Or when a separate line is obtained using the final weights for separating the positive responses from negative responses.

Implement AND function using perceptron networks with Bipolar inputs and targets. From step1

The initial weights and threshold are set to zero. i.e, $w_1 = w_2 = b = 0$ and $\theta = 0$. The learning rate α is set equal to 1.

EPOCH #1

⇒ for the first i/p pattern $x_1 = 1, x_2 = 1$ and $t = 1$ with weights and bias, $w_1 = 0, w_2 = 0$ and $b = 0$.

① Calculate the net i/p,

$$y_{in} = b + x_1 w_1 + x_2 w_2$$
$$= 0 + 1 \times 0 + 1 \times 0 = 0$$

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

Implement AND function using perceptron networks with Bipolar inputs and targets. From step1

② The net o/p y is computed by applying activations over the net i/p calculated:

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

Here $y_{in} = 0 \therefore f(y_{in}) = y = 0$.

③ check whether $t = y$. Here, $t = 1$ and $y = 0$, $\therefore t \neq y$, hence weight updation takes place

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

Implement AND function using perceptron networks with Bipolar inputs and targets. From step1

$w_i(new) = w_i(old) + \alpha t x_i$

$b_i(new) = b_i(old) + \alpha t$

i.e, $w_1(new) = w_1(old) + \alpha t x_1$

$= 0 + 1 \times 1 \times 1 = 1$

$w_2(new) = w_2(old) + \alpha t x_2$

$= 0 + 1 \times 1 \times 1 = 1$

$b(new) = b(old) + \alpha t = 0 + 1 \times 1 = 1$

The weights $w_1 = 1$, $w_2 = 1$ and $b = 1$ are the final weights after the first i/p pattern is presented.

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

Implement AND function using perceptron networks with Bipolar inputs and targets. From step1

⇒ for the second i/p pattern $x_1 = 1, x_2 = -1$ and $t = -1$ with weights and bias $w_1 = 1$, $w_2 = 1$ and $b = 1$.

① calculate the net i/p,

$$y_{in} = b + x_1 w_1 + x_2 w_2$$
$$= 1 + 1 \times 1 + -1 \times 1 = 1$$

② The o/p y is computed by applying activation over the net o/p calculated

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

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

Implement AND function using perceptron networks with Bipolar inputs and targets. From step1

x_1	x_2	t
1	1	1
1	-1	-1
-1	1	-1
-1	-1	-1

(5) check whether $y=t$. Here $t=-1$ and $y=1$, so $t \neq y$, hence weight updation takes place:

$w_2(new) = w_2(old) + \alpha t x_2$

$w_1(new) = w_1(old) + \alpha t x_1$

$b(new) = b(old) + \alpha t$

$w_1(new) = 1 + 1 \times -1 \times 1 = 0$

$b(new) = 1 + 1 \times -1 = 0$

$w_2(new) = 1 + 1 \times -1 \times -1 = 2$

Implement AND function using perceptron networks with Bipolar inputs and targets. From step1

The weights $w_1 = 0$, $w_2 = 2$ and $b = 0$ are the final weights after second i/p is presented.

⇒ for the third i/p pattern $x_1 = -1$, $x_2 = 1$ and $t = -1$ with weights and bias $w_1 = 0$ $w_2 = 2$ and $b = 0$

(1) Calculate the net i/p

$y_{in} = b + x_1 w_1 + x_2 w_2 = 0 + -1 \times 0 + 1 \times 2 = 2$

(2) The o/p, y is computed by applying activation over the net i/p calculated.

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

Here we have $y_{in} = 2$. $\therefore y = 1$

Implement AND function using perceptron networks with Bipolar inputs and targets. From step1

(3) check whether $t = y$. Here $t = -1$ and $y = 1$ so $t \neq y$, hence weight updation takes place

$w_1(new) = w_1(old) + \alpha t x_1$ $\quad w_1(new) = 0 + 1 \times -1 \times -1 = 1$

$w_2(new) = w_2(old) + \alpha t x_2$ $\quad w_2(new) = 2 + 1 \times -1 \times 1 = 1$

$b(new) = b(old) + \alpha t$ $\quad b(new) = 0 + 1 \times -1 = -1$

The weights $w_1 = 1$, $w_2 = 1$ and $b = -1$ are the final weights and bias after presenting third i/p pattern.

⇒ for the fourth i/p $x_1 = -1$, $x_2 = -1$ and $t = -1$ with weights and bias $w_1 = 1$, $w_2 = 1$ and $b = -1$

Implement AND function using perceptron networks with Bipolar inputs and targets. From step1

(1) Calculate the net i/p

$y_{in} = b + x_1 w_1 + x_2 w_2 = -1 + -1 \times 1 + -1 \times 1 = -3$

(2) The o/p y is computed by applying activations over the net i/p calculated.

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

Here $y_{in} = -3$ $\therefore y = -1$

(3) check whether $t = y$. Here $t = -1$ and $y = -1$, so $t = y$, hence no weight updation takes place

Implement AND function using perceptron networks with Bipolar inputs and targets. From step1

EPOCH #2

For the first i/p $x_1 = 1$, $x_2 = 1$ and $t = 1$
with weights and bias as $w_1 = 1$, $w_2 = 1$ and $b = -1$.

① Calculate the net i/p

$$y_{in} = b + x_1 w_1 + x_2 w_2 = -1 + 1 \times 1 + 1 \times 1 = 1$$

② The o/p y is computed by applying activation

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

Here $y_{in} = 1$, $\therefore y = 1$

Implement AND function using perceptron networks with Bipolar inputs and targets. From step1

③ Check whether $t = y$. Here $t = 1$ and $y = 1$
i.e, $t = y$, so no weight updation takes place.

For the second i/p pattern, $x_1 = 1$, $x_2 = -1$ and $t = -1$
with weights and bias $w_1 = 1$, $w_2 = 1$ and $b = -1$

① Calculate the net i/p.

$$y_{in} = b + x_1 w_1 + x_2 w_2 = -1 + 1 \times 1 + 1 \times -1 = -1$$

Implement AND function using perceptron networks with Bipolar inputs and targets. From step1

② The o/p y is computed by applying activation

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

Here $y_{in} = -1$, $\therefore y = -1$

③ Check whether $t = y$. Here $t = -1$ and $y = -1$ i.e, $t = y$,
so no weight updation takes place.

