Neural Network

Fundamentals

Khaleda Akther Papry

Assistant Professor,CSE,DUET

Email: papry.khaleda@duet.ac.bd

Room: 7023 (New academic building)

Course Overview

- Class time : Wednesday 2:00p.m-4:30p.m
- Reference Book: Fundamentals of Neural Networks; Architectures, Algorithms, and Applications – Laurance Fausett
- Marks Distribution (Tentative)
    - Attendance - 10%
    - Mid term exam - 20%
    - Project - 40% (Presentation + Report)
    - Term Final - 30%

Applications of neural networks

- Signal processing
- Control
- Pattern recognition, e.g. handwritten characters or face identification.
- Diagnosis or mapping symptoms to a medical case.
- Speech production and recognition
- Human Emotion Detection
- Business

History

ANN during 1940s to 1960s-

- **1943** − Physiologist, Warren McCulloch, and mathematician, Walter Pitts, in 1943, modeled a simple neural network using electrical circuits in order to describe how neurons in the brain might work.
- **1949** − Donald Hebb's book, *The Organization of Behavior*, put forth the fact that repeated activation of one neuron by another increases its strength each time they are used.
- **1956** − An associative memory network was introduced by Taylor.
- **1958** − A learning method for McCulloch and Pitts neuron model named Perceptron was invented by Rosenblatt.
- **1960** − Bernard Widrow and Marcian Hoff developed models called "ADALINE" and "MADALINE."

History

NN during 1960s to 1980s-

- **1961** − Rosenblatt made an unsuccessful attempt but proposed the "backpropagation" scheme for multilayer networks.
- **1964** − Taylor constructed a winner-take-all circuit with inhibitions among output units.
- 1969 − Multilayer perceptron MLP was invented by Minsky and Papert.
- 1971 − Kohonen developed Associative memories.
- 1976 − Stephen Grossberg and Gail Carpenter developed Adaptive resonance theory.

History

ANN from 1980s till Present−

- 1982 − The major development was Hopfield’s Energy approach.
- 1985 − Boltzmann machine was developed by Ackley, Hinton, and Sejnowski.
- 1986 − Rumelhart, Hinton, and Williams introduced Generalized Delta Rule.
- 1988 − Kosko developed Binary Associative Memory BAM and also gave the concept of Fuzzy Logic in ANN.

Different Types of NN

The types of Neural Networks are:

- Single Layer Perceptron
- Feed Forward Neural Network
- Multilayer Perceptron
- Recurrent Neural Network (RNN)
- Convolutional Neural Network (CNN)
- Radial Basis Function (RBF)
- Sequence to Sequence models
- Modular Neural Network
- Hopfield Network
- Boltzmann Machine
- Kohonen Self organizing Map

Network topology of NN

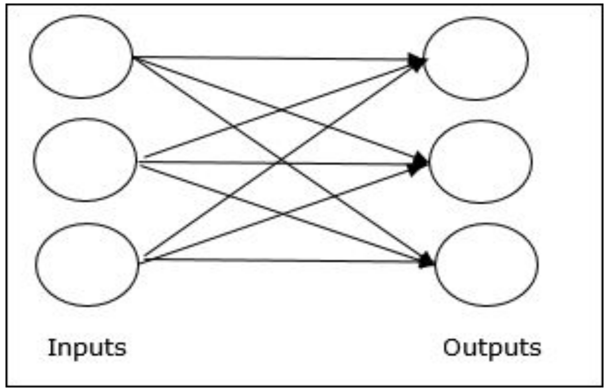
- A network topology is the arrangement of a network along with its nodes and connecting lines.
- According to the topology, ANN can be classified as the following kinds –
  - Feed forward
  - Feed back

Feed forward network

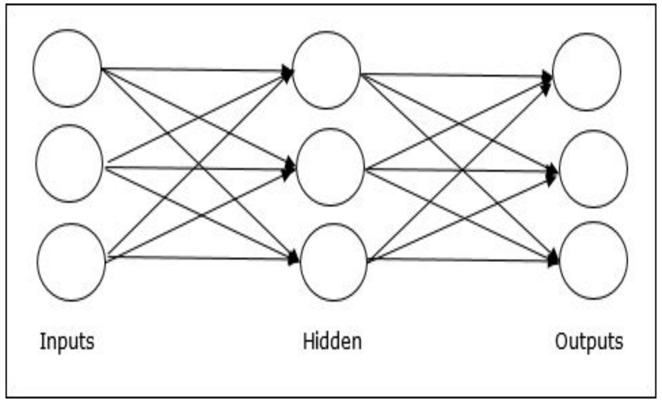
- It is a non-recurrent network having processing units/nodes in layers and all the nodes in a layer are connected with the nodes of the previous layers.
- There is no feedback loop means the signal can only flow in one direction, from input to output.
- It may be divided into the following two types –
    - Single layer
    - Multi layer

Feed forward network

- **Single layer feed forward network** − Only one weighted layer.

