INTRODUCTION TO SOFT COMPUTING

Paper Code - EOE-041

(Neural Network, Fuzzy Logic, Genetic Algorithms)

1. Artificial Intelligence

An area of computer science concerned with designing intelligent 'Computer Systems'. It means, to design a computer system that exhibits the characteristics we associate intelligence in humane behavior.

2. Soft Computing

The term 'soft computing' was introduced by Lofti A. Zadeh of University of California, Berkeley, U. S. A. According to Zadeh, soft computing differs from hard computing, hard computing methods predominantly based on the mathematical approaches therefore demands high degree of press ion and accuracy in their requirements, In fact, the role of human mind. Soft computing techniques have drawn there inherent characteristics from biological system, present effective methods for the solution for even difficult inverse problems. The guiding principle of soft computing is to exhibit the tolerance for impression, uncertainty and partial truth to achieve tractability, robustness, and low cost solution.

It is study of three technologies namely Neural Network (NA), Fuzzy Logic (FL) and Genetic Algorithms (GA). Hybrid intelligence systems deal with the synergistic integration of one or more technologies. As Shown in Figure 1.

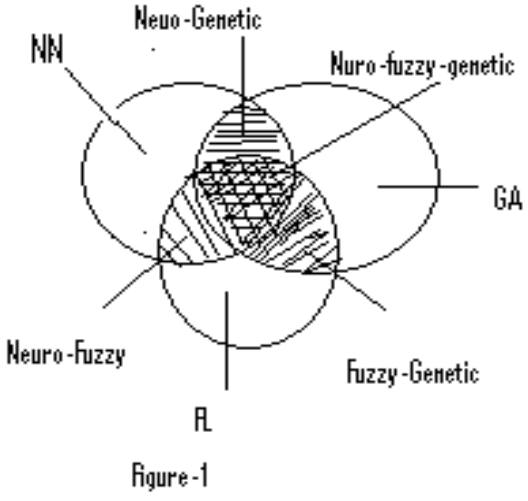

Figure-1

Here Combination of two technologies is called Nuro-genetic, nuro-fuzzy and fuzzy- genetic and combination of three technologies is called nero-fuzzy-

genetic systems.

a) Neural Networks

It is simplified model of human nervous system and therefore has drawn their motivation from the kind of computing performed by human brain. An NN, In general, is a highly interconnected network of a large number of processing elements called neurons an architecture inspired by human brain. An NN is massively parallel and therefore exhibit parallel distributed processing.

Neural network exhibits characteristics such as mapping capabilities of pattern association generalization, robustness, fault tolerance and parallel and high speed information processing.

Neural networks learn by examples. They can be therefore with known examples of a problem to 'acquire' knowledge about it. Once appropriately trained, the network can be used to effective use in solving 'unknown, problems

NNSs adopt various learning mechanisms of which supervised learning and unsupervised leaning methods are more popular.

Though NN architectures are broadly classified as single – layer feedforward networks, multilayer feedfarward network and recurrent networks, over the years several other NN architectures have been evolved. Some of the well known NN systems include *backpropagation networks, perceptron, ADALINE (ADAPTIVE LINEAL ELEMENT) associative Memory*, etc.

Neural networks have been successfully applied to problems in the field of pattern recognition, image processing, data compression, forecasting, and optimization etc.

b) Fuzzy Logic

Fuzzy set theory was proposed by Lotfi A. Zadeh (1965) is a generalization of classical set theory. ***Fuzzy logic representations founded on fuzzy set theory try to capture the way humans represent and***

reason with real-world knowledge in face of uncertainty. Uncertainty could arise due to generality, vagueness, ambiguity, chance, or incomplete knowledge**.**

A fuzzy set can be defined mathematically by assigning to each possible individual in the universe of discourse, a value representing its grade of membership in the fuzzy set. This grade corresponds to the degree to which that individual is similar or compatible with the concept represented by fuzzy set.

In classical set theory, an element either belongs to or does not belong to a set and hence, such sets termed crisp sets. But in fuzzy set, many degree of membership (between 0 and 1) and allowed. *Thus membership function* $\mu_{\tilde{A}}(x)$ *is associated with fuzzy set* $\tilde{A}$ *such that every function maps every element of the universe of discourse X to the interval [0, 1].*

The capability fuzzy sets to express gradual transitions from membership $0< (\mu_{\tilde{A}}(x) \leq 1)$ to non – membership $(\mu_{\tilde{A}}(x) =0)$. It not only provides for a meaningful and powerful representation of measurement of uncertainties but also provides for a meaningful representation of vague concepts expressed in natural language.

Operations such as union, intersection, subsethood, product, equality, difference and disjunction are also defined on fuzzy sets.

Crisp logic in based on crisp set theory, similarly fuzzy logic is based on fuzzy set theory. In fuzzy logic multivalued truth values such as true, absolutely true, fairly true, false, absolutely false, partly false and so on supported.

Fuzzy logic has found wide use control systems, pattern recognition, decision making etc.

c) Genetic Algorithms

Initiated and developed in early 1970s by John Holland (1973, 1975) are unorthodox search and optimization algorithms, which mimic some process of natural evaluation. *GA* ***performs a directed random search through a given set of alternatives with aim of finding best alternative with respect to given criteria of goodness***. ***These criteria are expressed in terms of objective function called fitness function.***

Fitness is defined as figure of merit, which is to be either minimized or maximized. It is further required that the alternatives ***be coded in some specific finite length which consists of symbols some finite alphabet. These strings are called chromosomes and symbols that form chromosomes are called genes. In case, alphabet (0, 1) the chromosomes are length of binary strings.***

Starting with an initial population of chromosome, one or more genetic inheritance operators are applied to generate offspring that competes for survival to make up next generation population. Genetic inheritance operators are reproduction, crossover, mutation, inversion, dominance, deletion, duplication, translocation, selection, migration etc**.**

Genetic algorithms are widely used in scientific and engineering areas including function optimization, machine learning, scheduling etc.

Unit 1 (Neural Network (Introduction & Architecture))

Lecture # 1 Neuron, Nerve Structure and Synapse

Neural Networks It is simplified model of biological nervous system. It is a massively parallel distributing processing system which is made of highly interconnected set of processing elements called *neurons*. They have ability to learn and acquire knowledge and make it available for use. There are various learning mechanisms, to enable NN acquire knowledge. The learning process is called is called training neural network.

NNs simplified imitations of human brain and motivation behind the computing performed by human brain. The structural constituents of human brain are termed as neurons

The networks which have been built on simplified imitation of computing the neurons of brain, called ANN (Artificial Neural Network), also called connectionist network, neuron-computers, or parallel distributed processors. Neurons are also called processing elements (PEs), nodes, neurodes, and units

1. Nerve Structure

The concept of neurons as the fundamental constituents of the brain

Brain contains about $\mathbf{10^{10}}$ basic units called *neurons*. Each unit in turn, is connected to $\mathbf{10^{4}}$ other neurons. A neuron is small cell that receives *electro-chemical signal* from its various sources and in term respond by transmitting *electrical impulses* to the other neurons. An average brain weighs about 1.5 kg and an average neuron have weight about 1.5×10^{-9} gms. Some of neurons of neurons perform input output operations referred afferent and efferent cells respectively. Remaining neurons are part of interconnected networks responsible for information storage and signal transmission.

2. Structure of Neuron

A neuron is composed of

Soma -A neuron is composed of a nucleus –a cell body called soma.

Dendrites-there is many filaments attached to soma are called dendrites. These are the input channels means inputs from other neurons arrive through dendrites.

Axon- Another type of links attached to soma is called axons. Axons are electrically active and serve as output channel.

Synapse- Junction point of axon and dendrite is called synapse.

Structure of biological neuron has shown in figure 1a

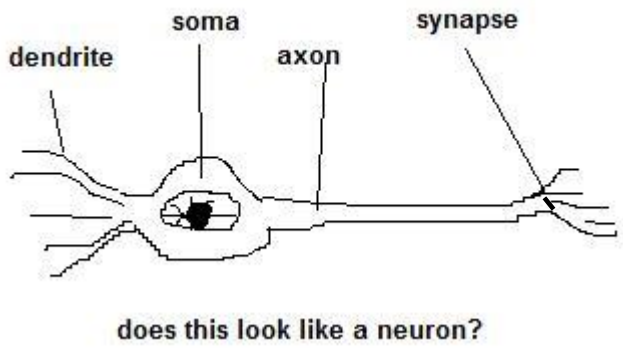

Figure1 a: Biological Neuron

If, the cumulative inputs received by soma raise the internal electrical potential of the cell, known as cell *membrane potential* then neuron 'fires' by propagating the action potential down the axon to excite other neurons. Axon terminates in specialized contact called synapse. The synapse is a minute gap between at the end of dendrite link contains a neuro-transmitted fluid. It is responsible for accelerating or retardating the electrical charges to the soma. In general a single neuron can have many synaptic inputs and synaptic outputs. The size of synapses is believed to be related to learning. The synapses with larger area are excitatory while those with smaller area are inhibitory.