For the third i/p pattern, $x_1 = -1$, $x_2 = 1$ and $t = -1$
with weights and bias $w_1 = 1$, $w_2 = 1$ and $b = -1$

Implement AND function using perceptron networks with Bipolar inputs and targets. From step1

① Calculate the net i/p,

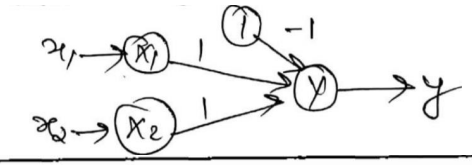
$$y_{in} = b + x_1 w_1 + x_2 w_2 = -1 + -1 \times 1 + 1 \times 1 = -1$$

② The o/p y is computed by applying activation

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

Here $y_{in} = -1$, $\therefore y = -1$

③ Check whether $t = y$. Here $t = -1$ and $y = -1$.
i.e, $t = y$, so no weight updation takes place.



Input			Target	Net input	Calculated output	Weight changes			Weights		
x_1	x_2	1	(t)	(y_{in})	(y)	Δw_1	Δw_2	Δb	w_1 (0	w_2 0	b 0)
EPOCH-1											
1	1	1	1	0	0	1	1	1	1	1	1
1	−1	1	−1	1	1	−1	1	−1	0	2	0
−1	1	1	−1	2	1	+1	−1	−1	1	1	−1
−1	−1	1	−1	−3	−1	0	0	0	1	1	−1
EPOCH-2											
1	1	1	1	1	1	0	0	0	1	1	−1
1	−1	1	−1	−1	−1	0	0	0	1	1	−1
−1	1	1	−1	−1	−1	0	0	0	1	1	−1
−1	−1	1	−1	−3	−1	0	0	0	1	1	−1

Perceptron OR

Implement OR function with Binary inputs and Bipolar targets using perceptron networks algorithm upto 3 EPOCHS

Truth Table		
A	B	Q
0	0	0
0	1	1
1	0	1
1	1	1
Read as A OR B gives Q		

x_1	x_2	t
1	1	1
1	0	1
0	1	1
0	0	−1

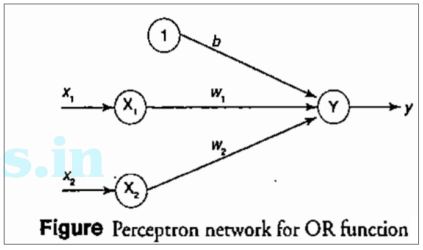


Figure Perceptron network for OR function

Implement OR function with Binary inputs and Bipolar targets using perceptron networks algorithm upto 3 EPOCHS

The initial values of the weights and bias are taken as zero, i.e.,

$$w_1 = w_2 = b = 0$$

Also the learning rate is 1 and threshold is 0.2.

the activation function becomes

$$f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} > 0.2 \\ 0 & \text{if } -0.2 \le y_{in} \le 0.2 \end{cases}$$

x_1	x_2	t
1	1	1
1	0	1
0	1	1
0	0	−1

Implement OR function with Binary inputs and Bipolar targets using perceptron networks algorithm upto 3 EPOCHS

The initial values of the weights and bias are taken as zero, i.e.,

$$w_1 = w_2 = b = 0$$

Also the learning rate is 1 and threshold is 0.2.

the activation function becomes

$$f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} > 0.2 \\ 0 & \text{if } -0.2 \le y_{in} \le 0.2 \end{cases}$$

x_1	x_2	t
1	1	1
1	0	1
0	1	1
0	0	−1

Input			Target	Net input	Calculated output	Weight changes			Weights		
x_1	x_2	1	(t)	(y_{in})	(y)	Δw_1	Δw_2	Δb	w_1 (0	w_2 0	b 0)
EPOCH-1											
1	1	1	1	0	0	1	1	1	1	1	1
1	0	1	1	2	1	0	0	0	1	1	1
0	1	1	1	2	1	0	0	0	1	1	0
0	0	1	−1	1	1	0	0	−1	1	1	0
EPOCH-2											
1	1	1	1	2	1	0	0	0	1	1	0
1	0	1	1	1	1	0	0	0	1	1	0
0	1	1	1	1	1	0	0	0	1	1	0
0	0	1	−1	0	0	0	0	0	1	1	−1
EPOCH-3											
1	1	1	1	1	1	0	0	0	1	1	−1
1	0	1	1	0	0	1	0	1	2	1	0
0	1	1	1	1	1	0	0	0	2	1	0

Perceptron ANDNOT

Find-the weights using perceptron network for AND NOT function when all the inputs are presented only one time. Use bipolar inputs and targets.

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

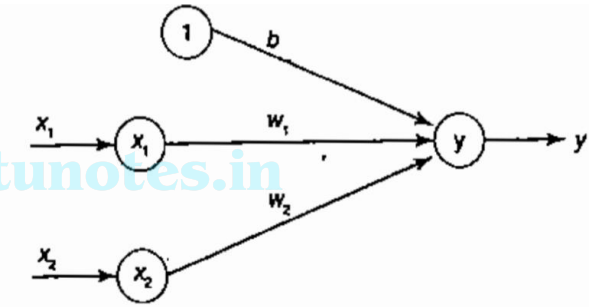


Figure Network for ANDNOT function

Find-the weights using perceptron network for AND NOT function when all the inputs are presented only one time. Use bipolar inputs and targets.

Let the initial weights be zero and $\alpha = 1, \theta = 0$.

For the first input sample, compute the net input as

$$y_{in} = b + x_1 w_1 + x_2 w_2 = 0 + 1 \times 0 + 1 \times 0 = 0$$

Applying the activation function over the net input,

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

output $y = f(y_{in}) = 0$.

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

$$w_1(\text{new}) = w_1(\text{old}) + \alpha t x_1 = 0 + 1 \times -1 \times 1 = -1$$

$$w_2(\text{new}) = w_2(\text{old}) + \alpha t x_2 = 0 + 1 \times -1 \times 1 = -1$$

$$b(\text{new}) = b(\text{old}) + \alpha t = 0 + 1 \times -1 = -1$$

Find-the weights using perceptron network for AND NOT function when all the inputs are presented only one time. Use bipolar inputs and targets.

The weights after presenting the first sample are

$$w = [-1 \; -1 \; -1]$$

For the second input sample, calculate the net input as

$$y_{in} = b + x_1 w_1 + x_2 w_2$$
$$= -1 + 1 \times -1 + (-1 \times -1) = -1 - 1 + 1 = -1$$

The output $y = f(y_{in})$ is obtained by applying activation function, hence $y = -1$. Since $t \neq y$, the new weights are calculated as

$$w_1(\text{new}) = w_1(\text{old}) + \alpha t x_1 = -1 + 1 \times 1 \times 1 = 0$$
$$w_2(\text{new}) = w_2(\text{old}) + \alpha t x_2 = -1 + 1 \times 1 \times -1 = -2$$
$$b(\text{new}) = b(\text{old}) + \alpha t = -1 + 1 \times 1 = 0$$

weights after presenting the second sample are $w = [0 \quad -2 \quad 0]$

Find-the weights using perceptron network for AND NOT function when all the inputs are presented only one time. Use bipolar inputs and targets.