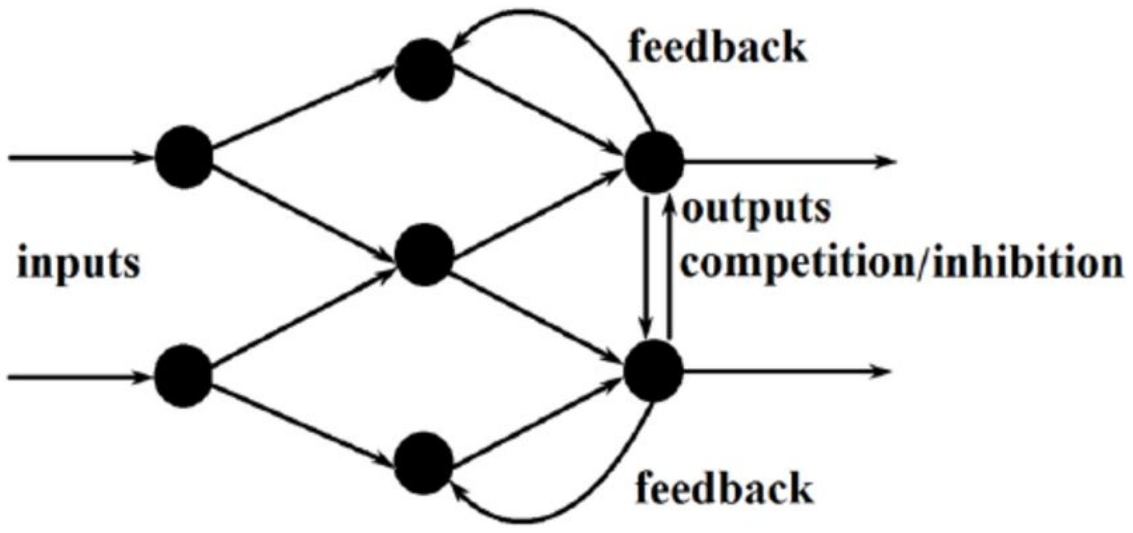


- **Multilayer feed forward network** −more than one weighted layer.



Feedback Neural Network

- Signals can travel in both the directions in Feedback neural networks.
- Feedback neural networks are very powerful and can get very complicated.
- Feedback neural networks are dynamic. The ‘state’ in such network keep changing until they reach an equilibrium point.

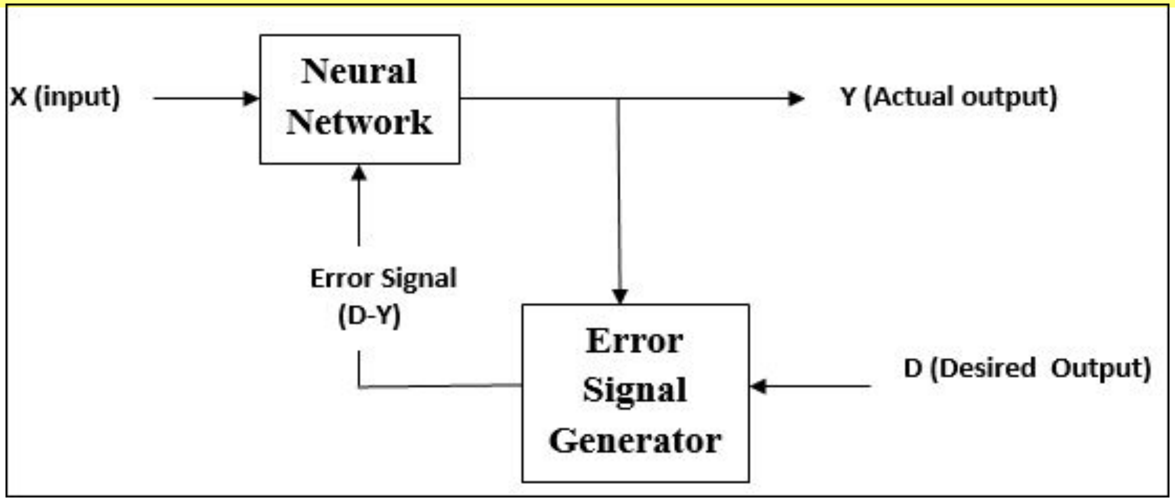


Adjustments of Weights or Learning

- Learning, in NN, is the method of modifying the weights of connections between the neurons of a specified network.
- Learning in ANN can be classified into three categories:
  - Supervised learning
  - Unsupervised learning
  - Reinforcement learning.