Lecture # 2 Artificial Neuron and Its Model

Artificial neural network (ANN) is an efficient information processing system which resembles in characteristics with biological network. ANN possesses large number of highly interconnected processing elements called nodes or units or neurons which usually operate in parallel and are configured in regular architectures. Neurons are connected to other by a connection links. Each connection link is associated with weights. This link contains information about input signal. This information is used by neuron to solve a particular problem. ANN's collective behavior characterized by their ability to learn, recall and generalize pattern or data similar to that of human brain thereby capability to model network of original neurons as found in brain. Thus ANN's processing elements are called neurons or artificial neurons.

To understand basic operation of a neural net, consider a neural net as shown in *figure 2 a* below. Neurons X1 and X2, transmitting signal to neuron Y (output neuron). Inputs are connected to output neuron Y over interconnection links (W1 and W2).

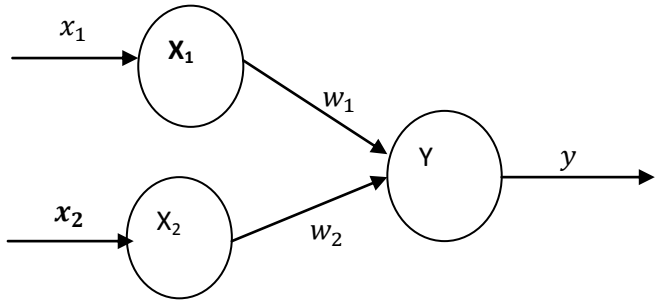

Figure2a. Architecture of a simple Artificial Neural network

Herex_1, x_2 are inputs and w_1, w_2 are weights attached to input links.

. Thus, weights here are multiplicative factors of inputs to account for the strength for synapse.

Net input $y_{in} = x_1 w_1 + x_2 w_2$

To generate the final output Y, the sum is passed to ***activation function (f)*** *or* ***transfer function of squash function*** which releases the output. Hence

$$y = f(y_{in})$$

The above calculation of net input is similar to the calculation of the output the output for pure straight line. As given in *figure 2b* below

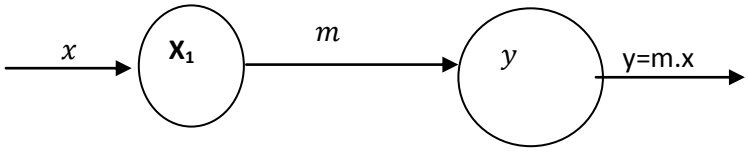

Figure 2 b. Neuron to model straight line

Here to obtain y slope is directly multiplied to input signal. It can be represented graphically as *figure 3 c*

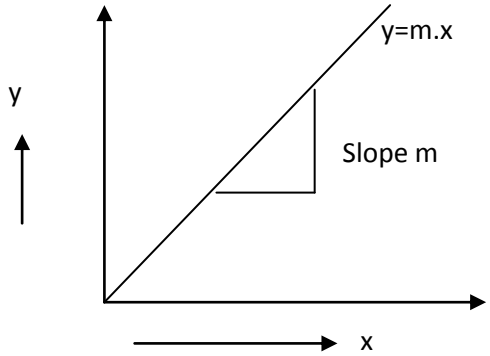

Figure 2 c. Graphical representation of y=mx

The mathematical representation of model of artificial neuron is shown in *figure 2 d* below

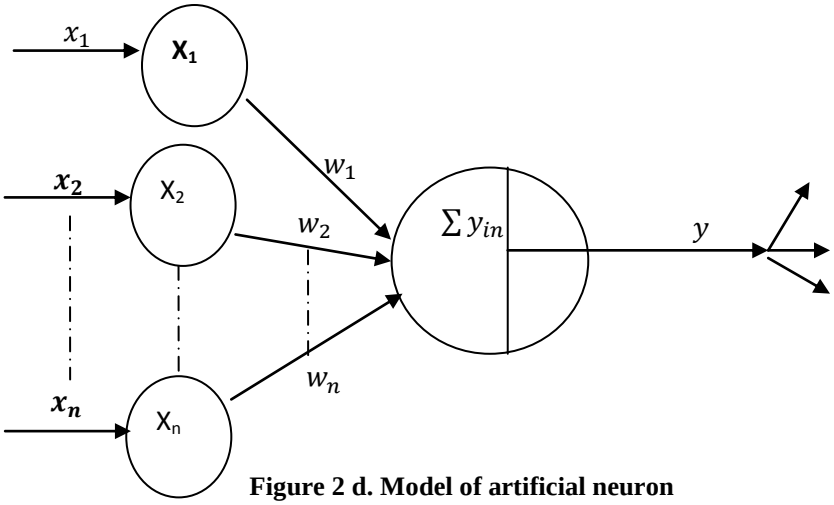

Figure 2 d. Model of artificial neuron

Here

Net input $y_{in} = x_1 w_1 + x_2 w_2 + \ldots\ldots\ldots\ldots\ldots + x_n w_n$ ***OR***

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

Terminology Relationship between biological neuron and artificial neuron

Biological Neuron	Artificial Neuron
Cell	Neuron
Dendrites	Interconnections
Soma	Net input
axon	output
synapse	Strength of signal

Comparison between biological Neuron and artificial Neuron

Properties	Biological Neuron	Artificial Neuron
Speed	Cycle Time of Execution in few milliseconds	Cycle Time of Execution is few nanoseconds
Processing	Slower	Faster
Size & Complexity	Highly Complex	Less Complex
Storage Capacity	Information is stores in synapse	Information is stored in contiguous memory locations
Tolerance	Possesses fault tolerant capability	They do not have fault tolerant capability
Control Mechanism	No control mechanism like artificial neurons	Artificial neurons are modeled using computer, so controlled by CPU.

Example 2.1

For the network shown in *figure 2 e* calculate net input to the output neuron**.**

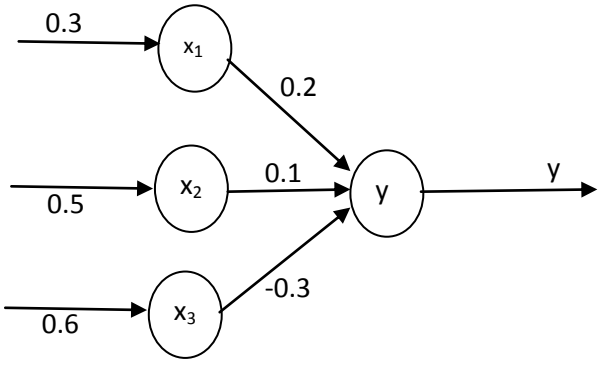

Figure 2 e

Sol. input vector X=$[x_1, x_2, x_3]$ = [0.3, 0.5, 0.6]
Weight Vector W=$[w_1, w_2, w_3]$
= [0.2, 0.1, - and 0.3]

The Net Input

$Y_{in} = x_1 w_{11} + x_2 w_{21} + x_3 w_{31}$

$= 0.3\times0.2 + 0.5\times0.1 + 0.6\times-0.3 = -0.07$

So Net input to Y is = -0.07

Lecture # 3 Activation Functions

To find output of a neuron we apply activation function on net input to that neuron.

There are several activation functions. Some of them are

1. Identity Function

It is also called linear function and can be defined as

$$y = y_{in}$$

It means output of neuron will be equal to its net input.

The linear function can be shown as in figure *3 a*

This function is basically used to find activation of input layer neurons

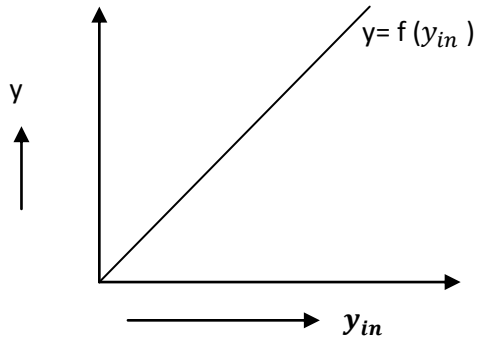

Figure 3 a. Identity Function

2. Binary Step Function

This function can be defined as,

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

Where, θ is a threshold value.

