



## **Fundamentals of Neural Networks**

### **Soft Computing**

*Neural network, topics : Introduction, biological neuron model, artificial neuron model, neuron equation. Artificial neuron : basic elements, activation and threshold function, piecewise linear and sigmoidal function. Neural network architectures : single layer feed-forward network, multi layer feed-forward network, recurrent networks. Learning methods in neural networks : unsupervised Learning - Hebbian learning, competitive learning; Supervised learning - stochastic learning, gradient descent learning; Reinforced learning. Taxonomy of neural network systems : popular neural network systems, classification of neural network systems as per learning methods and architecture. Single-layer NN system : single layer perceptron, learning algorithm for training perceptron, linearly separable task, XOR problem, ADaptive LINear Element (ADALINE) - architecture, and training. Applications of neural networks: clustering, classification, pattern recognition, function approximation, prediction systems.*

# Fundamentals of Neural Networks

## Soft Computing

### Topics

(Lectures 07, 08, 09, 10, 11, 12, 13, 14 8 hours)

	Slides
<b>1. Introduction</b>	03-12
Why neural network ?, Research History, Biological Neuron model, Artificial Neuron model, Notations, Neuron equation.	
<b>2. Model of Artificial Neuron</b>	13-19
Artificial neuron - basic elements, Activation functions – Threshold function, Piecewise linear function, Sigmoidal function, Example.	
<b>3. Neural Network Architectures</b>	20-23
Single layer Feed-forward network, Multi layer Feed-forward network, Recurrent networks.	
<b>4. Learning Methods in Neural Networks</b>	24-29
Learning algorithms: Unsupervised Learning - Hebbian Learning, Competitive learning; Supervised Learning : Stochastic learning, Gradient descent learning; Reinforced Learning;	
<b>5. Taxonomy Of Neural Network Systems</b>	30-32
Popular neural network systems; Classification of neural network systems with respect to learning methods and architecture types.	
<b>6. Single-Layer NN System</b>	32-39
Single layer perceptron : Learning algorithm for training Perceptron, Linearly separable task, XOR Problem; ADaptive LINEar Element (ADALINE) : Architecture, Training.	
<b>7. Applications of Neural Networks</b>	39
Clustering, Classification / pattern recognition, Function approximation, Prediction systems.	
<b>8. References :</b>	40

## Fundamentals of Neural Networks

### What is Neural Net ?

- A neural net is an artificial representation of the human brain that tries to simulate its learning process. An artificial neural network (ANN) is often called a "Neural Network" or simply Neural Net (NN).
- Traditionally, the word neural network is referred to a network of biological neurons in the nervous system that process and transmit information.
- Artificial neural network is an interconnected group of artificial neurons that uses a mathematical model or computational model for information processing based on a connectionist approach to computation.
- The artificial neural networks are made of interconnecting artificial neurons which may share some properties of biological neural networks.
- Artificial Neural network is a network of simple processing elements (neurons) which can exhibit complex global behavior, determined by the connections between the processing elements and element parameters.

## 1. Introduction

Neural Computers mimic certain processing capabilities of the human brain.

- Neural Computing is an **information processing paradigm**, inspired by biological system, composed of a large number of highly interconnected processing elements (neurons) working in unison to solve specific problems.
- Artificial Neural Networks (ANNs), like people, **learn by example**.
- An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process.
- Learning in biological systems involves adjustments to the synaptic connections that exist between the neurons. This is true of ANNs as well.

## 1 Why Neural Network

Neural Networks follow a different paradigm for computing.

- The conventional computers are good for - fast arithmetic and does what programmer programs, ask them to do.
- The conventional computers are not so good for - interacting with noisy data or data from the environment, massive parallelism, fault tolerance, and adapting to circumstances.
- The neural network systems help where we can not formulate an algorithmic solution or where we can get lots of examples of the behavior we require.
- Neural Networks follow different paradigm for computing.

The von Neumann machines are based on the processing/memory abstraction of human information processing.

The neural networks are based on the parallel architecture of biological brains.

- Neural networks are a form of multiprocessor computer system, with
  - simple processing elements ,
  - a high degree of interconnection,
  - simple scalar messages, and
  - adaptive interaction between elements.

## 2 Research History

The history is relevant because for nearly two decades the future of Neural network remained uncertain.

McCulloch and Pitts (1943) are generally recognized as the designers of the **first neural network**. They combined many simple processing units together that could lead to an overall increase in computational power. They suggested many ideas like : a neuron has a threshold level and once that level is reached the neuron fires. It is still the fundamental way in which ANNs operate. The McCulloch and Pitts's network had a fixed set of weights.

Hebb (1949) developed the **first learning rule**, that is if two neurons are active at the same time then the strength between them should be increased.

In the 1950 and 60's, many researchers (Block, Minsky, Papert, and Rosenblatt worked on **perceptron**. The neural network model could be proved to converge to the correct weights, that will solve the problem. The **weight adjustment** (learning algorithm) used in the perceptron was found more powerful than the learning rules used by Hebb. The perceptron caused great excitement. It was thought to produce programs that could think.

Minsky & Papert (1969) showed that perceptron could not learn those functions which are not linearly separable.

The neural networks **research declined** throughout the 1970 and until mid 80's because the perceptron could not learn certain important functions.

Neural network **regained importance** in 1985-86. The researchers, Parker and LeCun discovered a learning algorithm for **multi-layer networks called back propagation** that could solve problems that were not linearly separable.

### Biological Neuron Model

The human brain consists of a large number, more than a billion of neural cells that process information. Each cell works like a simple processor. The massive interaction between all cells and their parallel processing only makes the brain's abilities possible.

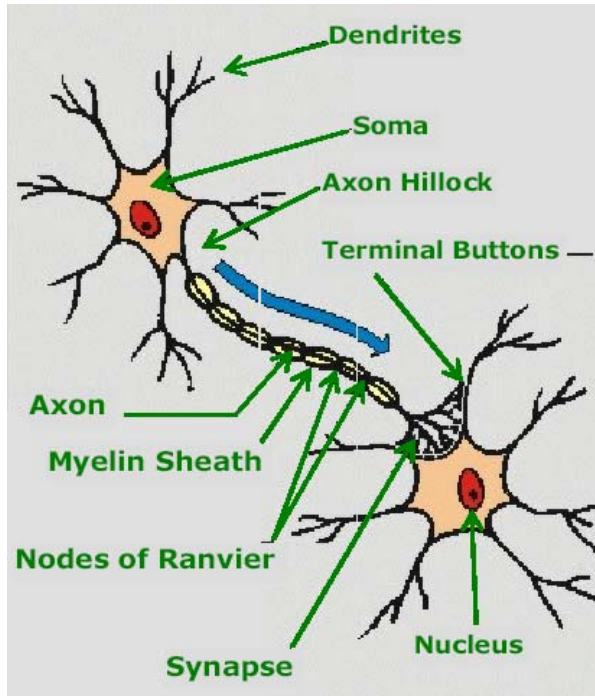


Fig. Structure of Neuron

**Dendrites** are branching fibers that extend from the cell body or soma.

**Soma or cell body** of a neuron contains the nucleus and other structures, support chemical processing and production of neurotransmitters.