For the third input sample, $x_1 = -1$, $x_2 = 1$, $t = -1$, the net input is

$$y_{in} = b + x_1 w_1 + x_2 w_2 = 0 + -1 \times 0 + 1 \times -2 = 0 + 0 - 2 = -2$$

$$w_1(\text{new}) = w_1(\text{old}) + \alpha t x_1 = -1 + 1 \times 1 \times 1 = 0$$
$$w_2(\text{new}) = w_2(\text{old}) + \alpha t x_2 = -1 + 1 \times 1 \times -1 = -2$$
$$b(\text{new}) = b(\text{old}) + \alpha t = -1 + 1 \times 1 = 0$$

weights after presenting the second sample are $w = [0 \quad -2 \quad 0]$

For the third input sample, $x_1 = -1$, $x_2 = 1$, $t = -1$,

the net input is calculated as, $y_{in} = b + x_1 w_1 + x_2 w_2$

$$= 0 + -1 \times 0 + 1 \times -2 = 0 + 0 - 2 = -2$$

Find-the weights using perceptron network for AND NOT function when all the inputs are presented only one time. Use bipolar inputs and targets.

The output is obtained as $y = f(y_{in}) = -1$. Since $t = y$, no weight changes. Thus, even after presenting the third input sample, the weights are $w = [0 \quad -2 \quad 0]$

For the fourth input sample, $x_1 = -1, x_2 = -1$, $t = -1$, the net input is calculated as

$$y_{in} = b + \sum_{i=1}^{n} x_i w_i = b + x_1 w_1 + x_2 w_2 = 0 + -1 \times 0 + (-1 \times -2)$$
$$= 0 + 0 + 2 = 2$$

Find-the weights using perceptron network for AND NOT function when all the inputs are presented only one time. Use bipolar inputs and targets.

The output is obtained as $y = f(y_{in}) = 1$. Since $t \neq y$, the new weights on updating are given as

$$w_1(\text{new}) = w_1(\text{old}) + \alpha t x_1 = 0 + 1 \times -1 \times -1 = 1$$

$$w_2(\text{new}) = w_2(\text{old}) + \alpha t x_2 = -2 + 1 \times -1 \times -1 = -1$$

$$b(\text{new}) = b(\text{old}) + \alpha t = 0 + 1 \times -1 = -1$$

The weights after presenting fourth input sample are

$$w = [1 \quad -1 \quad -1]$$

Perceptron problem

Find-the weights using perceptron network for AND NOT function when all the inputs are presented only one time. Use bipolar inputs and targets.

One epoch of training for ANDNOT function using

Input			Target	Net input	Calculated output	Weights		
x_1	x_2	b	(t)	(y_{in})	(y)	w_1 (0	w_2 0	b 0)
1	1	1	−1	0	0	−1	−1	−1
1	−1	1	1	−1	−1	0	−2	0
−1	1	1	−1	−2	−1	0	−2	0
−1	−1	1	−1	2	1	1	−1	−1

Perceptron problem

- Find the weights required to perform the following classification using perceptron network. The (vectors (1,1, 1, 1) and (-1, 1 -1, -1) are belonging to the class (so have target value 1), vectors (1, 1, 1, -1) and (1, -1, -1, 1) are not belonging to the class (so have target value -1). Assume learning rate as 1,threshold as 0.2 and initial weights as 0.

Input				
x_1	x_2	x_3	x_4	Target (t)
1	1	1	1	1
−1	1	−1	−1	1
1	1	1	−1	−1
1	−1	−1	1	−1

Perceptron problem

Let $w_1 = w_2 = w_3 = w_4 = b = 0$ and the learning rate $\alpha = 1$.

Since the threshold $\theta = 0.2$, so the activation function is

$$y = \begin{cases} 1 & \text{if } y_{in} > 0.2 \\ 0 & \text{if } -0.2 \le y_{in} \le 0.2 \\ -1 & \text{if } y_{in} < -0.2 \end{cases}$$

The net input is given by $y_{in} = b + x_1 w_1 + x_2 w_2 + x_3 w_3 + x_4 w_4$

Input				
x_1	x_2	x_3	x_4	Target (t)
1	1	1	1	1
−1	1	−1	−1	1
1	1	1	−1	−1
1	−1	−1	1	−1

Inputs					Target	Net input	Output	Weight changes					Weights				
x_1	x_2	x_3	x_4	b	(t)	(y_{in})	(y)	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	1	0	0	1	1	1	1	1	1	1	1	1	1
−1	1	−1	−1	1	1	−1	−1	−1	1	−1	−1	1	0	2	0	0	2
1	1	1	−1	1	−1	4	1	−1	−1	−1	1	−1	−1	1	−1	1	1
1	−1	−1	1	1	−1	1	1	−1	1	1	−1	−1	−2	2	0	0	0
EPOCH-2																	
1	1	1	1	1	1	0	0	1	1	1	1	1	−1	3	1	1	1
−1	1	−1	−1	1	1	3	1	0	0	0	0	0	−1	3	1	1	1
1	1	1	−1	1	−1	4	1	−1	−1	−1	1	−1	−2	2	0	2	0
1	−1	−1	1	1	−1	−2	−1	0	0	0	0	0	−2	2	0	2	0
EPOCH-3																	
1	1	1	1	1	1	2	1	0	0	0	0	0	−2	2	0	2	0
−1	1	−1	−1	1	1	2	1	0	0	0	0	0	−2	2	0	2	0
1	1	1	−1	1	−1	−2	−1	0	0	0	0	0	−2	2	0	2	0
1	−1	−1	1	1	−1	−2	−1	0	0	0	0	0	−2	2	0	2	0

Perceptron problem

Perceptron problem

- Classify the two-dimensiona1 input pattern shown in Figure using perceptron network .The symbol "*" indicates the data representation to be + 1 and "•" indicates data to be -1. The patterns are I-F. For pattern I, the target is + 1, and for F, the target is -1.

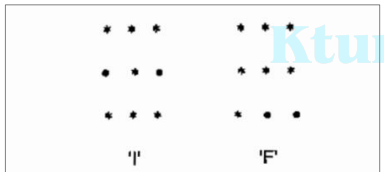


Figure I-F data representation.

Pattern	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	Target (t)
I	1	1	1	−1	1	−1	1	1	1	1
F	1	1	1	1	1	1	1	−1	−1	−1

Perceptron problem

Pattern	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	Target (t)
I	1	1	1	−1	1	−1	1	1	1	1
F	1	1	1	1	1	1	1	−1	−1	−1

The initial weights are all assumed to be zero, i.e., $\theta = 0$ and $\alpha = 1$.