A very commonly used activation function is thresholding function. In this, sum is compared with threshold value θ . The step function is also known as ***Heaviside function*** .and is such that. It can be shown as in *figure 3 b*

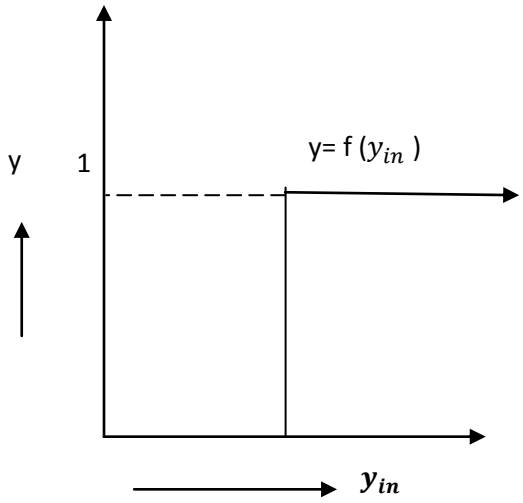

Figure 3 b. Binary Step Function

The function is used to convert net input to an output (1, 0).

3. Bipolar Step Function

This function can be defined as,

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

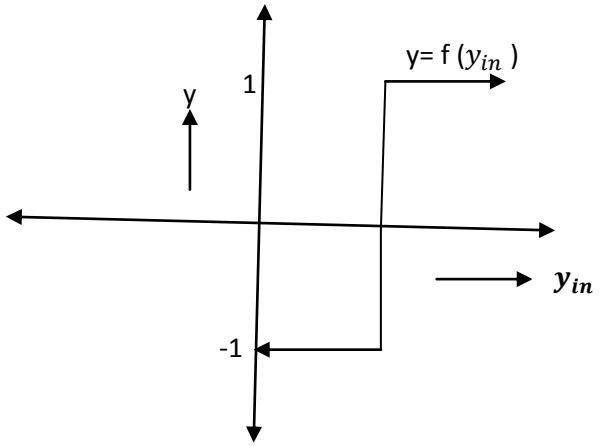

Figure 3 c Bipolar Step Function

Where, θ is a threshold value.

A very commonly used activation function is thresholding function. In this, sum is compared with threshold value θ .

The step function is also known as ***Signum or Quantizer function*** .and is such that. It can be shown as in *figure 3 c.*

The function is used to convert net input to an output (1,-1).

4. Binary Sigmoid Function

This function is a continuous function as shown in *figure 3d* that varies gradually between the asymptotic values ***0*** and ***1 (Unipolar Sigmoidal, or Binary))*** and is given by

$$y = \frac{1}{1+e^{-\lambda y_{in}}}$$

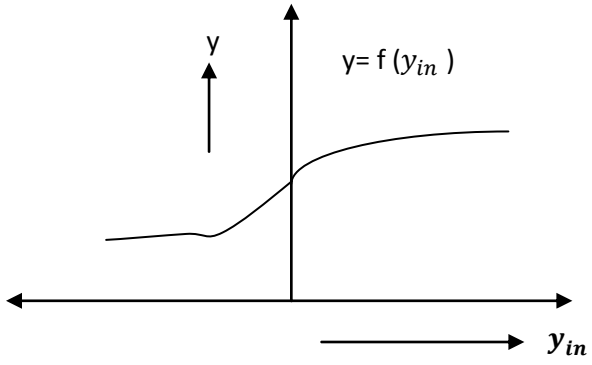

Figure 3 d. Binary Sigmoidal

Where **λ** is slope parameter, which adjust the abruptness of the function as it changes between two asymptotic values. This function is differentiable which an important feature in NN theory.

Derivatives Bipolar Sigmoid functions

$$y' = \lambda . y . (1-y)$$

(Note: - Sigmodal and Hyperbolic tangent functions are non - linear functions and rests are linear functions)

5. Bipolar Sigmoid Function

This function is a continuous function that varies gradually between the asymptotic values ***-1*** and ***1***, and is given by

$$\mathbf{y= \frac{2}{1+e^{-\lambda y_{in}}} - 1}$$

Where **λ** is slope parameter, which adjust the abruptness of the function as it changes between two asymptotic values. The graphical representation of bipolar sigmoidal is shown in *figure 3e.*

(In function 4 and 5 value of λ is1for representing the graph)

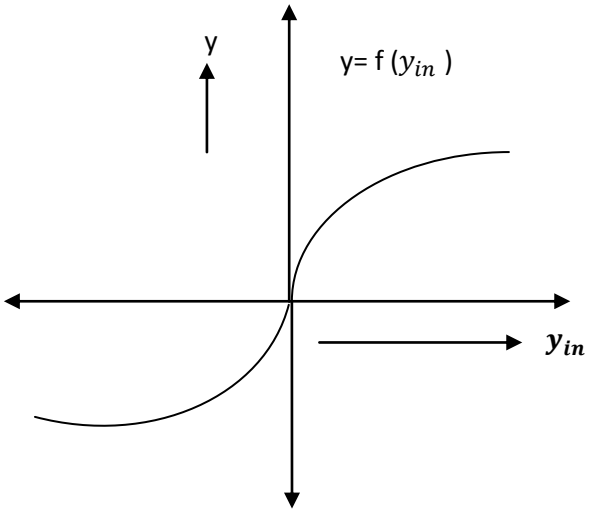

Figure 3 e Bipolar Sigmoidal

Derivative of Bipolar Sigmoid functions

$$\mathbf{y' = \frac{\lambda}{2}. (1+y). (1-y)}$$

6. Ramp Function

The function is defined as

$$y = \begin{cases} 1 & \text{if yin} > 1 \\ \text{yin} & 0 \le \text{yin} \le 1 \\ 0 & \text{xyin} < 0 \end{cases}$$

This function is also called piece-wise linear function

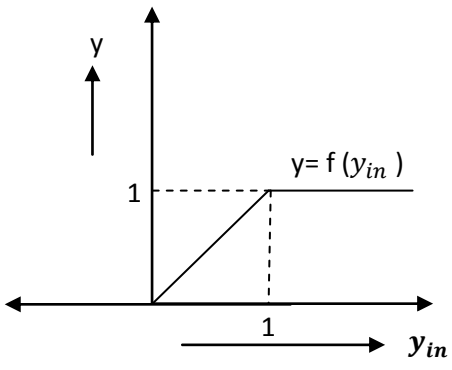

Figure 3 f Ramp Function

7. Tangent Function

It is given by $\mathbf{y = \tanh(yin)}$ and used to produce negative output values

Example 3.1

Calculate the net input and output to the neuron Y for the network *figure 3g* with bias input applied to the network using binary step function using threshold value *0.5*

Sol. input vector X=$[x_1, x_2, b]$ = [0.2, 0.6, 1]

Weight Vector W=$[w_{11}, w_{21}, w_0]$ = [0.3, 0.7, and 0.45]

And θ = 0.5

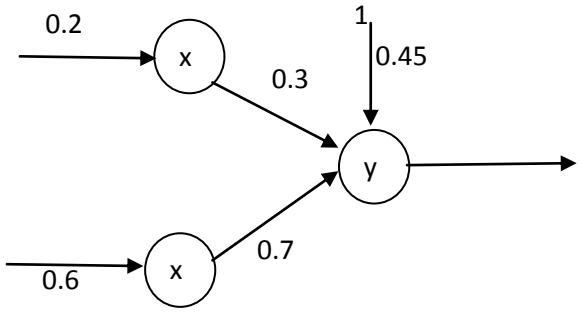

Figure 3 g

Net Input, Y_{in}=b+ x_1w_{11}+x_2w_{21}

= 0.45+0.2×0.3+0.6×0.7

=0.45+0.06+0.42

=0.93

As we know using step function

$$\mathbf{y} = \begin{cases} \mathbf{1} & \boldsymbol{if} \quad \boldsymbol{y_{in}} \geq \boldsymbol{\theta} \\ \mathbf{0} & \boldsymbol{if} \quad \boldsymbol{y_{in}} < \boldsymbol{\theta} \end{cases}$$ **So** Output y = 1 because $Y_{in} \geq 0.5$

Example 3.2

Obtain the output of neuron Y for the network shown in figure 3 f using activation functions as

(i) Binary Sigmoidal

(ii) Bipolar Sigmoidal

Sol. input vector X=$[x_1, x_2, x_3]$ = [0.8, 0.6, and 0.4]

Weight Vector W=$[w_{11}, w_{21}, w_{31}]$ = [0.1, 0.3, - and 0.2]