**Axon** is a singular fiber carries information away from the soma to the synaptic sites of other neurons (dendrites and somas), muscles, or glands.

**Axon hillock** is the site of summation for incoming information. At any moment, the collective influence of all neurons that conduct impulses to a given neuron will determine whether or not an action potential will be initiated at the

axon hillock and propagated along the axon.

**Myelin Sheath** consists of fat-containing cells that insulate the axon from electrical activity. This insulation acts to increase the rate of transmission of signals. A gap exists between each myelin sheath cell along the axon. Since fat inhibits the propagation of electricity, the signals jump from one gap to the next.

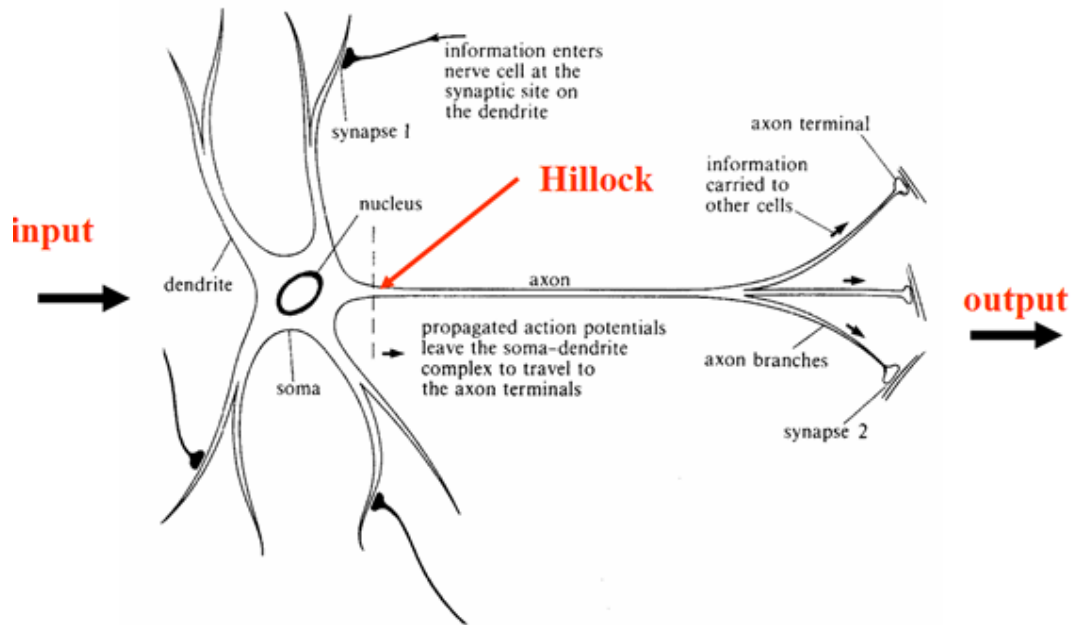
**Nodes of Ranvier** are the gaps (about 1  $\mu\text{m}$ ) between myelin sheath cells long axons are Since fat serves as a good insulator, the myelin sheaths speed the rate of transmission of an electrical impulse along the axon.

**Synapse** is the point of connection between two neurons or a neuron and a muscle or a gland. Electrochemical communication between neurons takes place at these junctions.

**Terminal Buttons** of a neuron are the small knobs at the end of an axon that release chemicals called neurotransmitters.

## Information flow in a Neural Cell

The input /output and the propagation of information are shown below.



**Fig. Structure of a neural cell in the human brain**

- Dendrites receive activation from other neurons.
- Soma processes the incoming activations and converts them into output activations.
- Axons act as transmission lines to send activation to other neurons.
- Synapses the junctions allow signal transmission between the axons and dendrites.
- The process of transmission is by diffusion of chemicals called neuro-transmitters.

McCulloch-Pitts introduced a simplified model of this real neurons.

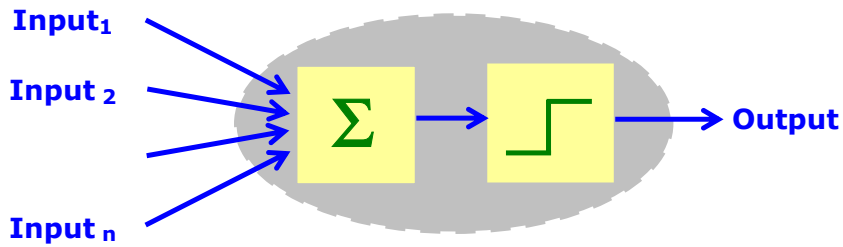


## 4 Artificial Neuron Model

An artificial neuron is a mathematical function conceived as a simple model of a real (biological) neuron.

### ● The McCulloch-Pitts Neuron

This is a simplified model of real neurons, known as a Threshold Logic Unit.



- A set of input connections brings in activations from other neurons.
- A processing unit sums the inputs, and then applies a non-linear activation function (i.e. squashing / transfer / threshold function).
- An output line transmits the result to other neurons.

In other words ,

- The input to a neuron arrives in the form of signals.
- The signals build up in the cell.
- Finally the cell discharges (cell fires) through the output .
- The cell can start building up signals again.

## 5 Notations

Recaps : Scalar, Vectors, Matrices and Functions

- **Scalar** : The number  $x_i$  can be added up to give a scalar number.

$$s = x_1 + x_2 + x_3 + \dots + x_n = \sum_{i=1}^n x_i$$

- **Vectors** : An ordered sets of related numbers. Row Vectors  $(1 \times n)$

$$X = (x_1, x_2, x_3, \dots, x_n), \quad Y = (y_1, y_2, y_3, \dots, y_n)$$

**Add** : Two vectors of same length added to give another vector.

$$Z = X + Y = (x_1 + y_1, x_2 + y_2, \dots, x_n + y_n)$$

**Multiply**: Two vectors of same length multiplied to give a scalar.

$$p = X \cdot Y = x_1 y_1 + x_2 y_2 + \dots + x_n y_n = \sum_{i=1}^n x_i y_i$$

**Matrices :**  $m \times n$  matrix , row no =  $m$  , column no =  $n$

$$W = \begin{pmatrix} W_{11} & W_{11} & \dots & \dots & W_{1n} \\ W_{21} & W_{21} & \dots & \dots & W_{2n} \\ \cdot & \cdot & \dots & \dots & \cdot \\ \cdot & \cdot & \dots & \dots & \cdot \\ W_{m1} & W_{11} & \dots & \dots & W_{mn} \end{pmatrix}$$

**Add or Subtract :** Matrices of the same size are added or subtracted component by component.  $A + B = C$  ,  $c_{ij} = a_{ij} + b_{ij}$

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} + \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} c_{11} = a_{11} + b_{11} & c_{12} = a_{12} + b_{12} \\ c_{21} = a_{21} + b_{21} & c_{22} = a_{22} + b_{22} \end{pmatrix}$$