The activation function is given by

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

For the first input sample, $x_1 = [1\ 1\ 1\ -1\ 1\ -1\ 1\ 1\ 1]$, $t = 1$,

the net input is calculated as

$$y_{in} = b + x_1w_1 + x_2w_2 + x_3w_3 + x_4w_4 + x_5w_5 + x_6w_6 + x_7w_7 + x_8w_8 + x_9w_9$$
$$= 0 + 1 \times 0 + 1 \times 0 + 1 \times 0 + (-1) \times 0 + 1 \times 0 + (-1) \times 0 + 1 \times 0 + 1 \times 0 + 1 \times 0$$
$$y_{in} = 0$$

Therefore, by applying the activation function the output is given by $y = f(y_{in}) = 0$. Now since $t \neq y$, the new weights are computed as

$$w_1(\text{new}) = w_1(\text{old}) + \alpha t x_1 = 0 + 1 \times 1 \times 1 = 1$$
$$w_2(\text{new}) = w_2(\text{old}) + \alpha t x_2 = 0 + 1 \times 1 \times 1 = 1$$
$$w_3(\text{new}) = w_3(\text{old}) + \alpha t x_3 = 0 + 1 \times 1 \times 1 = 1$$
$$w_4(\text{new}) = w_4(\text{old}) + \alpha t x_4 = 0 + 1 \times 1 \times -1 = -1$$
$$w_5(\text{new}) = w_5(\text{old}) + \alpha t x_5 = 0 + 1 \times 1 \times 1 = 1$$
$$w_6(\text{new}) = w_6(\text{old}) + \alpha t x_6 = 0 + 1 \times 1 \times -1 = -1$$
$$w_7(\text{new}) = w_7(\text{old}) + \alpha t x_7 = 0 + 1 \times 1 \times 1 = 1$$
$$w_8(\text{new}) = w_8(\text{old}) + \alpha t x_8 = 0 + 1 \times 1 \times 1 = 1$$
$$w_9(\text{new}) = w_9(\text{old}) + \alpha t x_9 = 0 + 1 \times 1 \times 1 = 1$$
$$b(\text{new}) = b(\text{old}) + \alpha t = 0 + 1 \times 1 = 1$$

The weights after presenting first input sample are

$$w = [1\ 1\ 1\ -1\ 1\ -1\ 1\ 1\ 1]$$

For the second input sample, $x_2 = [1\ 1\ 1\ 1\ 1\ 1\ 1\ -1\ 1]$, $t = -1$

the net input is calculated as

$$y_{in} = b + \sum_{i=1}^{9} x_i w_i$$
$$= b + x_1w_1 + x_2w_2 + x_3w_3 + x_4w_4 + x_5w_5 + x_6w_6 + x_7w_7 + x_8w_8 + x_9w_9$$
$$= 1 + 1 \times 1 + 1 \times 1 + 1 \times 1 + 1 \times -1 + 1 \times 1 + 1 \times -1 + 1 \times 1 + (-1) \times 1 + (-1) \times 1$$
$$y_{in} = 2$$

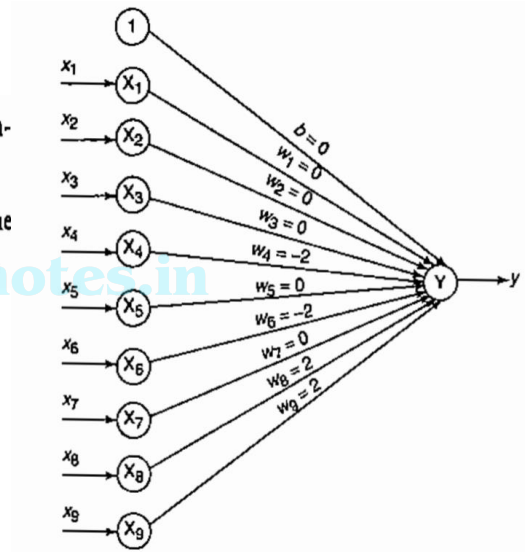
Therefore the output is given by $y = f(y_{in}) = 1$.

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

$$w_1(\text{new}) = w_1(\text{old}) + \alpha t x_1 = 1 + 1 \times -1 \times 1 = 0$$
$$w_2(\text{new}) = w_2(\text{old}) + \alpha t x_2 = 1 + 1 \times -1 \times 1 = 0$$
$$w_3(\text{new}) = w_3(\text{old}) + \alpha t x_3 = 1 + 1 \times -1 \times 1 = 0$$
$$w_4(\text{new}) = w_4(\text{old}) + \alpha t x_4 = -1 + 1 \times -1 \times 1 = -2$$
$$w_5(\text{new}) = w_5(\text{old}) + \alpha t x_5 = 1 + 1 \times -1 \times 1 = 0$$
$$w_6(\text{new}) = w_6(\text{old}) + \alpha t x_6 = -1 + 1 \times -1 \times 1 = -2$$
$$w_7(\text{new}) = w_7(\text{old}) + \alpha t x_7 = 1 + 1 \times -1 \times 1 = 0$$
$$w_8(\text{new}) = w_8(\text{old}) + \alpha t x_8 = 1 + 1 \times -1 \times -1 = 2$$
$$w_9(\text{new}) = w_9(\text{old}) + \alpha t x_9 = 1 + 1 \times -1 \times -1 = 2$$
$$b(\text{new}) = b(\text{old}) + \alpha t = 1 + 1 \times -1 = 0$$

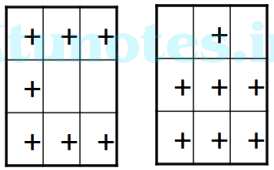
The weights after presenting the second input sample are $w = [0\ 0\ 0\ -2\ 0\ -2\ 0\ 2\ 2\ 0]$

The network architecture is as shown in Figure . The network can be further trained for its convergence.



Tutorial questions

- Implement NOR function using perceptron network for bipolar input and targets
- Classify the 2D pattern shown in the figure below using perceptron network



ADALINE Network

Adaptive Linear Neuron (Adaline)

- In 1959, Bernard Widrow and Marcian Hoff of Stanford, developed models called ADALINE (Adaptive Linear Neuron) and MADALINE (Multilayer ADALINE).
- These models were named for their use of Multiple ADAptive LINear Elements.
- MADALINE was the first neural network to be applied to a real world problem.
- It is an adaptive filter which eliminates echoes on phone lines.
- In terms of architecture, there is a noticeable difference between perceptron and Adaline: **the perceptron uses the threshold function whereas Adaline uses a linear activation function**
- Both Adaline and the Perceptron are (single-layer) neural network models.
- The Perceptron is one of the oldest and simplest learning algorithms out there, and **we consider Adaline as an improvement over the Perceptron**.