The Net Input

Y_{in}=b + x_1w_{11}+x_2w_{21}+x_3w_{31}

= 0.35+0.8×0.1+ 0.6×0.3+0.4×-0.2

=53

(i) Binary Sigmoidal

Output Y = $\frac{1}{1+e^{-y_{in}}} = \frac{1}{1+e^{-53}}$ = 0.625

(i) Bipolar Sigmoidal

Output Y = $\frac{2}{1+e^{-y_{in}}}$ - 1 = $\frac{2}{1+e^{-53}}$ – 1 =0.259

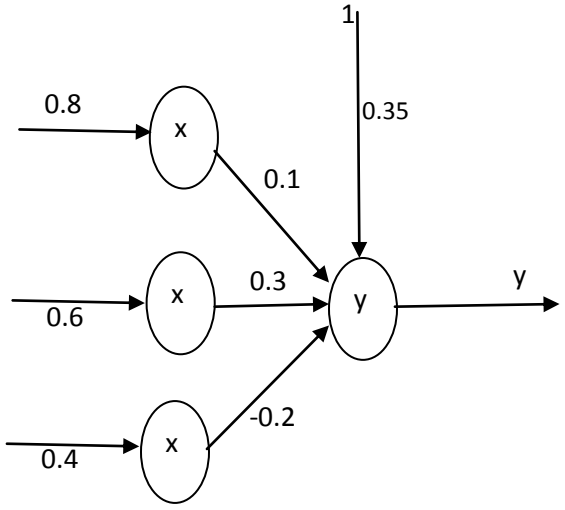

Figure 3 h

Lecture #4 Neural Network Architecture

An, ANN architecture is represented using directed graph. A graph G (V, E) is 2-tuple where V represents set of vertices and E represents set off edges. It assumes significance in neuron because signals in NN systems are restricted to flow in specific directions.

The vertices of the graph may represent neuron and the edges the synaptic links. There are several classes of NN according to their learning mechanism. There are three fundamental classes off networks

a. Single Layer Feedforward

The input layer neurons receive input signals and output neurons receive output signals. The synaptic links carry weights from every input neuron to every output neuron but not vice versa. This network is called single layer feedforward network and acyclic in nature. Since computations are done in output layer, it is called single layer feedforward network in spite of there are two layers.

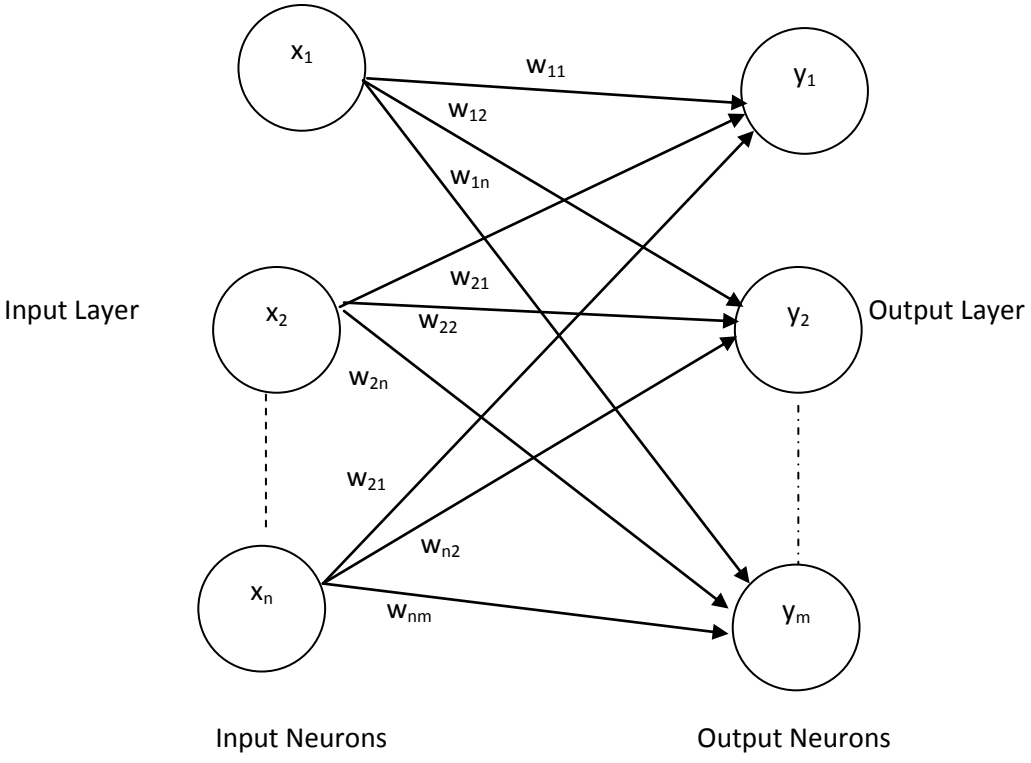

Figure 4 a: Single Layer Architecture (n-m architecture)

b. Multilayer Feedforward Network

As its name indicates is made up of multiple layers. Beside input layer and output layer this architecture has several intermediary layers called hidden layers. Computation of hidden unit is called hidden neurons. The hidden layer aids in performing useful intermediary computations before directing input to the output layer. The input layer neurons are linked to hidden layer neurons and the synaptic weights are called input-hidden weights. Again hidden layer neurons are linked to output neurons and corresponding weights are called hidden-output weights. The figure given below is called ***l – m – n*** architecture because there are l input neurons, ***m*** hidden neurons and ***n*** output neurons

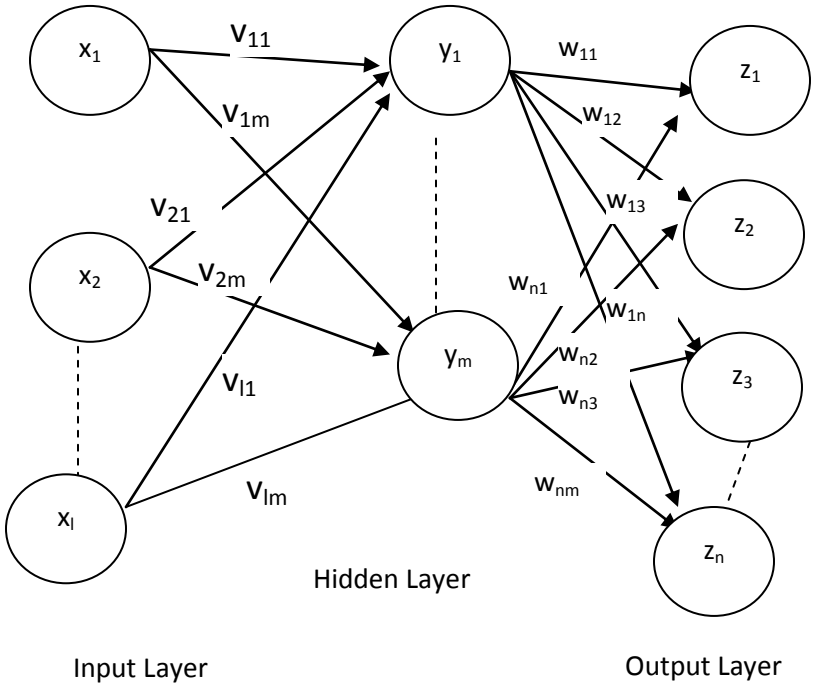

Figure 4 b: l-m-n multilayer architecture

Where x_i is input neuron, y_i is hidden neuron z_i is output neuron v_{ij} weight of interconnection between i^{th} input neuron and j^{th} hidden neuron and w_{jk} weight of interconnection between j^{th} hidden neuron and k^{th} output neuron.

c. Recurrent Networks

These networks differ from feedforward network architecture in the sense that there is at least one feedback loop. There could also be neurons with self-feedback link as shown figure. If a neurons feedback in same layer it is called *lateral feedback*..