**Multiply :** matrix **A** multiplied by matrix **B** gives matrix **C**.  
 $(m \times n)$   $(n \times p)$   $(m \times p)$

elements  $c_{ij} = \sum_{k=1}^n a_{ik} b_{kj}$

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \times \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix}$$

$$c_{11} = (a_{11} \times b_{11}) + (a_{12} \times b_{21})$$

$$c_{12} = (a_{11} \times b_{12}) + (a_{12} \times b_{22})$$

$$c_{21} = (a_{21} \times b_{11}) + (a_{22} \times b_{21})$$

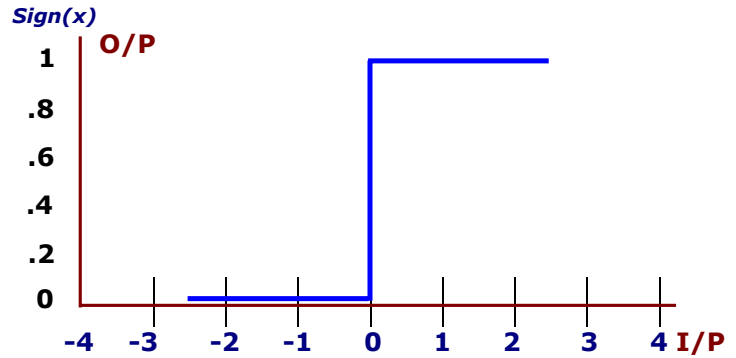
$$c_{22} = (a_{21} \times b_{12}) + (a_{22} \times b_{22})$$

## 1.6 Functions

The Function  $y = f(x)$  describes a relationship, an input-output mapping, from  $x$  to  $y$ .

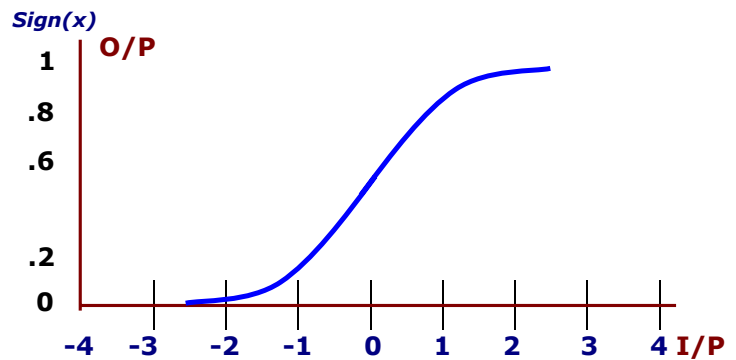
- **Threshold or Sign function :**  $\text{sgn}(x)$  defined as

$$\text{sgn}(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$



- **Threshold or Sign function :**  $\text{sigmoid}(x)$  defined as a smoothed (differentiable) form of the threshold function

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$



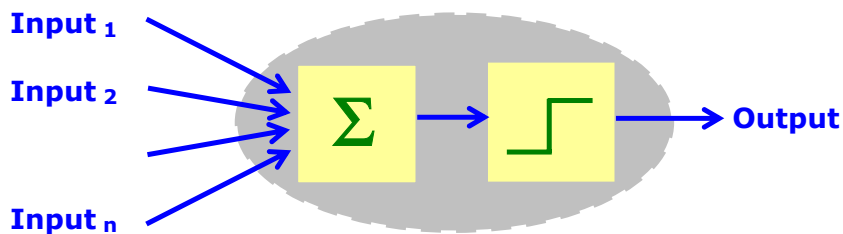
## 2. Model of Artificial Neuron

A very simplified model of real neurons is known as a Threshold Logic Unit (TLU). The model is said to have :

- A set of synapses (connections) brings in activations from other neurons.
- A processing unit sums the inputs, and then applies a non-linear activation function (i.e. squashing / transfer / threshold function).
- An output line transmits the result to other neurons.

### 2.1 McCulloch-Pitts (M-P) Neuron Equation

McCulloch-Pitts neuron is a simplified model of real biological neuron.



**Simplified Model of Real Neuron  
(Threshold Logic Unit)**

The equation for the output of a McCulloch-Pitts neuron as a function of **1** to **n** inputs is written as

$$\text{Output} = \text{sgn} \left( \sum_{i=1}^n \text{Input } i - \Phi \right)$$

where  $\Phi$  is the neuron's activation threshold.

$$\text{If } \sum_{i=1}^n \text{Input } i \geq \Phi \text{ then Output} = 1$$

$$\text{If } \sum_{i=1}^n \text{Input } i < \Phi \text{ then Output} = 0$$

In this McCulloch-Pitts neuron model, the missing features are :

- Non-binary input and output,
- Non-linear summation,
- Smooth thresholding,
- Stochastic, and
- Temporal information processing.

## Artificial Neuron - Basic Elements

Neuron consists of three basic components - weights, thresholds, and a single activation function.

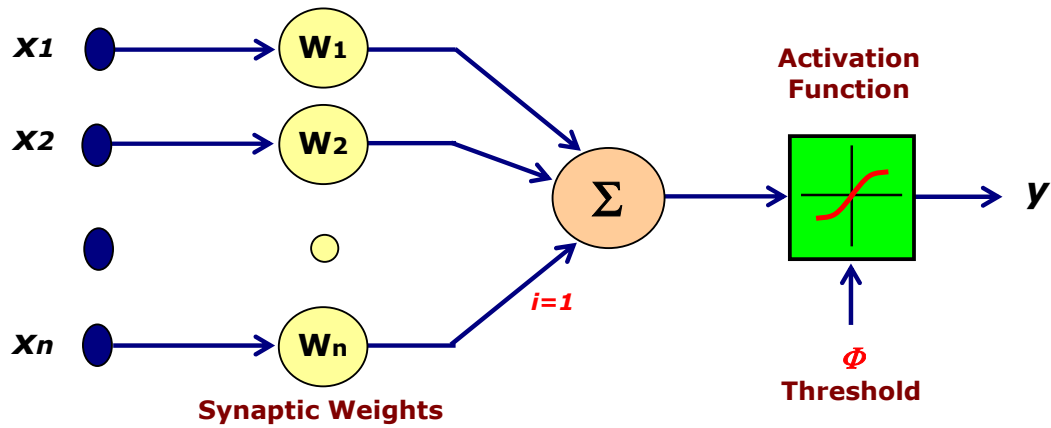


Fig Basic Elements of an Artificial Linear Neuron

### ■ Weighting Factors $w$

The values  $w_1, w_2, \dots, w_n$  are weights to determine the strength of input vector  $\mathbf{X} = [x_1, x_2, \dots, x_n]^T$ . Each input is multiplied by the associated weight of the neuron connection  $\mathbf{X}^T \mathbf{W}$ . The +ve weight excites and the -ve weight inhibits the node output.

$$\mathbf{I} = \mathbf{X}^T \cdot \mathbf{W} = x_1 w_1 + x_2 w_2 + \dots + x_n w_n = \sum_{i=1}^n x_i w_i$$

### ■ Threshold $\Phi$

The node's internal threshold  $\Phi$  is the magnitude offset. It affects the activation of the node output  $\mathbf{y}$  as:

$$\mathbf{Y} = \mathbf{f}(\mathbf{I}) = \mathbf{f} \left\{ \sum_{i=1}^n x_i w_i - \Phi_k \right\}$$

To generate the final output  $\mathbf{Y}$ , the sum is passed on to a non-linear filter  $\mathbf{f}$  called Activation Function or Transfer function or Squash function which releases the output  $\mathbf{Y}$ .