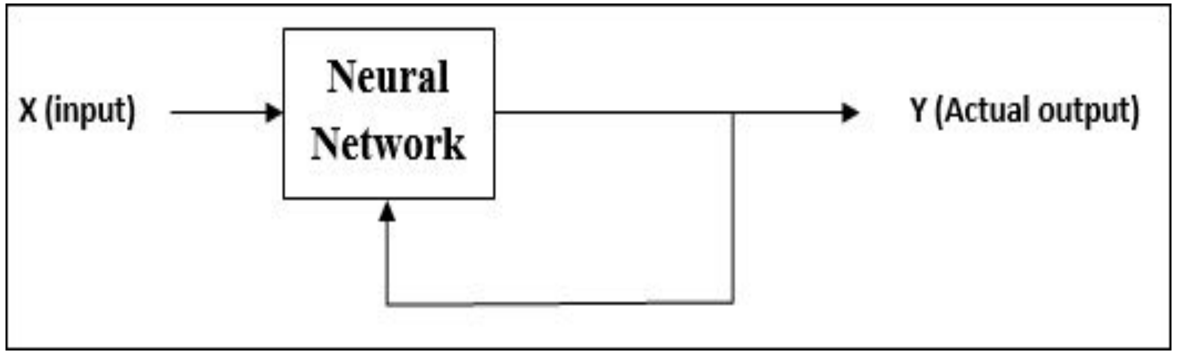
Supervised Learning

- This learning process is dependent.
- During the training of ANN under supervised learning, output vector is compared with the desired output vector.
- An error signal is generated, if there is a difference between the actual output and the desired output vector.
- On the basis of this error signal, the weights are adjusted until the actual output is matched with the desired output.



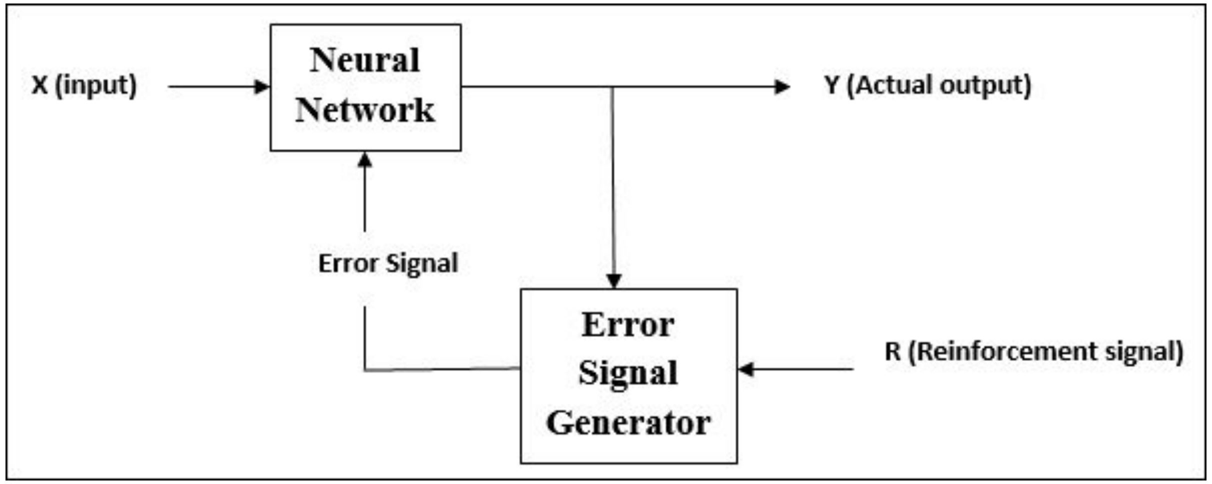
Unsupervised Learning

- This learning process is independent.
- During the training of ANN under unsupervised learning, the input vectors of similar type are combined to form clusters.
- When a new input pattern is applied, then the neural network gives an output response indicating the class to which the input pattern belongs.
- There is no feedback from the environment as to what should be the desired output and if it is correct or incorrect.
- Hence, in this type of learning, the network itself must discover the patterns and features from the input data, and the relation for the input data over the output.