Single layer network is shown figure blow with lateral feedback

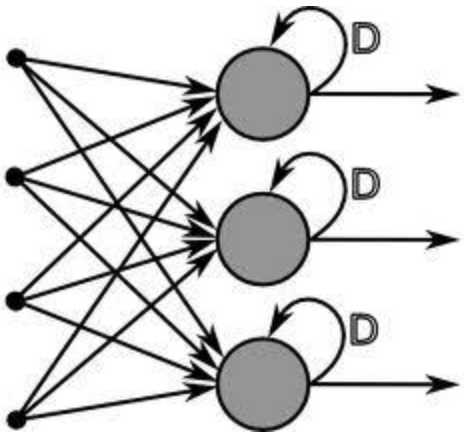

Figure 4 c: Single layer Recurrent Network

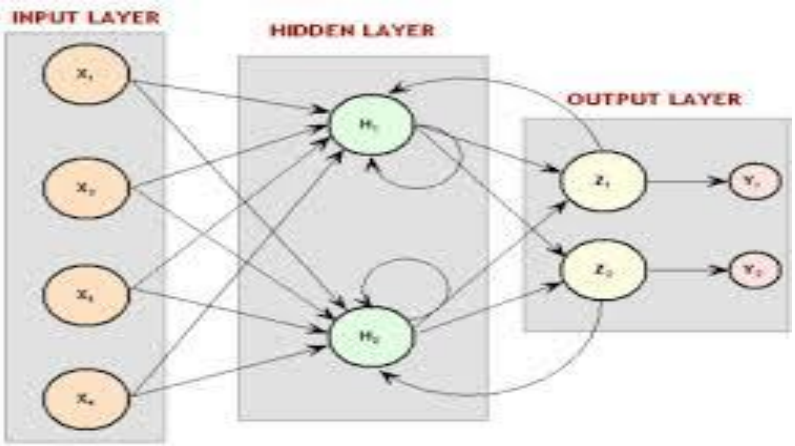

Figure 4 d. Multilayer Recurrent Network

Lecture 5# Various Learning Methods

In NN literature, learning methods can be classifies as

a. Supervised Learning

In this learning method, every input pattern that is used to train the network is associated with an output pattern, which is a target or desired pattern. A teacher is assumed to be present during the learning process, where a comparison is made between networks computed output and desired output, to determine error. The error is used to change network parameters. It results in an improvement in performance.

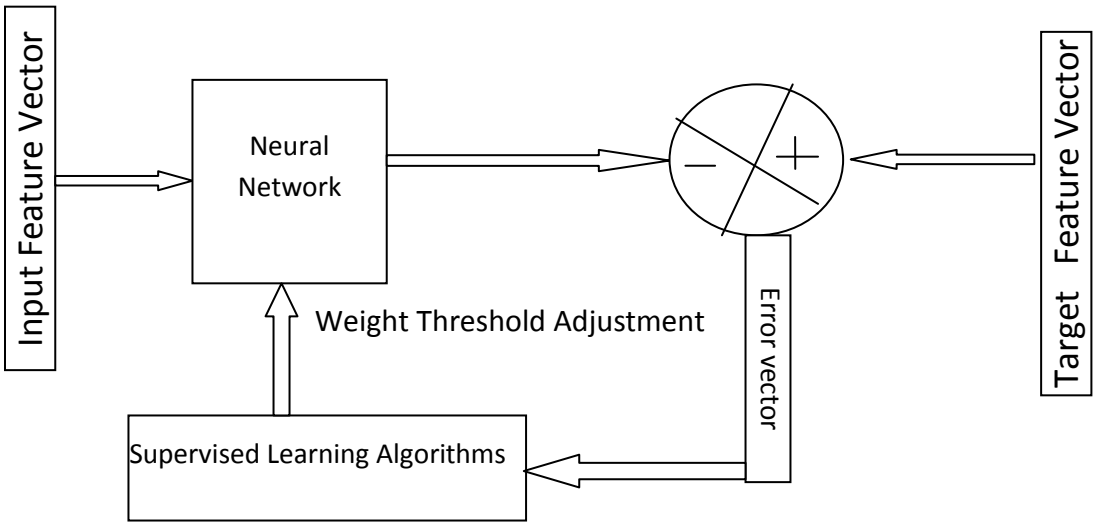

Figure 5a Supervised Learning

b. Unsupervised Learning

In this method, the target output is not presented to the network. It is as if there is no teacher to present the desired pattern and hence the system learns of its own by discovering and adapting to structural features in the input patterns.

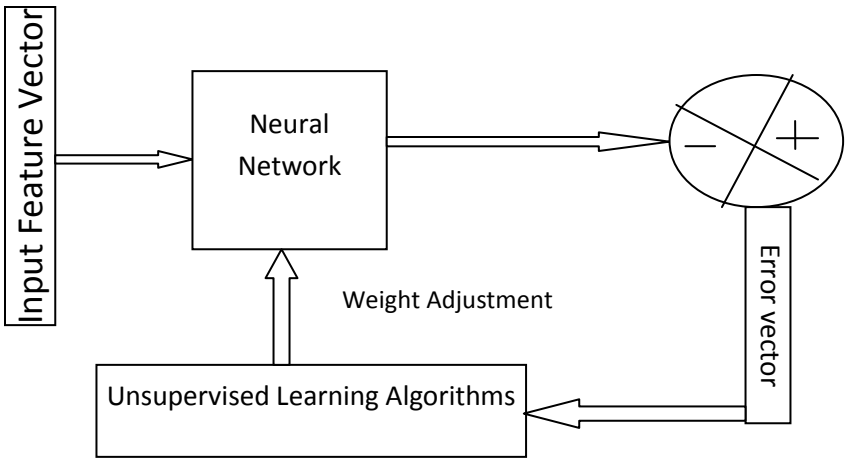

Figure 5 b Unsupervised learning

c. Reinforced Learning

In this method, a teacher through available does not present the expected answer but only indicate if the computed output is correct or not. The information is provided to the network in learning process, a reward is given to the correct answer computed and penalty for the wrong answer computed. It is not poplar form of learning

Supervised and unsupervised learning methods which are most popular form of learning have formed expression through various rules. Some of the widely used rules are

Hebbian Rule

Rule was proposed by Hebb (1949) and is based on correlative weight adjustment. This is mechanism inspired by Biology.

In this method input-output pattern pairs **(x_i, y_i)** are associated by weight **W**, known correlation matrix. It is computed as

$$W=\sum_{i=1}^{n} x_i \, y_i^T$$

Where y_i^T transpose is associated to y_i

Gradient Descent Learning

This is based on minimization of error ***E*** defined in terms of weight and activation function of the network. Also, it is required that the activation function employed by network is differentiable, as weight update is dependent on gradient of error ***E***

Thus, if Δw_{ij} is the weight update of the link connecting i^{th} and j^{th} neuron of the two neighboring layers, then

$\Delta \boldsymbol{w_{ij}} = \boldsymbol{\eta} \frac{\delta E}{\delta w_{ij}}$ where is $\boldsymbol{\eta}$ learning rate parameter, and $\frac{\delta E}{\delta w_{ij}}$ is the error gradient with reference to the weight $\Delta \boldsymbol{w_{ij}}$

The Widrow and Hoff's Delta Rule and back propagation rule all are example of gradient descent learning.

Competitive Learning

In this method, those neurons which respond strongly to the input stimuli have their weight updated. When an input pattern is presented, all neurons in the layer competent and the winning neuron undergo weight adjustment. Hence winner-take-all strategy.

Stochastic Learning Method

In this method weights are adjusted in probabilistic manner.

We can represent this method hierarchically as shown figure below

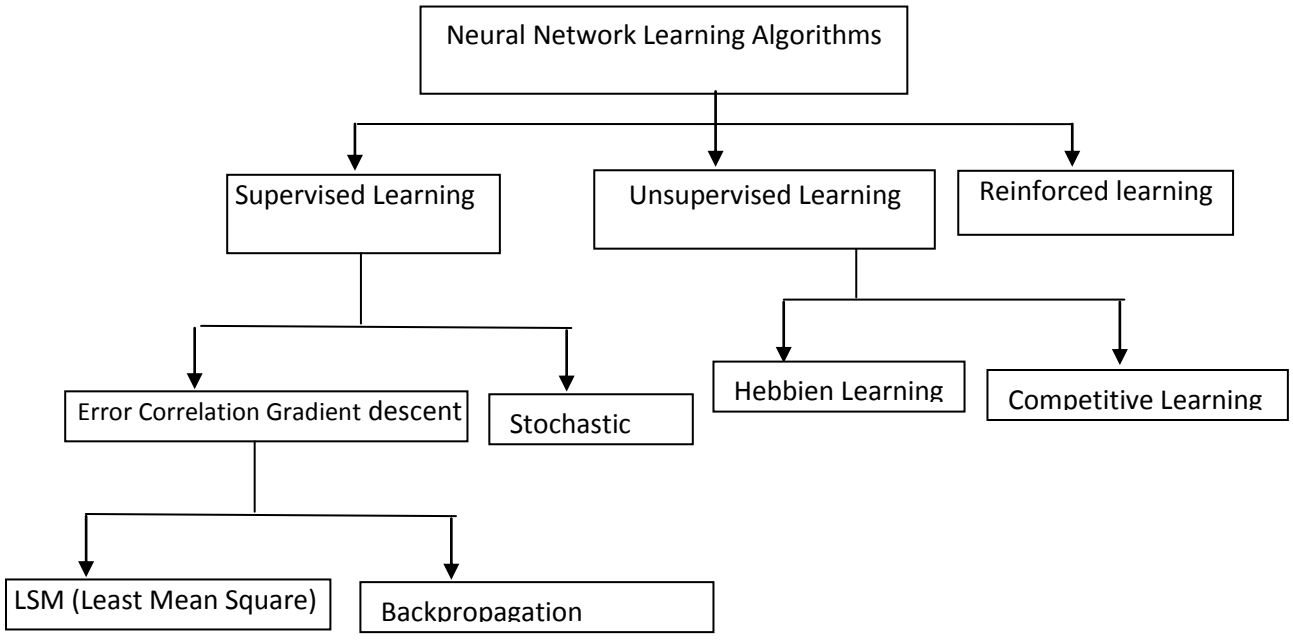

Figure 5 c. Hierarchical representation of Learning Methods

(Read this topic for your knowledge But in not your syllabus you May Skip this section)