### ■ Threshold for a Neuron

In practice, neurons generally do not fire (produce an output) unless their total input goes above a threshold value.

The total input for each neuron is the sum of the weighted inputs to the neuron minus its threshold value. This is then passed through the sigmoid function. The equation for the transition in a neuron is :

$$a = 1/(1 + \exp(-x)) \quad \text{where}$$

$$x = \sum_i a_i w_i - Q$$

**a** is the activation for the neuron

**a<sub>i</sub>** is the activation for neuron **i**

**w<sub>i</sub>** is the weight

**Q** is the threshold subtracted

### ■ Activation Function

An activation function **f** performs a mathematical operation on the signal output. The most common activation functions are:

- Linear Function,
- Piecewise Linear Function,
- Tangent hyperbolic function
- Threshold Function,
- Sigmoidal (S shaped) function,

The activation functions are chosen depending upon the type of problem to be solved by the network.

## 2 Activation Functions **f** - Types

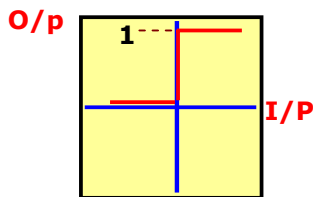
Over the years, researches tried several functions to convert the input into an outputs. The most commonly used functions are described below.

- **I/P** Horizontal axis shows sum of inputs .
- **O/P** Vertical axis shows the value the function produces ie output.
- All functions **f** are designed to produce values between **0** and **1**.

### ● Threshold Function

A threshold (hard-limiter) activation function is either a binary type or a bipolar type as shown below.

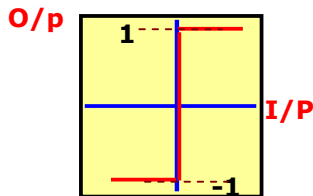
**binary threshold** Output of a binary threshold function produces :



- 1** if the weighted sum of the inputs is +ve,
- 0** if the weighted sum of the inputs is -ve.

$$Y = f(I) = \begin{cases} 1 & \text{if } I \geq 0 \\ 0 & \text{if } I < 0 \end{cases}$$

**bipolar threshold** Output of a bipolar threshold function produces :



- 1** if the weighted sum of the inputs is +ve,
- 1** if the weighted sum of the inputs is -ve.

$$Y = f(I) = \begin{cases} 1 & \text{if } I \geq 0 \\ -1 & \text{if } I < 0 \end{cases}$$

Neuron with hard limiter activation function is called McCulloch-Pitts model.



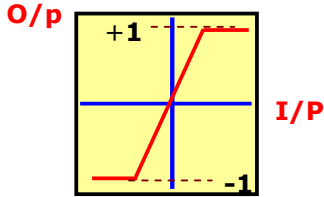
**Piecewise Linear Function**

This activation function is also called saturating linear function and can have either a binary or bipolar range for the saturation limits of the output. The mathematical model for a symmetric saturation function is described below.

**Piecewise Linear**

This is a sloping function that produces :

- 1 for a -ve weighted sum of inputs,
- 1 for a +ve weighted sum of inputs.



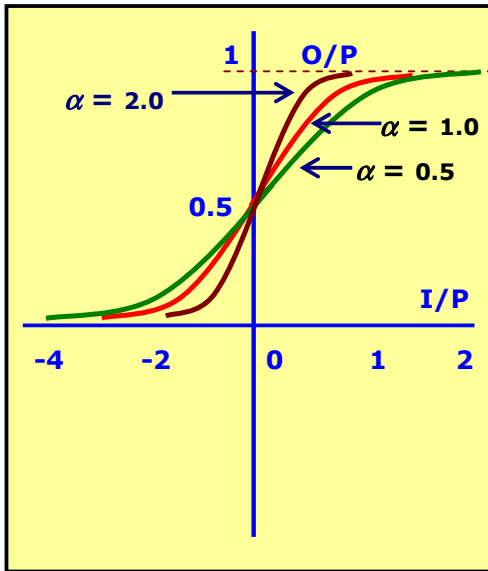
$\propto I$  proportional to input for values between +1 and -1 weighted sum,

$$Y = f(I) = \begin{cases} 1 & \text{if } I \geq 1 \\ I & \text{if } -1 \geq I \geq -1 \\ -1 & \text{if } I < -1 \end{cases}$$

**Sigmoidal Function** (S-shape function)

The nonlinear curved S-shape function is called the sigmoid function. This is most common type of activation used to construct the neural networks. It is mathematically well behaved, differentiable and strictly increasing function.

**Sigmoidal function**



A sigmoidal transfer function can be written in the form:

$$Y = f(I) = \frac{1}{1 + e^{-\alpha I}}, \quad 0 \leq f(I) \leq 1$$

$$= 1/(1 + \exp(-\alpha I)), \quad 0 \leq f(I) \leq 1$$

This is explained as

≈ 0 for large -ve input values,

1 for large +ve values, with

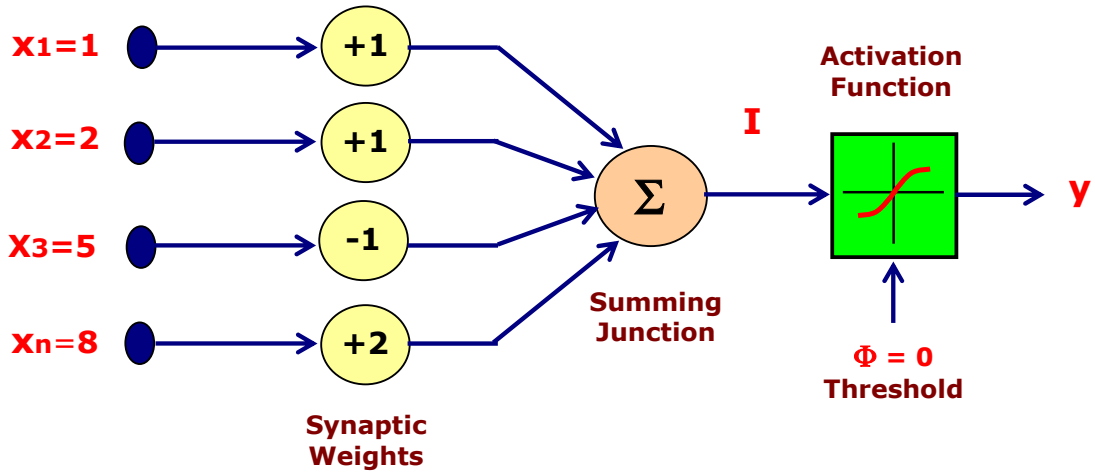
a smooth transition between the two.

α is slope parameter also called shape parameter; symbol the λ is also used to represented this parameter.

The sigmoidal function is achieved using exponential equation. By varying α different shapes of the function can be obtained which adjusts the abruptness of the function as it changes between the two asymptotic values.

**Example :**

The neuron shown consists of four inputs with the weights.