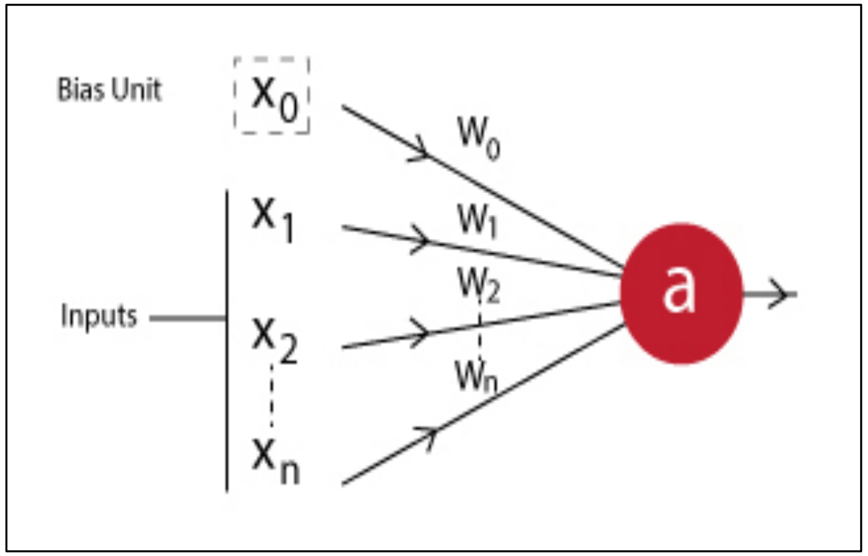


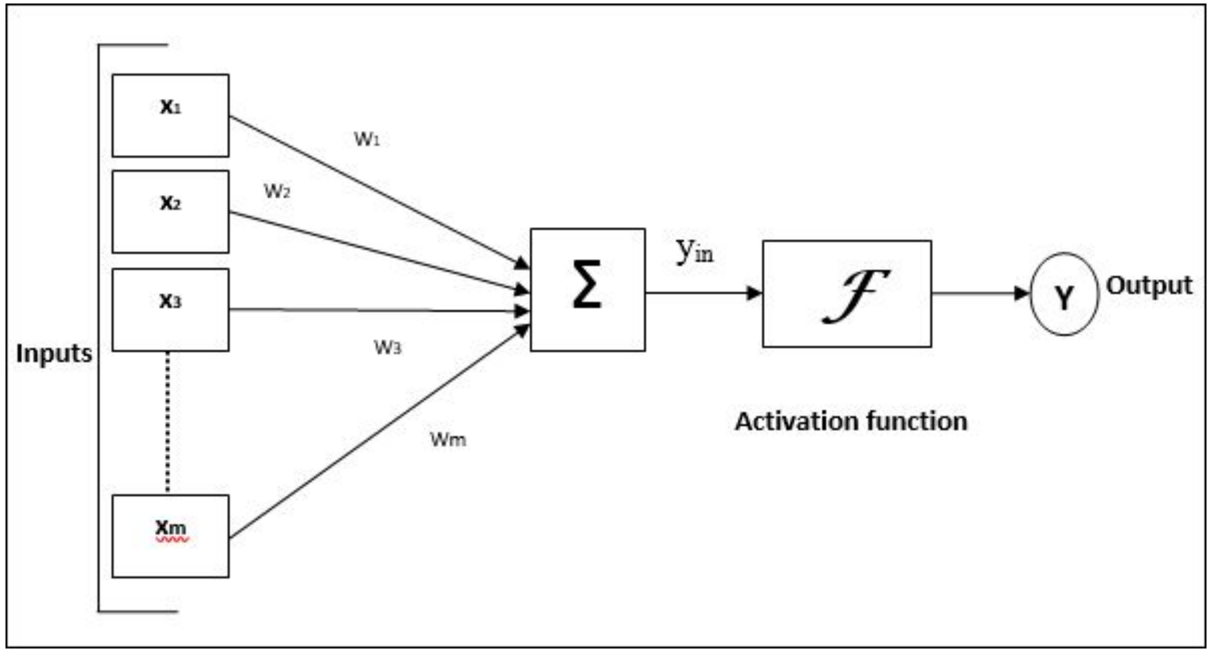
Reinforcement Learning

- This type of learning is used to reinforce or strengthen the network over some critic information. This learning process is similar to supervised learning, however we might have very less information.
- The network receives some feedback from the environment. However, the feedback obtained here is evaluative not instructive, which means after receiving the feedback, the network performs adjustments of the weights to get better critic information in future.