History of Neural Net Taxonomy of Neural Network Architectures

Over the years, several NN systems have been evolved. The following are the some of systems that have acquired significance

- ADLINE (Adaptive Linear Network)
- ART (Adaptive resonance Theory)
- AM (Associated Memory)
- BAM (Bidirectional Associated Memory)
- BM (Boltzmann Machine)
- BSB (Brain – state-in-a-box)
- CCN (Cascade Correlation)
- Cauchy Machine
- CPN (Counter Propagation Network)
- Hamming Network
- Hopfield Network
- LVQ (Least Vector Quantization)
- MADLINE (Many ADLINE)
- MLFF (Multilayer Feedforward Network
- Recognition
- Perceptron
- RBF(Radial Bias Function)
- RNN(Recurrent Neural Network)
- SOFM (Self Organizing Feature Map)

History of Neural Network (Not Important you may skip this topic)

The pioneering work done by Mc Cullouch & Pitts (1949) founded growth of NN architecture. The next significant development was Hebbs leaning rule. Hebbs proposed learning derived from a model based on the synaptic connection between some cells responsible for biological associative memory.

Hebbien rule was later refined by Rosenblatt in 1958.

The only important contribution made in 1970 was Self Organizing map architecture based on completive learning**.**

Type of Architecture	***Learning Methods***			
	Gradient Descent	***Hebbien***	***Competitive***	***Stochastic***
Single Layer feedforward	ADLINE, Hopfield Perceptron	AM Hopfield	LVQ	
Multilayer Feedforward	CCN, RBF, MLFF	Neocognition	-	
Recurrent	RNN	BAM, BSB Hopfield	ART	Boltzmann Machine, Cauchy Machine

Name of Neural Network	**Developer**	**Year**	**Remarks**
Adaptive Resonance Theory(ART)	Carpenter, Grossburg and others	1980 and onwards	It employ a new principle of self organization called adaptive resonance theory based on competitive leaning
Back propagation Networks	Rumehart, Williams, Hindon, Werbor, Parker	1985	The back propagation learning rule is applicable on any Feedforward network. Slow rate of convergence
Bidirectional Associative Memory	Bart Kosko	1988	There are two layer of hetroassociative networks that can store pattern pairs and retrieve them
Boltzman and Cuchy Machine	Hinton, Sejnowiks,	1983, 1985,	These are stochastic networks whose states are governed by Boltzman Distribution/Cauchy

	Szu H, E. Hartley	1986, 1987	Distribution. The heavy computational load is a drawback
Brain-State-in-a-box	James Anderson	177	A recurrent Auto-associative network used Hebbien/Descent learning.
CPN	Robert Hechit, Nelson	1987	The network belongs to category of self organizing network and functions as statistically optimal self programming look-up table. The weight adjustment between layers follow
Hopfield Network	John Hopfield	1982	Single layer network which make use of Hebbien Learning/or Gradient Descent Learning
MADLINE network	Bernard, Widrow	1960, 1888	Crated by combination of ADALINE networks follow supervised learning rule based on minimum disturbance principle.
Neocognition	Kununiko, Fukushima	1982	A hybrid hierarchical Feedforward/Feedback network which closely models a human vision systems. The network employs either supervised or unsupervised rules
Perceptron	Frank, Rosenblatt	1958	A single layer or multi layer Feedforward network is able to obtain weight foe linearly separable tasks.
Self-Organizing-feature map Networks	Kohenen	1982	The network is simplified model of the feature to localize region mapping of a brain. It employees completive learning.

Lecture # 6 Perceptions and Convergence Rule

M-P Neuron

First mathematical model of biological neuron was presented by Warren McCullough and Walter Pitts (1943). It was known as McCullough and Pitts Model. It is also called M –P neuron.

The activation of M – P neuron is binary. It means at any time a neuron may "fire" or may not "fire". The weights associated with links may excitatory (Positive weights) i**.e. w>0 or** inhibitory (negative weighs) **i.e. –p<0**. All excitatory weights entering to a particular neuron will have same weights. Threshold plays major role in M-P neuron.

It should be noted that any none zero inhibitory input would prevent neuron from firing.
These neurons are widely used in case of logic functions.

Architecture of M-P Neuron model can be shown below in figure 6 a.

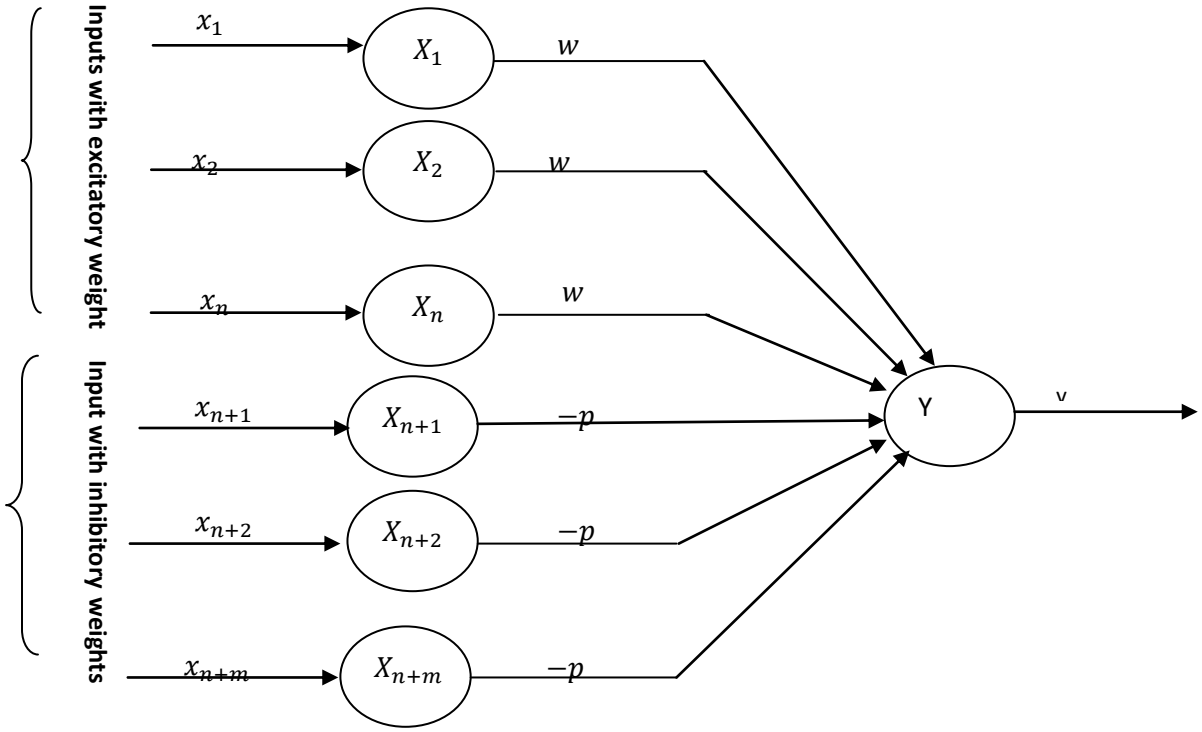

Figure 6 a. Model of M- P neuron

Since the firing of the output neuron is based upon the threshold, the activation function here is defined as,

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

For inhabitation to be absolute, the threshold with the activation must satisfy the following condition

$$\theta = n.\mathrm{w} - \mathrm{p}$$

The output neuron will fire if it receives say "k" or more excitatory inputs but no inhibitory inputs, where

$$k.w \geq \theta > (k-1).w$$

The M – P neuron has no particular testing algorithm. An analysis has to be performed to determine weights and threshold.