**Fig Neuron Structure of Example**

The output **I** of the network, prior to the activation function stage, is

$$\begin{aligned}
 \mathbf{I} &= \mathbf{X}^T \cdot \mathbf{W} = \begin{bmatrix} 1 & 2 & 5 & 8 \end{bmatrix} \cdot \begin{bmatrix} +1 \\ +1 \\ -1 \\ +2 \end{bmatrix} = 14 \\
 &= (1 \times 1) + (2 \times 1) + (5 \times -1) + (8 \times 2) = 14
 \end{aligned}$$

With a binary activation function the outputs of the neuron is:

$$\mathbf{y} \text{ (threshold)} = 1;$$

### 3. Neural Network Architectures

An Artificial Neural Network (ANN) is a data processing system, consisting large number of simple highly interconnected processing elements as artificial neuron in a network structure that can be represented using a directed graph  $G$ , an ordered 2-tuple  $(V, E)$ , consisting a set  $V$  of vertices and a set  $E$  of edges.

- The vertices may represent neurons (input/output) and
- The edges may represent synaptic links labeled by the weights attached.

Example :

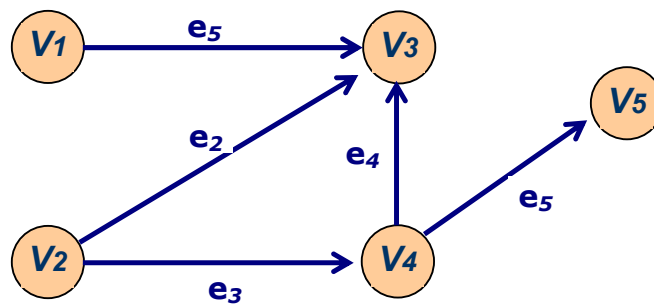


Fig. Directed Graph

Vertices  $V = \{ v_1, v_2, v_3, v_4, v_5 \}$

Edges  $E = \{ e_1, e_2, e_3, e_4, e_5 \}$

### 1 Single Layer Feed-forward Network

The Single Layer Feed-forward Network consists of a single layer of weights, where the inputs are directly connected to the outputs, via a series of weights. The synaptic links carrying weights connect every input to every output, but not other way. This way it is considered a network of **feed-forward** type. The sum of the products of the weights and the inputs is calculated in each neuron node, and if the value is above some threshold (typically **0**) the neuron fires and takes the activated value (typically **1**); otherwise it takes the deactivated value (typically **-1**).

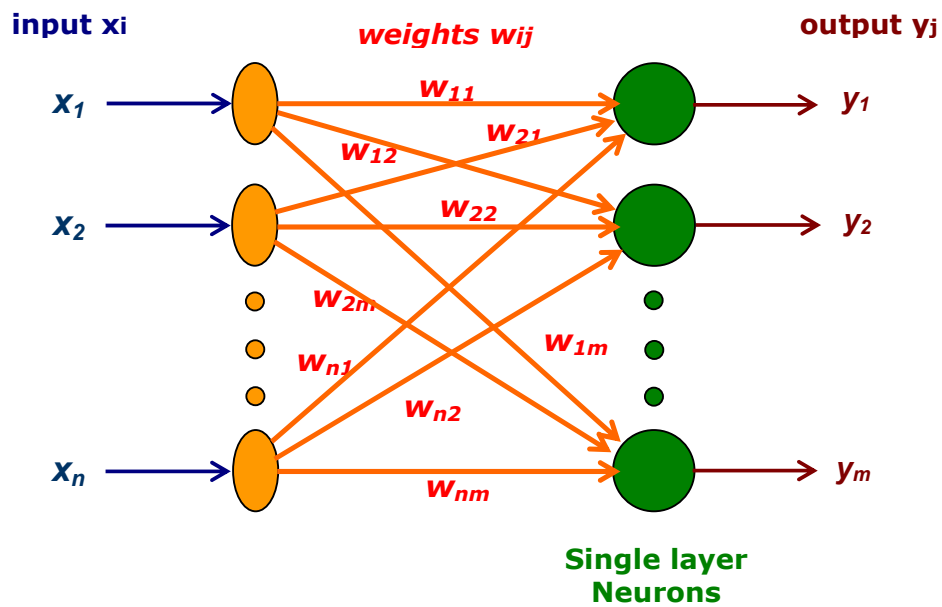


Fig. Single Layer Feed-forward Network

## 2 Multi Layer Feed-forward Network

The name suggests, it consists of multiple layers. The architecture of this class of network, besides having the input and the output layers, also have one or more intermediary layers called **hidden layers**. The computational units of the hidden layer are known as **hidden neurons**.

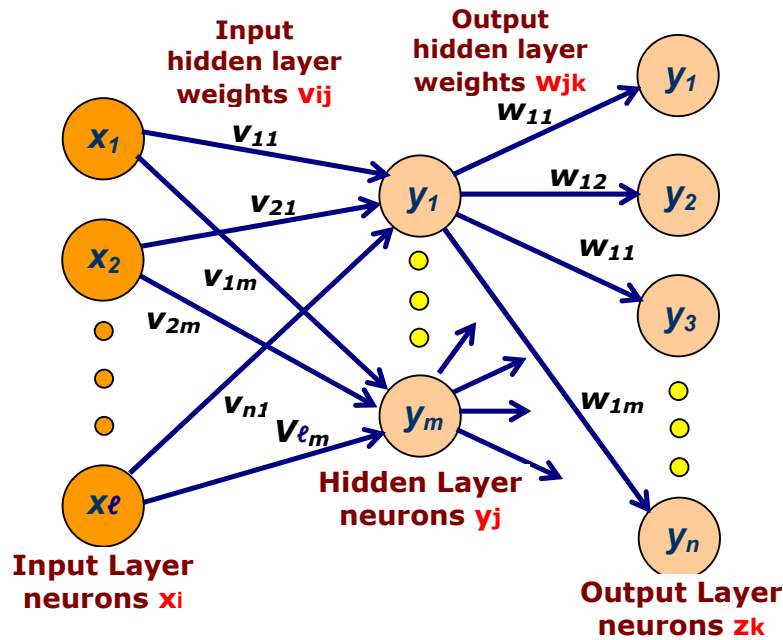


Fig. Multilayer feed-forward network in  $(\ell - m - n)$  configuration.

- The hidden layer does intermediate computation before directing the input to output layer.
- The input layer neurons are linked to the hidden layer neurons; the weights on these links are referred to as **input-hidden layer weights**.
- The hidden layer neurons and the corresponding weights are referred to as **output-hidden layer weights**.
- A multi-layer feed-forward network with  $\ell$  input neurons,  $m_1$  neurons in the first hidden layers,  $m_2$  neurons in the second hidden layers, and  $n$  output neurons in the output layers is written as  $(\ell - m_1 - m_2 - n)$ .

The Fig. above illustrates a multilayer feed-forward network with a configuration  $(\ell - m - n)$ .

### Recurrent Networks

The Recurrent Networks differ from feed-forward architecture. A Recurrent network has at least one feed back loop.