Adaptive Linear Neuron (Adaline)

- The units with linear activation function are called linear units.
- A network with a single linear unit is called an Adaline (adaptive linear neuron).
- 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**.

Adaptive Linear Neuron (Adaline)

- 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**.
- The **Widrow-Hoff rule** is very similar to percepton learning rule. However, their origins are different.
- The **perceptron learning rule** originates
  - from the Hebbian & M-P Neuron assumption

Adaptive Linear Neuron (Adaline)

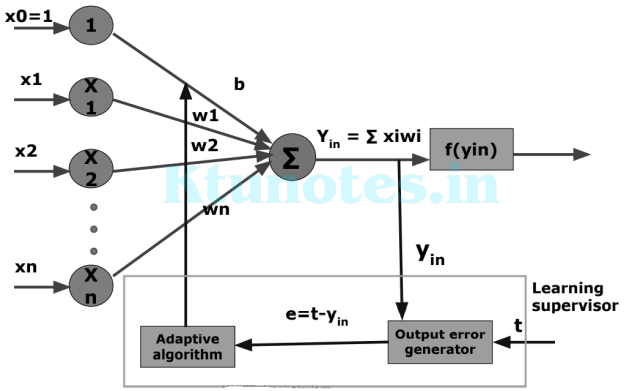
- The **delta rule** is derived from the **gradient- descent method,** which is an optimization algorithm used to find the values of parameters (coefficients) of a function (f) that minimizes a cost function (cost).[Rate of change of a function]
  - Gradient-calculating the loss or error using functions
  - Descent-Update the existing parameters in response to the gradient.
  - Gradient descent is best used when the parameters cannot be calculated analytically
- 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.

Delta Rule for single o/p -Adaline

- The **delta rule** updates the weights between the connections so as ***w*** **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 de $\Delta w_i = \alpha(t - y_{in})x_i$; the weight of ith pattern (i = 1 to *n)* is

  - where *Δwi* is the weight change;
  - *α* the learning rate;
  - xi the vector of activation of inp $Y = \sum_{i=1}^{n} x_i w_i$
  - Yin, the net input to output unit,
  - *t* the target output.

Architecture-Adaline Model



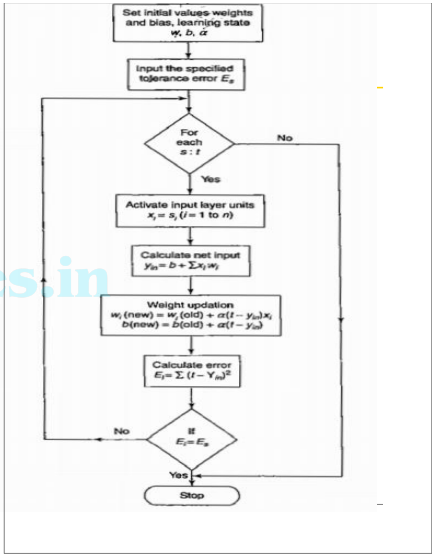
Architecture-Adaline Model

- Adaline is a **single unit neuron**, which receives input from several units and also from one unit called **bias**.
- The basic Adaline model consists of **trainable weights**.
- **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 function that restores the **output** to + 1 or -1.
- The **Adaline model** compares the actual output with the target output and on the basis of the training algorithm, the **weights are adjusted.**

TRAINING ALGORITHM & FLOWCHART

Flowchart-Adaline Model

- This gives a pictorial representation of the network training.
- The conditions necessary for weight adjustments have to be checked carefully.
- The weights and other required parameters are initialized.
- Then the net input is calculated, output is obtained and compared with the desired output for calculation of error.
- On the basis of the error Factor, weights are adjusted.



Adaline –Training Algorithm

- Step 0: Weights and bias are set to some random values but not zero.
- Set the learning rate parameter α. (The range can be between 0.1 and 1.0.)
- 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, xi=si
- 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.

$$wi(new) = wi(old) + \alpha(t - yin)xi$$

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

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

Testing Algorithm

Adaline –Testing Algorithm

- Step 0: Initialize the weights(from training algorithm)
- Step 1: Perform steps2-4 for each bipolar input vector x
- Step 2: Set the activations of the input units to x
- Step 3: Calculate the net input

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

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

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

Adaline problems-OR

Implement OR function with bipolar inputs and targets using ADALINE network

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, $x_1 = 1, x_2 = 1, t = 1$, calculate the net input

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

Implement OR function with bipolar inputs and targets using ADALINE network

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

Now compute $(t - y_{in}) = (1 - 0.3) = 0.7$.

Updating the weights we obrain,

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

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

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

The new weights are obtained as

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

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

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

calculate the error: $E = (t - y_{in})^2 = (0.7)^2 = 0.49$

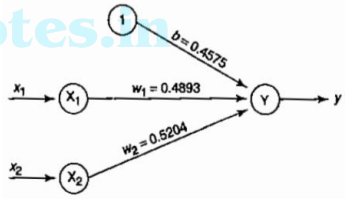
The final weights after presenting first input sample are $w = [0.17 \quad 0.17 \quad 0.17]$ and error $E = 0.49$.

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

Implement OR function with bipolar inputs and targets using ADALINE network

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.

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



Adaline problems-ANDNOT

Use Adaline network to train AND NOT function with bipolar inputs and targets. Perform 2 epochs of training.

Solution: The truth table for ANDNOT function with bipolar inputs and targets is shown in

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

Initially the weights and bias have assumed a random value say 0.2. The learning rate is also set to 0.2. The weights are calculated until the least mean square error is obtained.

The initial weights are $w_1 = w_2 = b = 0.2$, and $\alpha = 0.2$.

For the first input sample $x_1 = 1, x_2 = 1, t = -1$,

we calculate the net input as

$$y_{in} = b + x_1 w_1 + x_2 w_2$$
$$= 0.2 + 1 \times 0.2 + 1 \times 0.2 = 0.6$$

compute $(t - y_{in}) =$

Updating the weights we obtain

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

The new weights are obtained as

$$w_1(\text{new}) = w_1(\text{old}) + \alpha(t - y_{in})x_1$$
$$= 0.2 + 0.2 \times (-1.6) \times 1 = -0.12$$

Use Adaline network to train AND NOT function with bipolar inputs and targets. Perform 2 epochs of training.

$$w_2(\text{new}) = w_2(\text{old}) + \alpha(t - y_{in})x_2$$
$$= 0.2 + 0.2 \times (-1.6) \times 1 = -0.12$$

$$b(\text{new}) = b(\text{old}) + \alpha(t - y_{in})$$
$$= 0.2 + 0.2 \times (-1.6) = -0.12$$

compute the error,

$$E = (t - y_{in})^2 = (-1.6)^2 = 2.56$$

The final weights after presenting first input sample are $w = [-0.12 \; -0.12 \; -0.12]$ and error $E = 2.56$.