Example 6.1

Implement AND function using McCullough – Pitts Neuron

Sol[n].

The Truth Table for AND function is given as

input		*output*
0	0	0
0	1	0
1	0	0
1	1	1

Let $w_1 = 0$ *and* $w_2 = 2$

Net Input $Y_{in} = x_1 w_{11} + x_2 w_{21}$

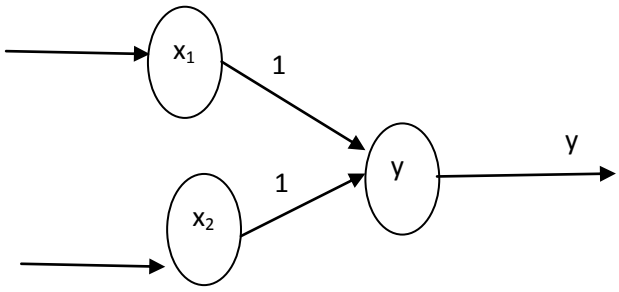

Figure 6.b. M – P Neuron Model for AND function

for Input (0, 0) , $0\times1+0\times1 = 0$
for Input (0, 1), $0\times1+1\times1 = 1$
for Input (0, 1), $1\times1+1\times1 = 1$
for Input (1, 1), $1\times1+1\times1 = 2$

$$y = f(Y_{in}) = \begin{cases} 1 & if\ Yin \geq \theta \\ 0 & if\ Yin < \theta \end{cases}$$

For AND function output will be 1 if net input $Y_{in} \geq \theta$, it means $2 \geq \theta$

Hence $\theta = 2$. The Neuron for AND net may be shown as in *figure 6 b*

Lecture # 7 Hebb Network

For a neural network, the Hebb rule is simple one. The rule was stated by Donald Hebb in 1949.

Hebb explained learning of brain by change in synaptic gap as

" When an axon of cell A is near enough to cell B, and repeatedly or permanently takes place in firing it, some growth process or metabolic change in one or both the cells such that A's efficiency as one of the cell firing B, is increased"

The weight update formula in Hebb rule is given by

$$w_{i\,(new)} = w_{i\,(old)} + x_i \,.\, y$$

Where, w_i is weigtht of link between i^{th} input neuron to output neuron Y. x_i is i^{th} input, and y is associated output**.**

The rule is well suited for bipolar data than binary data. If binary data is used, then above weight updating formula cannot distinguish, namely two conditions

1. ***A training pair in which an input unit is "on" and target value is "off".***
2. ***A training pair in which an input unit is "on" and target value is "off".***

Hebb rule is widely used in Pattern classification, Pattern Association, etc.

Training Algorithm for Hebb Net

Training algorithm used for calculation and adjustment of weights can be given below as

STEP 0: First weights and bias are initialized. They may be set to zero.

Means w_i =0 for (i = 1 to n where 'n' no of input neurons.)

STEP 1: Repeat step 2 -4 for each input training vector and target output pair s: t

STEP 2: Set activation of each input unit as,

$x_i = s_i$ **(i = 1 to n)**

Generally, the activation function of input layer is identity function.

STEP 3: Set activation of output unit as, **y = t**

STEP 4: Weight and bias adjustment is performed as,

$$w_{i\,(new)} = w_{i\,(old)} + x_i \,.\, y$$

$$b_{(new)} = b_{(old)} + .\, y$$

If inputs given in vector form, the weight updation formula is given as

$w_{(new)} = w_{(old)} + X \,.\, y$ ***Where*** $W = [w_1 \quad w_2 \ldots \quad w_i]$ ***and*** $X = [x_1 \quad x_2 \ldots \quad x_i]$

Example 1: Design a Hebb net to implement OR function using bipolar inputs and targets

Sol. The training pair for the OR function is given in Table below

x_1	x_2	1(bias)	target
1	1	1	1
1	-1	1	1
-1	1	1	1
-1	-1	1	-1

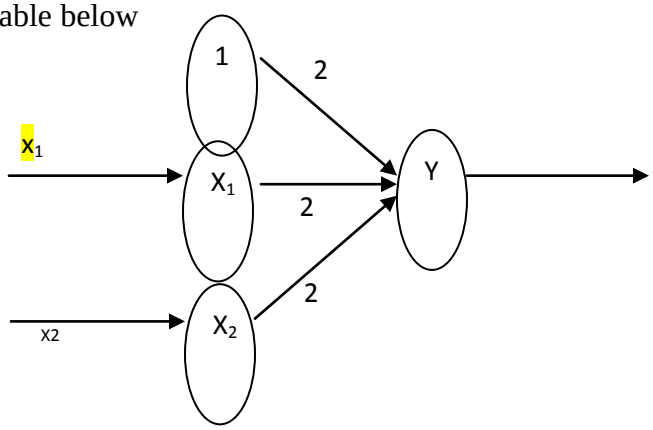

Figure 7 a. Network for OR function

Initially $w_1 = w_2 = b = 0$

The network is trained and final weights are obtained using the Hebb training algorithm as given in table below. When we input first training pair the initial weights will be the old weights. The weights obtained after presenting first pattern will be new weights. Similarly when we present second input pattern

weights obtained after presenting first pattern will be old weights. The process will be repeated for all input patterns. The final weights are shown in table below.

Inputs			**Output y**	**Weight Change**			***Initial Weights W1=0, w2=0, b=0***		
x1	**x2**	**1 (bias)**		**Δw1 = x1.y**	**Δw2= x2.y**	**Δb = y**	**w1(new)= w1(old)+Δw1**	**w2 (new) = w2(old)+ Δw2**	**b(new) = b(old)+ Δb**
-1	**-1**	**1**	**-1**	**1**	**1**	**-1**	**1**	**1**	**-1**
-1	1	1	1	-1	1	1	0	2	0
1	-1	1	1	1	-1	1	1	1	1
1	1	1	1	1	1	1	2	2	2

The final weights are $w_1=w_2=b=2$ for this figure can be given as above. The Hebb Net for OR function is shown in figure

Lecture # 8 Associative Memories

Motivation behind associative memory (A M) is human minds capability to associate patterns.
An associative memory (AM) which belongs to the class of singlelayer feedforward or recurrent network networks architecture depends on its association capability, exhibits Hebiean learning.

An associative memory network is storehouse of associated patterns which are encoded in some form. When the store house is triggered or incited with a pattern, the associated pattern pair is recalled (output) as shown in *figure 8 a* below

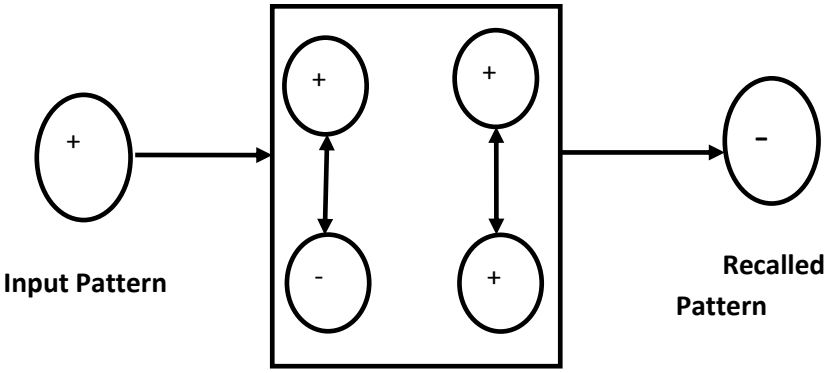

Figure 8 a. Associative Memory

If the associated pattern pairs (+, -) are different the model recalls + given - or vice-versa, then it is termed as **hetroassociative memory**. On the other hand, if + and - refer to the same pattern, then the model is termed as **autoassociative memory.**
Hetero-associative memories are useful for the association of patterns and autoassociative memories are used for image refinement, if in incomplete or partial or distorted or noisy pattern is input than whole pattern in its perfect form can be recalled.
Auto associative memories are known as autocorrerlator and hetroassociative memories are known as heterocorrelators.

a. Autoassociative Memory Network

The architecture of an associative neural net is shown in figure below. It consists of n input units in input layer and n output units in output layer. The input and output layers are connected through weighted interconnections. The input and output vectors are perfectly correlated with each other component by component.