Example :

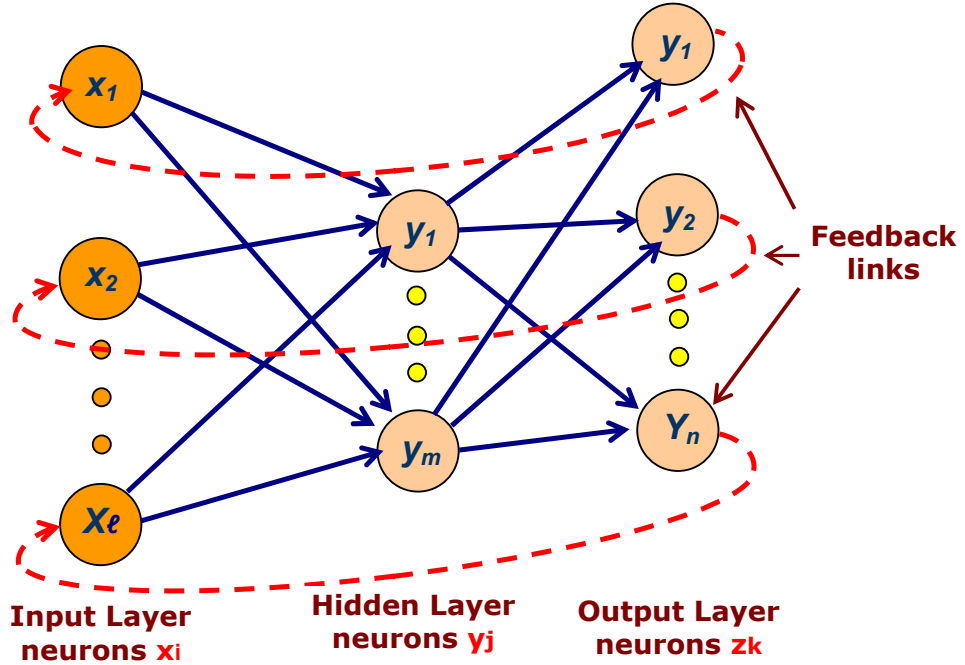


Fig Recurrent Neural Network

There could be neurons with self-feedback links; that is the output of a neuron is fed back into it self as input.

#### 4. Learning Methods in Neural Networks

The learning methods in neural networks are classified into three basic types :

- Supervised Learning,
- Unsupervised Learning and
- Reinforced Learning

These three types are classified based on :

- presence or absence of **teacher** and
- the information provided for the system to learn.

These are further categorized, based on the **rules** used, as

- Hebbian,
- Gradient descent,
- Competitive and
- Stochastic learning.



### Classification of Learning Algorithms

Fig. below indicate the hierarchical representation of the algorithms mentioned in the previous slide. These algorithms are explained in subsequent slides.

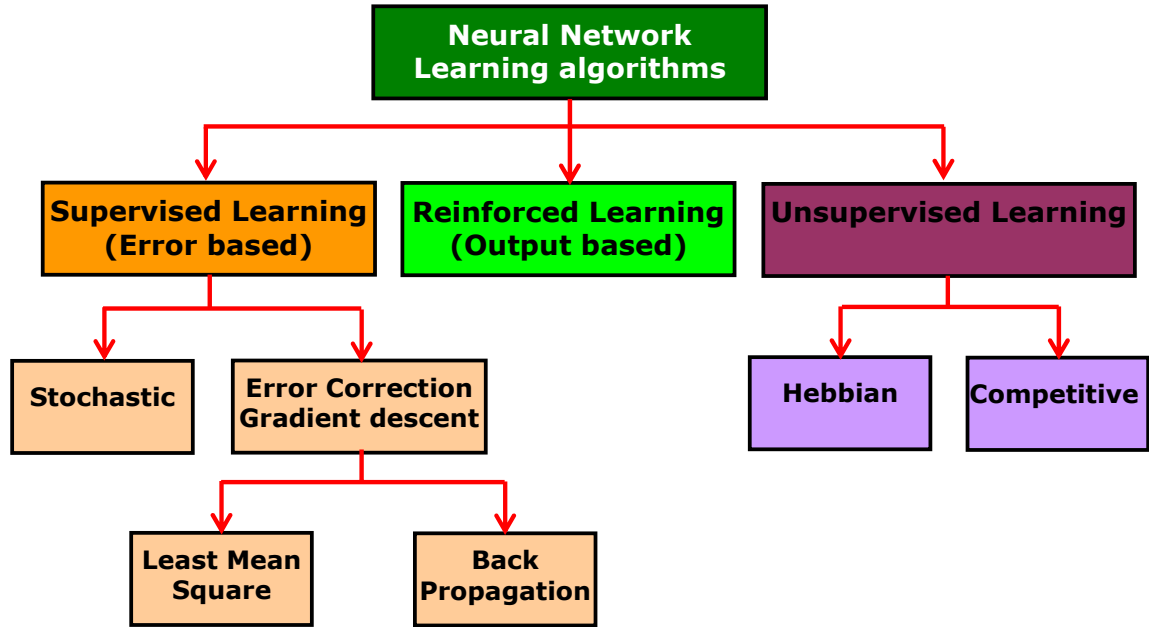


Fig. Classification of learning algorithms

- **Supervised Learning**

- A teacher is present during learning process and presents expected output.
- Every input pattern is used to train the network.
- Learning process is based on comparison, between network's computed output and the correct expected output, generating "error".
- The "error" generated is used to change network parameters that result improved performance.

- **Unsupervised Learning**

- No teacher is present.
- The expected or desired output is not presented to the network.
- The system learns of it own by discovering and adapting to the structural features in the input patterns.

- **Reinforced learning**

- A teacher is present but does not present the expected or desired output but only indicated if the computed output is correct or incorrect.
- The information provided helps the network in its learning process.
- A reward is given for correct answer computed and a penalty for a wrong answer.

Note : The Supervised and Unsupervised learning methods are most popular forms of learning compared to Reinforced learning.

## ● Hebbian Learning

Hebb proposed a rule based on correlative weight adjustment.

In this rule, the input-output pattern pairs  $(X_i, Y_i)$  are associated by the weight matrix  $W$ , known as correlation matrix computed as

$$W = \sum_{i=1}^n X_i Y_i^T$$

where  $Y_i^T$  is the transpose of the associated output vector  $Y_i$

There are many variations of this rule proposed by the other researchers (Kosko, Anderson, Lippman) .

## ● Gradient descent Learning

This is based on the minimization of errors  $E$  defined in terms of weights and the activation function of the network.

- Here, the activation function of the network is required to be differentiable, because the updates of weight is dependent on the gradient of the error  $E$ .
- If  $\Delta W_{ij}$  is the weight update of the link connecting the  $i$ th and the  $j$ th neuron of the two neighboring layers, then  $\Delta W_{ij}$  is defined as

$$\Delta W_{ij} = \eta (\partial E / \partial W_{ij})$$

where  $\eta$  is the learning rate parameters and  $(\partial E / \partial W_{ij})$  is error gradient with reference to the weight  $W_{ij}$ .

Note : The Hoffs Delta rule and Back-propagation learning rule are the examples of Gradient descent learning.

- **Competitive Learning**

- In this method, those neurons which respond strongly to the input stimuli have their weights updated.
- When an input pattern is presented, all neurons in the layer compete, and the winning neuron undergoes weight adjustment .
- This strategy is called "winner-takes-all".

- **Stochastic Learning**

- In this method the weights are adjusted in a probabilistic fashion.
- Example : Simulated annealing which is a learning mechanism employed by Boltzmann and Cauchy machines.