Inputs			Target	Net input		Weights			Error
x_1	x_2	1	t	y_{in}	$(t - y_{in})$	w_1 (0.2	w_2 0.2	b 0.2)	$(t - y_{in})^2$
EPOCH-1									
1	1	1	−1	0.6	−1.6	−0.12	−0.12	−0.12	2.56
1	−1	1	1	−0.12	1.12	0.10	−0.34	0.10	1.25
−1	1	1	−1	−0.34	−0.66	0.24	−0.48	−0.03	0.43
−1	−1	1	−1	0.21	−1.2	0.48	−0.23	−0.27	1.47
EPOCH-2									

Use Adaline network to train AND NOT function with bipolar inputs and targets. Perform 2 epochs of training.

Inputs			Target	Net input		Weights			Error
x_1	x_2	1	t	y_{in}	$(t - y_{in})$	w_1 (0.2	w_2 0.2	b 0.2)	$(t - y_{in})^2$
EPOCH-1									
1	1	1	−1	0.6	−1.6	−0.12	−0.12	−0.12	2.56
1	−1	1	1	−0.12	1.12	0.10	−0.34	0.10	1.25
−1	1	1	−1	−0.34	−0.66	0.24	−0.48	−0.03	0.43
−1	−1	1	−1	0.21	−1.2	0.48	−0.23	−0.27	1.47
EPOCH-2									
1	1	1	−1	−0.02	−0.98	0.28	−0.43	−0.46	0.95
1	−1	1	1	0.25	0.76	0.43	−0.58	−0.31	0.57
−1	1	1	−1	−1.33	0.33	0.37	−0.51	−0.25	0.106
−1	−1	1	−1	−0.11	−0.90	0.55	−0.38	0.43	0.8

2.56 + 1.25 + 0.43 + 1.47 =

5.71

Use Adaline network to train AND NOT function with bipolar inputs and targets. Perform 2 epochs of training.

The operational steps are carried for 2 epochs of training and network performance is noted. It is tabulated as shown in Table

The total mean square error at the end of two epochs is summation of the errors of all input samples as shown in Table

Table

Epoch	Total mean square error
Epoch 1	5.71
Epoch 2	2.43

Hence from Table it is clearly understood that the mean square error decreases as training progresses.

Also, it can be noted that at the end of the sixth epoch, the error becomes approximately equal to 1. The network architecture for ANDNOT function using Adaline network is shown in Figure

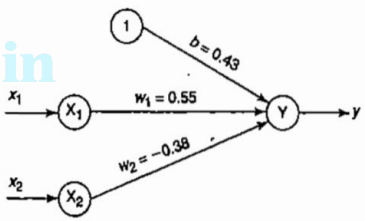


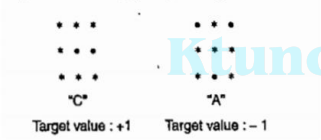
Figure Network architecture for ANDNOT function using Adaline network.

Assignment 1

Even Roll Numbers

Q1. Implement AND function using Adaline net work.

Q2. Classify the two-dimensional pattern shown in figure below using perceptron network.



Odd Roll Numbers

Q1: Using the delta rule, find the weights required to perform following classifications: Vectors (1, 1, −1, −1) and (−1, −1, −1, −1) are belonging to the class having target value 1; vectors (1, 1, 1, 1) and (−1, −1, 1, −1) are not belonging to the class having target value −1. Use a learning rate of 0.5 and assume random value of weights. Also, using each of the training vectors as input, test the response of the net.

Q2: Implement NOR function using perceptron network for bipolar inputs and targets.

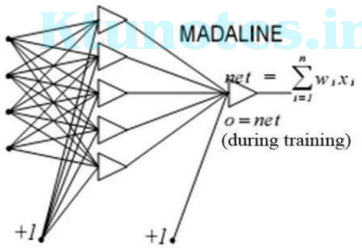
Submission date:March 17

Assignment 1

MADALINE Network

MADALINE network

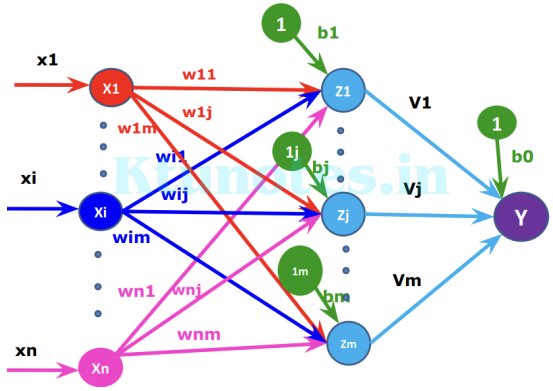
- MADALINE is a Multilayer Adaptive Linear Element.
- MADALINE was the first neural network to be applied to a real world problem.
- It is used in several adaptive filtering process.
- Two or more ADA[illegible]IADALINE model



MADALINE network

- MADALINE is a Multilayer Adaptive Linear Element.
- Used for nonlinearly separable logic functions (X-OR) function
- Used for adaptive noise cancellation and adaptive inverse control
- In noise cancellation the objective is to filter out an interference component by identifying a linear model of a measurable noise source and the corresponding immeasurable interference.
- ECG, echo elimination from long distance telephone transmission lines
- Simple MADALINE architecture consist of 'n' unit of input layer 'k' unit of Adaline layer and one unit of MADALINE layer
  - ✓ Each neuron in the adaline and MADALINE layer has a bias of excitation
  - ✓ ADALINE layer is present between the input layer and MADALINE layer hence the ADALINE layer can be considered as hidden layer

MADALINE network

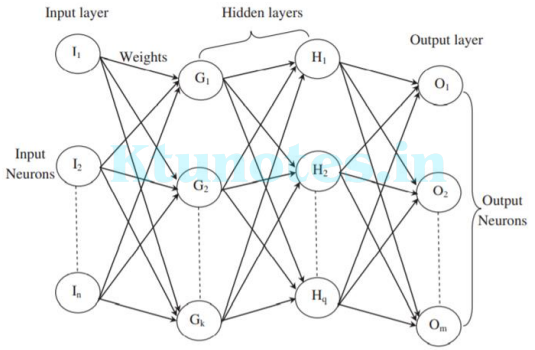


Back Propagation Network

Back Propagation Network

- The back-propagation learning algorithm is one of the most important developments in **neural networks** (Bryson and Ho, 1969; Werbos, 1974; Lecun, 1985; Parker, 1985; Rumelhan, 1986).
- This **network** has reawakened the **scientific** and **engineering community** to the modeling and processing of numerous phenomena using neural networks.
- This learning algorithm is applied to **multilayer feed-forward network** consisting on processing elements with continuous differentiable activation functions.
- The networks associated with **back-propagation learning algorithm** are also *called* ***back-propagation network*** **(BPNs).**
- For a given set of training input-output pair, this **algorithm** provides a procedure for changing the weights in a BPN to classify the given input patterns correctly.
- The basic concept for this weight update algorithm is simply the **gradient-descent method** as used in the case of simple perceptron networks with differentiable units.
- This is a method where the error is propagated back to the hidden unit