Activation function of NN





$$a = f(\sum_{i=0}^{N} w_i x_i)$$

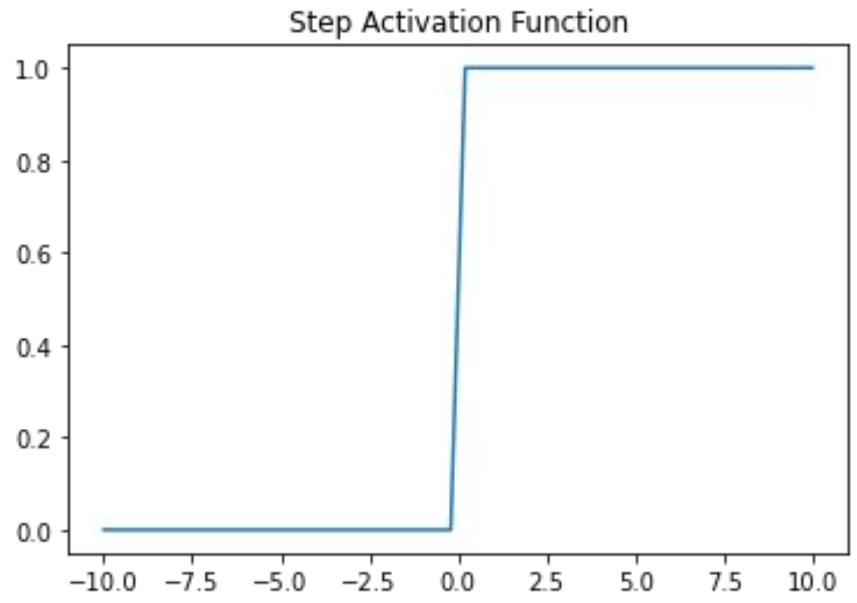
Here **f** is known an **activation function**. It can be a gaussian function, logistic function, hyperbolic function or even a linear function in simple cases.

Common activation function

1. Step Function: Step Function is one of the simplest kinds of activation functions. In this, we select a threshold value and if the value of the weighted sum input say Z is greater than the threshold then the neuron is activated.

The mathematical equation :

$$f(x) = \begin{cases} 0, & for\ x < 0 \\ 1, & for\ x \geq 0 \end{cases}$$



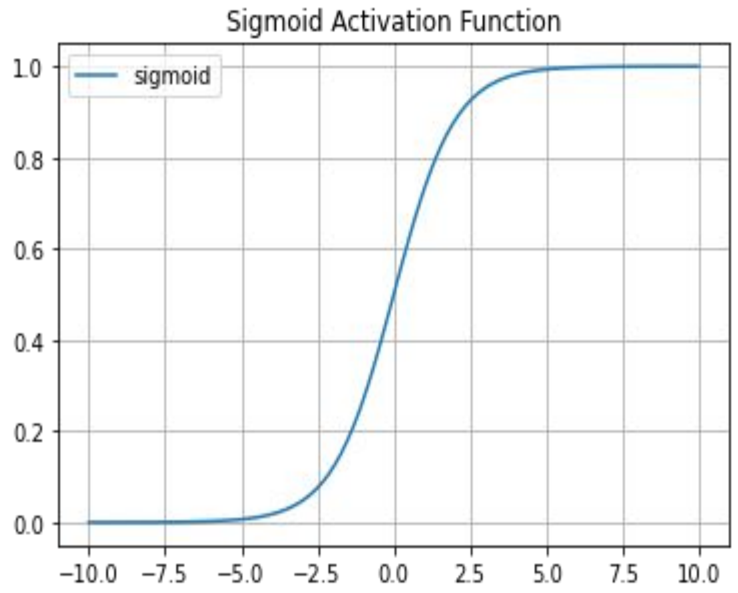
Common activation function

2. Sigmoid Function: Sigmoid Activation Function is one of the widely used activation functions in neural networks. The Curve of the sigmoid function is S-shaped.

- Sigmoid Function transforms the value between the range 0 and 1 (Binary).