## 5. Taxonomy Of Neural Network Systems

In the previous sections, the Neural Network Architectures and the Learning methods have been discussed. Here the popular neural network systems are listed. The grouping of these systems in terms of architectures and the learning methods are presented in the next slide.

### ● Neural Network Systems

- ADALINE (Adaptive Linear Neural Element)
- ART (Adaptive Resonance Theory)
- AM (Associative Memory)
- BAM (Bidirectional Associative Memory)
- Boltzmann machines
- BSB ( Brain-State-in-a-Box)
- Cauchy machines
- Hopfield Network
- LVQ (Learning Vector Quantization)
- Neoconition
- Perceptron
- RBF ( Radial Basis Function)
- RNN (Recurrent Neural Network)
- SOFM (Self-organizing Feature Map)

## Classification of Neural Network

A taxonomy of neural network systems based on Architectural types and the Learning methods is illustrated below.

		Learning Methods			
		Gradient descent	Hebbian	Competitive	Stochastic
Types of Architecture	Single-layer feed-forward	ADALINE, Hopfield, Perceptron,	AM, Hopfield,	LVQ, SOFM	-
	Multi-layer feed-forward	CCM, MLFF, RBF	Neocognition		
	Recurrent Networks	RNN	BAM, BSB, Hopfield,	ART	Boltzmann and Cauchy machines

**Table : Classification of Neural Network Systems with respect to learning methods and Architecture types**

## 6. Single-Layer NN Systems

Here, a simple Perceptron Model and an ADALINE Network Model is presented.

### 6.1 Single layer Perceptron

Definition : An arrangement of one input layer of neurons feed forward to one output layer of neurons is known as Single Layer Perceptron.

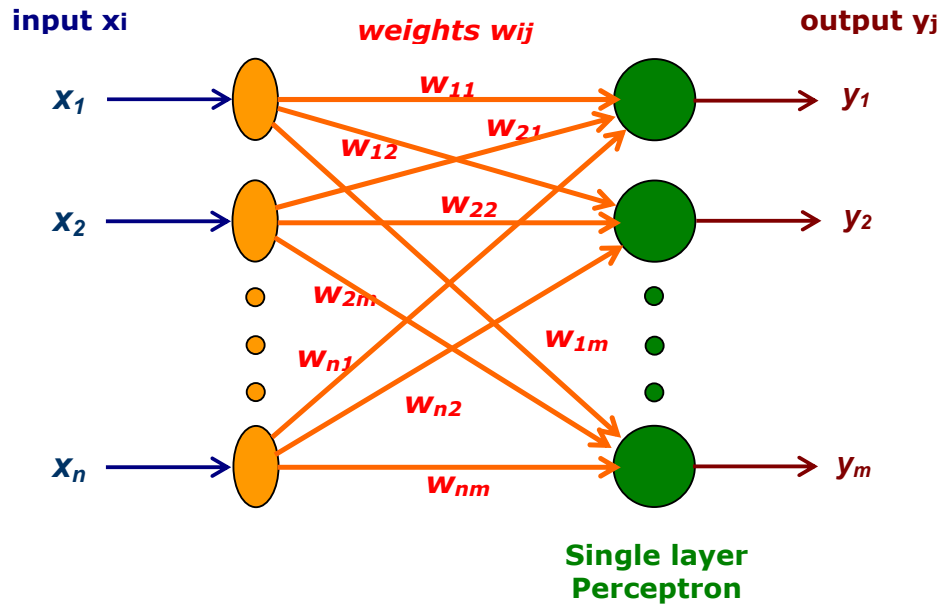


Fig. Simple Perceptron Model

$$y_j = f(\text{net}_j) = \begin{cases} 1 & \text{if } \text{net}_j \geq 0 \\ 0 & \text{if } \text{net}_j < 0 \end{cases} \quad \text{where } \text{net}_j = \sum_{i=1}^n x_i w_{ij}$$



## Learning Algorithm : Training Perceptron

The training of Perceptron is a supervised learning algorithm where weights are adjusted to minimize error when ever the output does not match the desired output.

- If the output is correct then no adjustment of weights is done.

$$\text{i.e. } W_{ij}^{K+1} = W_{ij}^K$$

- If the output is **1** but should have been **0** then the weights are decreased on the active input link

$$\text{i.e. } W_{ij}^{K+1} = W_{ij}^K - \alpha \cdot x_i$$

- If the output is **0** but should have been **1** then the weights are increased on the active input link

$$\text{i.e. } W_{ij}^{K+1} = W_{ij}^K + \alpha \cdot x_i$$

Where

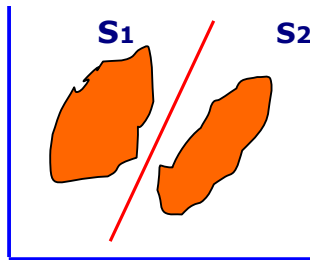
$W_{ij}^{K+1}$  is the new adjusted weight,  $W_{ij}^K$  is the old weight  
 $x_i$  is the input and  $\alpha$  is the learning rate parameter.  
 $\alpha$  small leads to slow and  $\alpha$  large leads to fast learning.

## Perceptron and Linearly Separable Task

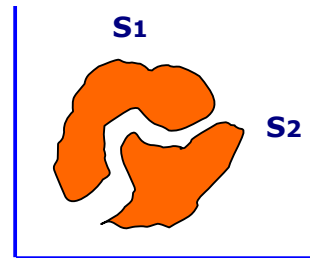
Perceptron can not handle tasks which are not separable.

- Definition : Sets of points in 2-D space are **linearly separable** if the sets can be separated by a straight line.
- Generalizing, a set of points in n-dimensional space are linearly separable if there is a hyper plane of (n-1) dimensions separates the sets.

### Example



(a) Linearly separable patterns



(b) Not Linearly separable patterns

Note : Perceptron cannot find weights for classification problems that are not linearly separable.

**XOR Problem :**

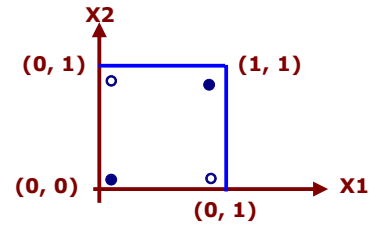
Exclusive OR operation

Input x1	Input x2	Output
0	0	0
1	1	0
0	1	1
1	0	1

} Even parity ●

} Odd parity ○

**XOR truth table**



**Fig. Output of XOR in X1 , x2 plane**

Even parity is, even number of 1 bits in the input

Odd parity is, odd number of 1 bits in the input

- There is no way to draw a single straight line so that the circles are on one side of the line and the dots on the other side.
- Perceptron is unable to find a line separating even parity input patterns from odd parity input patterns.

## Perceptron Learning Algorithm

The algorithm is illustrated step-by-step.

■ **Step 1 :**

Create a peceptron with **(n+1)** input neurons **x<sub>0</sub> , x<sub>1</sub> , . . . . . , x<sub>n</sub>** , where **x<sub>0</sub> = 1** is the bias input.

Let **O** be the output neuron.