Back Propagation Network



Multilayer Feed-Forward Network

Back Propagation Network

- The **aim** of the 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 the input** that is **similar** but **not identical** to the one that is used in training (generalization).
  - Memorization
    - Ability to remember training Data
    - Poor classification on testing data
  - Generalization
    - Ability to learn from training Data
    - Good classification on testing data
- The back-propagation algorithm is different from other networks in respect to the process by which the weights are calculated during the learning period of the network.
- The **general difficulty** with the multilayer perceptron is calculating the weights of the hidden layers in an efficient way that would result in a very small or zero output error.
- When the **hidden layers are increased** the network training becomes more complex.

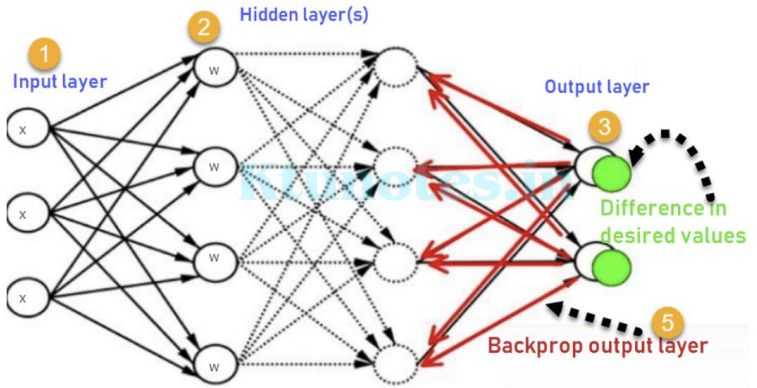
Back Propagation Network

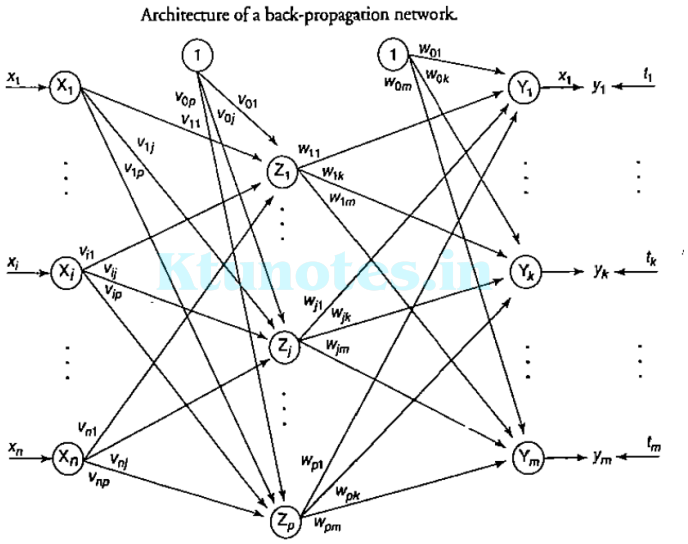
- To **update weights**, the error must be calculated. The error, Which is the difference between the actual (calculated) and the desired (target) output, is easily measured at the output layer.
- It should be noted that at the **hidden layers**, there is no direct information of the error.
- Therefore, **other techniques should be used to calculate an error at the hidden layer**, **which will cause minimization of the output error**, and this is the ultimate goal.
- The training of the BPN is done in three stages –
  - the feed-forward of the input training pattern,
  - the calculation and back-propagation of the error,
  - updation of weights.

Back Propagation Network

- 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 , *sig*nals 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

Back Propagation Network





BPN-Training Algorithm

Back Propagation Network-Training Algorithm

The training involves three stages

1. Feed forward of the input training pattern
2. Back propagation of the associated error
3. Adjustments of the weights.

During feed forward, each input unit (Xi) receives an input signal and sends this signal to each of the hidden units Z1, Z2, ...Zn.

Each hidden unit computes its activation and sends its signal to each output unit.

Each output unit computes its activation to compute the output or the response of the neural net for the given input pattern.

Back Propagation Network-Training Algorithm

- During training, each output unit compares its computed activation yk, with its target value tk to determine the associated error for the particular pattern.
- Based on this error the factor δk for all m values are computed.
- This computed δ k is used to propagate the error at the output unit Yk back to all units in the hidden layer.
- At a later stage it is also used for updation of weights between the output and the hidden layer.
- In the same way δj for all p values are computed for each hidden unit Zj.
- The values of δj are not sent back to the input units but are used to update the weights between the hidden layer and the input layer.

Back Propagation Network-Training Algorithm

- Once all the ∂ factrs are known, the weights for all layers are changed simultaneously.
- The adjustment to all weights wjk is based on the factor ∂k and the activation zj of the hidden unit Zj.
- The change in weight to the connection between the input layer and the hidden layer is based on ∂j and the activation xi of the input unit

Step 0: Initialize weights and learning rate (take some small random values).

Step 1: Perform Steps 2–9 when stopping condition is false.

Step 2: Perform Steps 3–8 for each training pair.

Feed-forward phase (Phase I)

Step 3: Each input unit receives input signal x_i and sends it to the hidden unit ($i = 1$ to n).

Step 4: Each hidden unit $z_j (j = 1 \text{ to } p)$ 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} (binary or bipolar sigmoidal activation function): $z_j = f(z_{inj})$

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 II):

Step 6: Each output unit $y_k (k = 1 \text{ to } m)$ receives a target pattern corresponding to the input training pattern and computes the error-correction term: $\delta_k = (t_k - y_k) f'(y_{ink})$

The derivative $f'(y_{ink})$ can be calculated. On the basis of the calculated error correction term, update the change in weights and bias:

$\Delta w_{jk} = \alpha \delta_k z_j; \quad \Delta w_{0k} = \alpha \delta_k$ Also, send δ_k to the hidden layer backwards.