- The mathematical equation :

$$f(x) = \frac{1}{1 + \exp(-\sigma x)}.$$

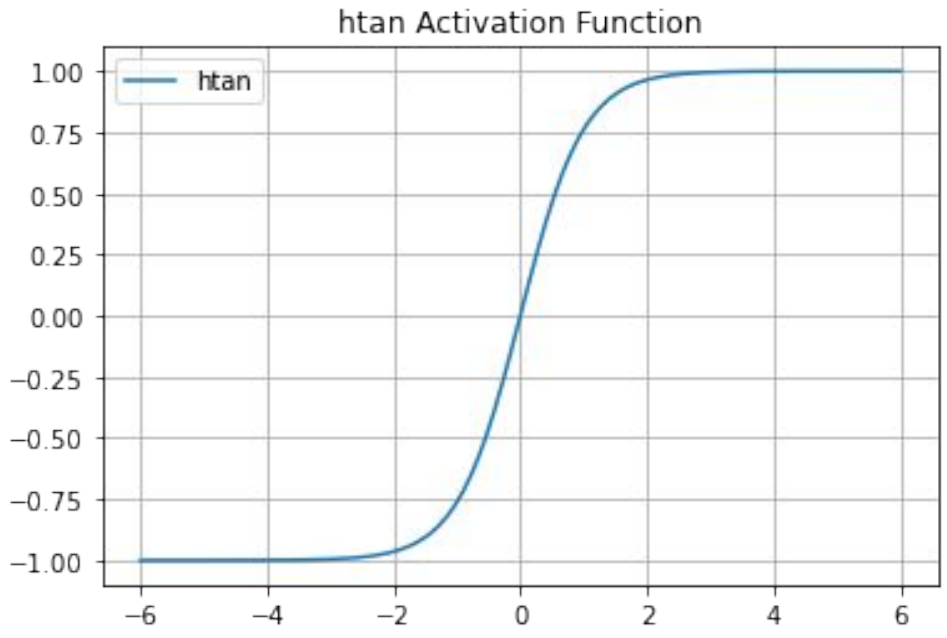


Common activation function

3. Tanh: Hyperbolic Tangent (Tanh) Activation function (Bipolar Sigmoid) is similar to sigmoid function i.e it has a shape like S.The output ranges from -1 to 1.

- The Mathematical equation:

$$h(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)} = \frac{1 - \exp(-2x)}{1 + \exp(-2x)} .$$



Learning and Adaptation

- What Is Learning in ANN?
  - Do and adapt the change in itself as and when there is a change in environment.
  - ANN is a complex adaptive system, which can change its internal structure based on the information passing through it.
- Why Is It important?
  - Being a complex adaptive system, learning in ANN implies that a processing unit is capable of changing its input/output behavior due to the change in environment.
  - When a particular network is constructed, to change the input/output behavior, we need to adjust the weights as the fixed activation function as well as the input/output vector are fixed as well

Neural Network Learning Rules

- During ANN learning, to change the input/output behavior, we need to adjust the weights.
- Hence, a method is required with the help of which the weights can be modified. These methods are called Learning rules
- Some learning rules for the neural network –
  - Hebbian Learning Rule
  - Perceptron Learning Rule
  - Delta Learning Rule (Widrow-Hoff Rule)

Hebbian Learning Rule

- It is a kind of feed-forward, unsupervised learning.
- **Basic Concept** : Connections between two neurons might be strengthened if the neurons fire at the same time and might weaken if they fire at different times.
- **Mathematical Formulation** – According to Hebbian learning rule, following is the formula to increase the weight of connection at every time step:

$$\Delta w_{ji}(t) = \alpha x_i(t) . y_j(t)$$

  - $\Delta w_{ji}(t)$ = increment by which the weight of connection increases at time step t
  - α = the positive and constant learning rate
  - $x_i(t)$ = the input value from pre-synaptic neuron at time step t
  - $y_i(t)$ = the output of pre-synaptic neuron at same time step t

Perceptron Learning Rule

- This rule is an error correcting the supervised learning algorithm of single layer feedforward networks with linear activation function.
- Mathematical Formulation − Suppose we have 'n' number of finite input vectors, x_n, along with its desired/target output vector t_n.
- Now the output 'y' can be calculated, as:

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

Where **θ** is threshold.
The updating of weight can be done in the following two cases −
Case I − when **t ≠ y**, then

w(new)=w(old)+tx

Case II − when **t = y**, then No change in weight

Delta Learning Rule

- Also called Least Mean Square LMS LMS method, to minimize the error over all training patterns.
- 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.
- Mathematical Formulation − To update the synaptic weights, delta rule is given by