■ **Step 2 :**

Initialize weight **W = (w<sub>0</sub> , w<sub>1</sub> , . . . . . , w<sub>n</sub> )** to random weights.

■ **Step 3 :**

Iterate through the input patterns **X<sub>j</sub>** of the training set using the weight set; ie compute the weighted sum of inputs **net j =  $\sum_{i=1}^n x_i w_i$**  for each input pattern **j** .

■ **Step 4 :**

Compute the output **y<sub>j</sub>** using the step function

$$y_j = f(\text{net } j) = \begin{cases} 1 & \text{if net } j \geq 0 \\ 0 & \text{if net } j < 0 \end{cases} \quad \text{where} \quad \text{net } j = \sum_{i=1}^n x_i w_{ij}$$

■ **Step 5 :**

Compare the computed output **y<sub>j</sub>** with the target output **y<sub>j</sub>** for each input pattern **j** .

If all the input patterns have been classified correctly, then output (read) the weights and exit.

■ **Step 6 :**

Otherwise, update the weights as given below :

If the computed outputs **y<sub>j</sub>** is **1** but should have been **0**,

Then **w<sub>i</sub> = w<sub>i</sub> - α x<sub>i</sub> , i = 0, 1, 2, . . . . , n**

If the computed outputs **y<sub>j</sub>** is **0** but should have been **1**,

Then **w<sub>i</sub> = w<sub>i</sub> + α x<sub>i</sub> , i = 0, 1, 2, . . . . , n**

where **α** is the learning parameter and is constant.

■ **Step 7 :**

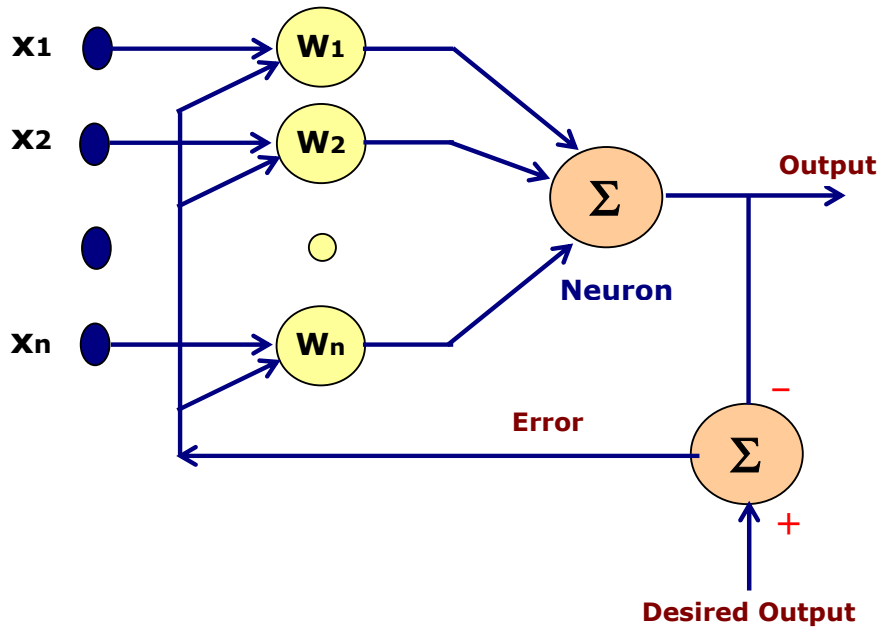
goto step 3

■ **END**

## ADaptive LINEar Element (ADALINE)

An ADALINE consists of a single neuron of the McCulloch-Pitts type, where its weights are determined by the normalized least mean square (LMS) training law. The LMS learning rule is also referred to as **delta rule**. It is a well-established **supervised training** method that has been used over a wide range of diverse applications.

- Architecture of a simple ADALINE



The basic structure of an ADALINE is similar to a neuron with a linear activation function and a feedback loop. During the training phase of ADALINE, the input vector as well as the desired output are presented to the network.

*[The complete training mechanism has been explained in the next slide.]*

## ADALINE Training Mechanism

(Ref. Fig. in the previous slide - Architecture of a simple ADALINE)

- The basic structure of an ADALINE is similar to a linear neuron with an extra feedback loop.
- During the training phase of ADALINE, the input vector  $\mathbf{X} = [x_1, x_2, \dots, x_n]^T$  as well as desired output are presented to the network.
- The weights are adaptively adjusted based on delta rule.
- After the ADALINE is trained, an input vector presented to the network with fixed weights will result in a scalar output.
- Thus, the network performs an  $n$  dimensional mapping to a scalar value.
- The activation function is not used during the training phase. Once the weights are properly adjusted, the response of the trained unit can be tested by applying various inputs, which are not in the training set. If the network produces consistent responses to a high degree with the test inputs, it is said that the network could generalize. The process of training and generalization are two important attributes of this network.

### Usage of ADLINE :

In practice, an ADALINE is used to

- Make binary decisions; the output is sent through a binary threshold.
- Realizations of logic gates such as AND, NOT and OR .
- Realize only those logic functions that are linearly separable.

## 7. Applications of Neural Network

Neural Network Applications can be grouped in following categories:

- **Clustering:**

A clustering algorithm explores the similarity between patterns and places similar patterns in a cluster. Best known applications include data compression and data mining.

- **Classification/Pattern recognition:**

The task of pattern recognition is to assign an input pattern (like handwritten symbol) to one of many classes. This category includes algorithmic implementations such as associative memory.

- **Function approximation :**

The tasks of function approximation is to find an estimate of the unknown function subject to noise. Various engineering and scientific disciplines require function approximation.

- **Prediction Systems:**

The task is to forecast some future values of a time-sequenced data. Prediction has a significant impact on decision support systems. Prediction differs from function approximation by considering time factor. System may be dynamic and may produce different results for the same input data based on system state (time).

## 8. References : Textbooks

1. "Neural Network, Fuzzy Logic, and Genetic Algorithms - Synthesis and Applications", by S. Rajasekaran and G.A. Vijayalaksmi Pai, (2005), Prentice Hall, Chapter 2, page 11-33.
2. "Soft Computing and Intelligent Systems Design - Theory, Tools and Applications", by Fakhreddine karray and Clarence de Silva (2004), Addison Wesley, chapter 4, page 223-248.
3. "Neural Networks: A Comprehensive Foundation", by Simon S. Haykin, (1999), Prentice Hall, Chapter 1-7, page 1-363.
4. "Elements of Artificial Neural Networks", by Kishan Mehrotra, Chilukuri K. Mohan and Sanjay Ranka, (1996), MIT Press, Chapter 1-5, page 1-214.
5. "Fundamentals of Neural Networks: Architecture, Algorithms and Applications", by Laurene V. Fausett, (1993), Prentice Hall, Chapter 1-4, page 1-214.
6. "Neural Network Design", by Martin T. Hagan, Howard B. Demuth and Mark Hudson Beale, ( 1996) , PWS Publ. Company, Chapter 1-7, page 1-1 to 7-31.
7. "An Introduction to Neural Networks", by James A. Anderson, (1997), MIT Press, Chapter 1- 12, page 1-401.
8. Related documents from open source, mainly internet. An exhaustive list is being prepared for inclusion at a later date.