Step 7: Each hidden unit $(z_j, j = 1 \text{ to } p)$ sums its delta inputs from the output units:

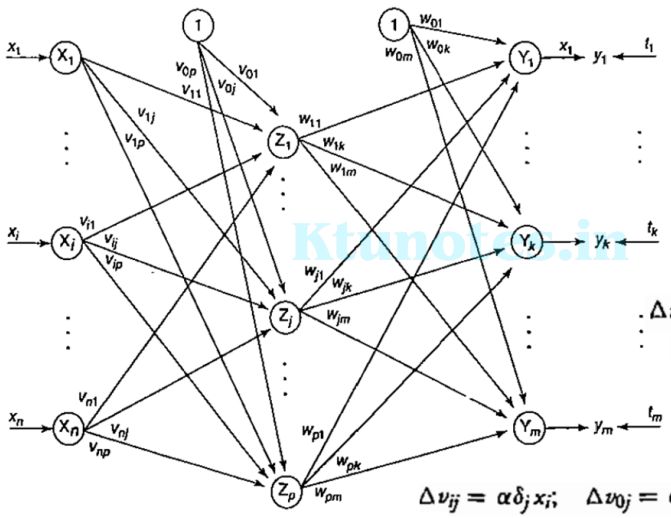
$$\delta_{inj} = \sum_{k=1}^{m} \delta_k w_{jk}$$

The term δ_{inj} gets multiplied with the derivative of $f(z_{inj})$ to calculate the error term:

$$\delta_j = \delta_{inj} f'(z_{inj})$$

The derivative $f'(z_{inj})$ can be calculated. On the basis of the calculated δ_j, update the change in weights and bias: $\Delta v_{ij} = \alpha \delta_j x_i; \quad \Delta v_{0j} = \alpha \delta_j$

Architecture of a back-propagation network.



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

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

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

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

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

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

send δ_k to the hidden layer

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

$$\delta_j = \delta_{inj} f'(z_{inj})$$

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

Weight and bias updation (Phase III):

Step 8: Each output unit $(y_k, k = 1 \text{ to } m)$ 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 $(z_j, j = 1 \text{ to } p)$ 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 epochs reached 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 being changed 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.

BPN-Terminologies

The terminologies used in the flowchart and in the training algorithm using a BPN

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

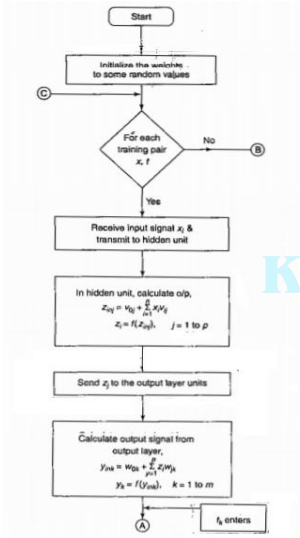
w_{0k} = bias on kth output unit

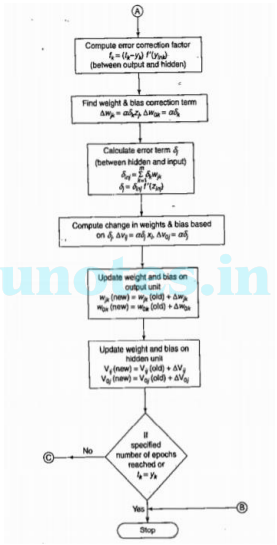
z_j = hidden unit j. The net input to z_j is $z_{inj} = v_{0j} + \sum_{i=1}^{n} x_i v_{ij}$ and the output is $z_j = f(z_{inj})$

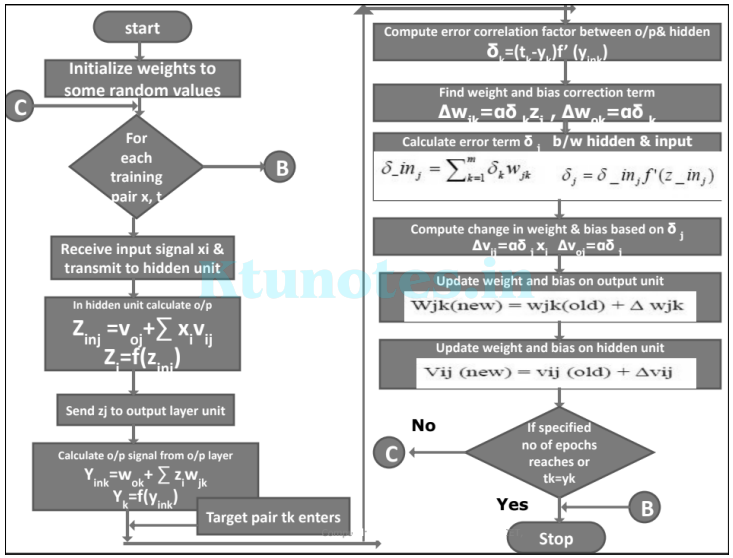
y_k = output unit k. The net input to y_k is $y_{ink} = w_{0k} + \sum_{j=1}^{p} z_j w_{jk}$ and the output is $y_k = f(y_{ink})$

δ_k = error correction weight adjustment for w_{jk} that is due to an error at output unit y_k, which is back-propagated to the hidden units that feed into unit y_k

δ_j = error correction weight adjustment for v_{ij} that is due to the back-propagation of error to the hidden unit z_j.







Back Propagation Network

Weight and bias adjustment

Each **output unit** (yk, *k* = 1 to m) updates the bias and weights:

wjk(new) = wjk(old)+Δwjk ➔ wjk(new) = wjk(old)+$\alpha\delta kzj$

w0k(new) = w0k(old)+Δw0k ➔ w0k(new) = w0k(old)+$\alpha\delta k$

$$\Delta wjk = \alpha\delta kzj; \ \Delta wok = \alpha\delta k;$$

Each **hidden unit** (zj, *j* = 1 to p) updates its bias and weights:

vij(new) = vij(old)+Δvij ➔ vij(new) = vij(old)+$\alpha\delta jxi$

v0j(new) = v0j(old)+Δv0j ➔ v0j(new) = v0j(old)+$\alpha\delta j$

$$\Delta vij = \alpha\ \delta jxi; \quad \Delta voj = \alpha\ \delta j;$$

Important Questions

1. What is supervised learning and how is it different from unsupervised learning?
2. How does learning take place in supervised learning?
3. From a mathematical point of view, what is the process of learning in supervised learning?
4. What is the building block of the perceptron?
5. Does perceptron require supervised learning? If no, what does it require?
6. List the limitations of perceptron.
7. State the activation function used in perceptron network.
8. What is the importance of threshold in perceptron network?
9. Mention the applications of perceptron network.
10. What are feature detectors?
11. With a neat flowchart, explain the training process of perceptron network.
12. What is the significance of error signal in perceptron network?