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

  - Here Δw_i = weight change for ith 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}

Delta Learning Rule

- The updating of weight can be done in the following two cases –
- **Case-I** – when $t \neq y$, then, $w(new)=w(old)+\Delta w$
- **Case-II** – when $t = y$, then No change in weight

Perceptron Learning Rule

- 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.
- Training Algorithm
- Perceptron network can be trained for single output unit as well as multiple output units.
- Training Algorithm for Single Output Unit
  - Step 1 – Initialize the following to start the training –
    - Weights
    - Bias
    - Learning rate α

    [For easy calculation and simplicity, weights and bias must be set equal to 0 and the learning rate must be set equal to 1]
  - Step 2 – Continue step 3-8 when the stopping condition is not true.

Training Algorithm for Single Output Unit (continued…)

- Step 3 – Continue step 4-6 for every training vector x.
- Step 4 – Activate each input unit x_i (i=1 to n)
- Step 5 – Now obtain the net input with the following relation – $y_{in} = b + \sum_{i}^{n} x_i.\, w_i$
- Here 'b' is bias and 'n' is the total number of input neurons.
- Step 6 – Apply the following activation function to obtain the final output.

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

- Step 7 – Adjust the weight and bias as follows –
- Case 1 – if y ≠ t then, w_i(new)=w_i(old)+$\alpha t x_i$ and b(new)=b(old)+αt
- Case 2 – if y = t then, w_i(new)=w_i(old) and b(new)=b(old)

  Here 'y' is the actual output and 't' is the desired/target output.
- Step 8 – Test for the stopping condition, which would happen when there is no change in weight.

Training Algorithm for Multiple Output Units

Step 5: $$y_{in} = b + \sum_{i}^{n} x_i w_{ij}$$

Step 6: $$f(y_{in}) = \begin{cases} 1 & if\ y_{inj} > \theta \\ 0 & if\ -\theta \leqslant y_{inj} \leqslant \theta \\ -1 & if\ y_{inj} < -\theta \end{cases}$$

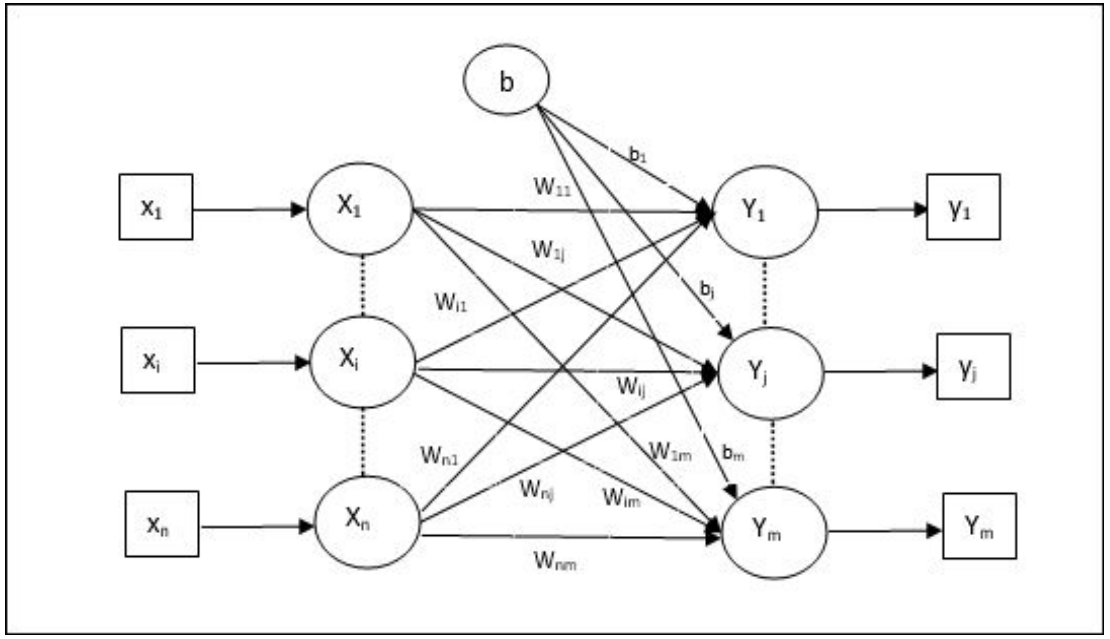
Step 7: Adjust the weight and bias for x = 1 to n and j = 1 to m as follows −

Case 1 – if $y_j \neq t_j$ then, **$w_{ij}(new)=w_{ij}(old)+\alpha t_j x_i$**

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

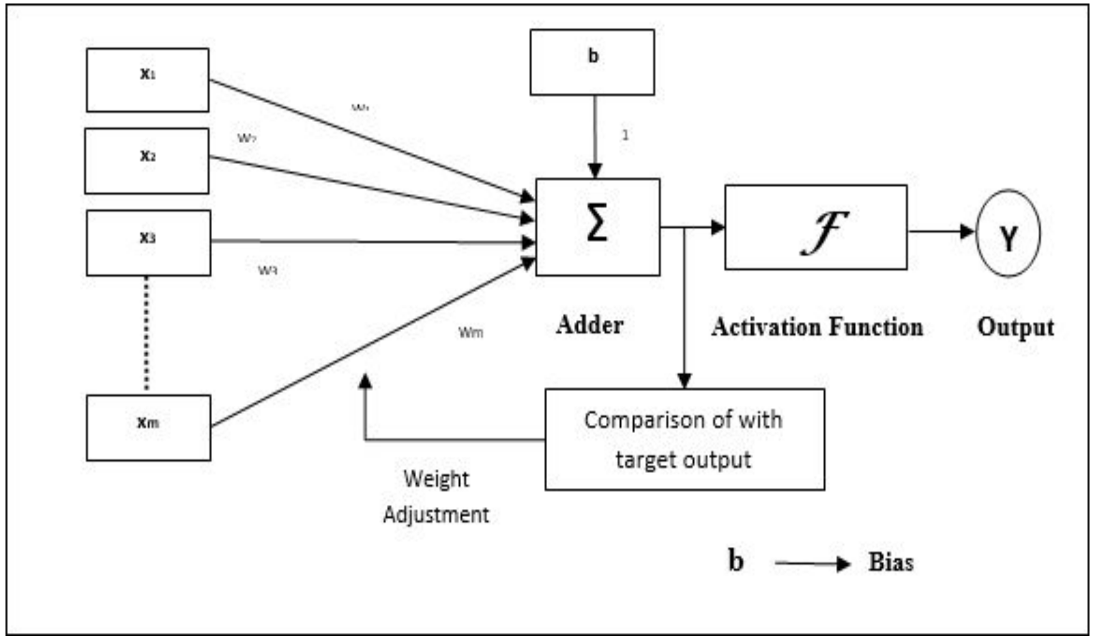
Case 2 – if $y_j = t_j$ then, **$w_{ij}(new)=w_{ij}(old)$**

$b_j(new)=b_j(old)$



Adaptive Linear Neuron (Adaline)

- Adaline is a network having a single linear unit.
- It uses bipolar activation function.
- It uses delta rule for training to minimize the Mean-Squared Error (MSE) between the actual output and the desired/target output.
- The weights and the bias are adjustable.
- Architecture:
  - The basic structure of Adaline is similar to perceptron having an extra feedback loop which the actual outputis compared with the desired/target output.
  - After comparison on the basis of training algorithm, the weights and bias will be updated.



Training Algorithm (Adaline)

- Step 1 − Initialize the following to start the training −
  - Weights
  - Bias
  - Learning rate α

[For easy calculation and simplicity, weights and bias must be set equal to 0 and the learning rate must be set equal to 1]

- Step 2 − Continue step 3-8 when the stopping condition is not true.
- Step 3 − Continue step 4-6 for every training vector x.
- Step 4 − Activate each input unit x_i (i=1 to n)
- Step 5 − Now obtain the net input with the following relation −

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

Here ‘b’ is bias and ‘n’ is the total number of input neurons.

Training Algorithm (Adaline)

- Step 6 – Apply the following activation function to obtain the final output.

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

- Step 7 – Adjust the weight and bias as follows –
  - **Case 1 –** if **$y \neq t$** then, **$w_i(new)=w_i(old)+\alpha(t-y_{in})x_i$** **$b(new)=b(old)+\alpha(t-y_{in})$**
  - **Case 2 –** if **$y = t$** then, **$w_{ij}(new)=w_{ij}(old)$**

    $b_j(new)=b_j(old)$
  - Here 'y' is the actual output and 't' is the desired/target output and $(t-y_{in})$ is the computed error
- Step 8 – Test for the stopping condition, which would happen when there is no change in weight.

THE END