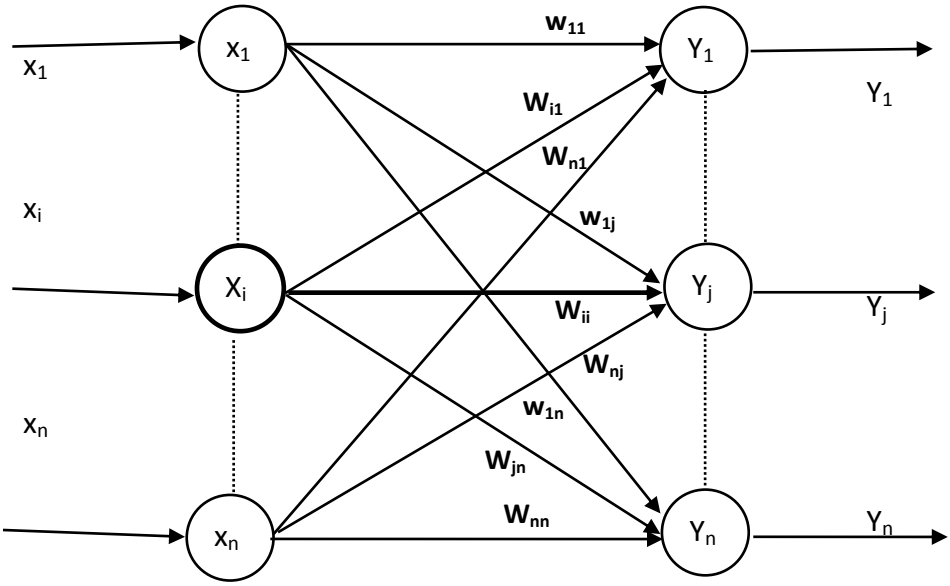

Figure 8 b: ***Architecture of autoassociative network***

Training Algorithm

Step 0: Initialize all weights to 0,

w_{ij} *= 0 (for i=1 to n for j=1 to n)*

Step 1: For each of the vector that has to be stored perform ***2 -4***.

Step 2: Activate each of the input unit

$x_i = s_i$ *(for i=1 to n)*

Step 3: Activate each of the output unit

$y_j = s_j$ *(for j =1 to n)*

Step 4: Adjust the weights

W_{ij} *(new)* $= w_{ij}$ *(old)* $+ x_i y_i$

OR

The weights can be also obtained as using formula

$$W = \sum_{p=1}^{p} s^T(p)\, s(p)$$

Testing Algorithms

An autoassociative memory neural network can be used to determine whether the given input vector is a "known" vector or an "unknown" vector. The net is said to recognize a "known" vector if the net produces a pattern of activation on the output units which is same as one of the vector stored in it. The procedure is given below

Step 1: Set the weights obtained for Hebb's rule or outer products.

Step 2: For each of the testing input vector presented perform 2 ***-4***.

Step 3: Set the activations of the input units equal to that of input vector.

Step 4: Calculate the net input to each output unit, j=***1 to n***

$$\mathbf{y_{inj} = \sum_{i=1}^{n} x_i w_{ij}}$$

Step 5: Calculate the output by applying the activation over the net input;

$$y_j = f(y_{inj}) = \begin{cases} +1 & if\ y_{inj} > 0 \\ -1 & if\ y_{inj} \leq 0 \end{cases}$$

These networks are dynamic Nature and follow feedback mechanism to recall stored patterns. **When network is with no self-connection the diagonal elements are set to 0**

These networks are widely used in speech processing, pattern classification, etc.

Example 1. Solved Numerical on Autoassociative network

Use outer product rule to store the vectors $[1 \quad 1 \quad 1 \quad 1]$ and $[-1 \quad 1 \quad 1 \quad -1]$ in an autossociative network.

(a) Weight matrix (with self connection)

(b) Test the vector$[1 \quad 1 \quad 1 \quad 1]$

(c) One mistake in the input vector

(d) One missing entry in the input vector

Solution

(a) Given vectors are $A_1 = [1 \quad 1 \quad 1 \quad 1]$ and $A_2 = [-1 \quad 1 \quad 1 \quad -1]$

. Weight matrix (W) = $\sum_{i=1}^{2} s^t(p)s(p)$

$= [A_1]^T[A_1] + [A_2]^T[A_2]$

$[1 \quad 1 \quad 1 \quad 1]^T.[1 \quad 1 \quad 1 \quad 1] + [-1 \quad 1 \quad 1 \quad -1]^T.[-1 \quad 1 \quad 1 \quad -1]$

$$\begin{bmatrix}1\\1\\1\\1\end{bmatrix}[1 \quad 1 \quad 1 \quad 1] + \begin{bmatrix}-1\\1\\1\\-1\end{bmatrix}[-1 \quad 1 \quad 1 \quad -1] = \begin{bmatrix}2 & 0 & 0 & 2\\0 & 2 & 2 & 0\\0 & 2 & 2 & 0\\2 & 0 & 0 & 2\end{bmatrix}$$

Since the network is with self connection set diagonal element to **0**

Hence W = $\begin{bmatrix}0 & 0 & 0 & 2\\0 & 0 & 2 & 0\\0 & 2 & 0 & 0\\2 & 0 & 0 & 0\end{bmatrix}$

(b) Testing of input vector x = $[1 \quad 1 \quad 1 \quad 1]$ the net input

y_{inj} = x. W=$[1 \quad 1 \quad 1 \quad 1]$ $\begin{bmatrix}0 & 0 & 0 & 2\\0 & 0 & 2 & 0\\0 & 2 & 0 & 0\\2 & 0 & 0 & 0\end{bmatrix}$ ==$[2 \quad 2 \quad 2 \quad 2]$

Now applying the activation we get output vector y_j = $[1 \quad 1 \quad 1 \quad 1]$. Hence correct respond is obtained.

(c) **One mistake in input pattern [-1 1 1-1].**

Let third bit 1 is changed to -1 that new pattern A_3= [-1 1 -1 -1] find Hamming Distance (HD) of A_3 from A_2 and A_1

HD (A_3, A_1) = [|-1-1|+|1-1|+|-1-1|+|-1-1|]

= [2+0+2+2]=6

HD (A_3, A_2) = [|-1+1|+|1-1|+|-1+1|+|-1+1|]

= [0+2+0+0]=2

Hence vector A_3 will be similar to A_2.

Net input

y_{inj} = x. W=$[-1 \quad 1 \quad -1 \quad -1] \begin{bmatrix} 0 & 0 & 0 & 2 \\ 0 & 0 & 2 & 0 \\ 0 & 2 & 0 & 0 \\ 2 & 0 & 0 & 0 \end{bmatrix}$

= $[-2 \quad -2 \quad 1 \quad -2]$

Now applying the activation we get output vector y_j = $[-1 \quad -1 \quad 1 \quad -1]$

Hence correct respond is obtained.

(d). Let us Consider One missing entry in $[-1 \quad 1 \quad 1 \quad -1]$ is $[-1 \quad 1 \quad 1 \quad 0]$

y_{inj} = x. W=$[-1 \quad 1 \quad 1 \quad 0] \begin{bmatrix} 0 & 0 & 0 & 2 \\ 0 & 0 & 2 & 0 \\ 0 & 2 & 0 & 0 \\ 2 & 0 & 0 & 0 \end{bmatrix}$ Net input vector

$[0 \quad 2 \quad 2 \quad -2]$ Now applying the activation we get output vector y_j = $[-1 \quad 1 \quad 1 \quad -]$

Hence correct respond is obtained.

Lecture #9 Hetroassociative Network

In case of hetroassociative network, the training inputs and target outputs are different. The input layer consists of ***n*** input units and output layer consist of ***m*** output units as shown in figure below.

Training algorithm for Heteroassociative Network

Step 0: Initialize all weights to 0,

w_{ij} ***= 0 (for i=1 to n for j=1 to m)***

Step 1: For each of the vector that has to be stored perform ***2 -4***.

Step 2: Activate each of the input unit

$x_i = s_i$ ***(for i=1 to n)***

Step 3: Activate each of the output unit

$y_j = s_j$ ***(for j =1 to m)***

Step 4: Adjust the weights

W_{ij} ***(new)*** $= w_{ij}$ ***(old)*** $+ x_i y_i$

Testing Algorithms

The testing algorithm used for testing the heteroassociative net with either noisy or with known input is as follows.

Step 0: Initialize weights from the training algorithm

Step 1: For each input vector presented perform 2 ***-4***

Step 2: Set the activations of the input units equal to that of the current input vector given, xi

Step 3: Calculate the net input to each output unit *j=1* to *n*

$$y_{inj} = \sum_{i=1}^{n} x_i w_{ij} \text{ (j = 1 to m)}$$

Step 4: Determine the activation of the output unit over the calculated net input. Calculate the output by applying the activation over the net input;

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

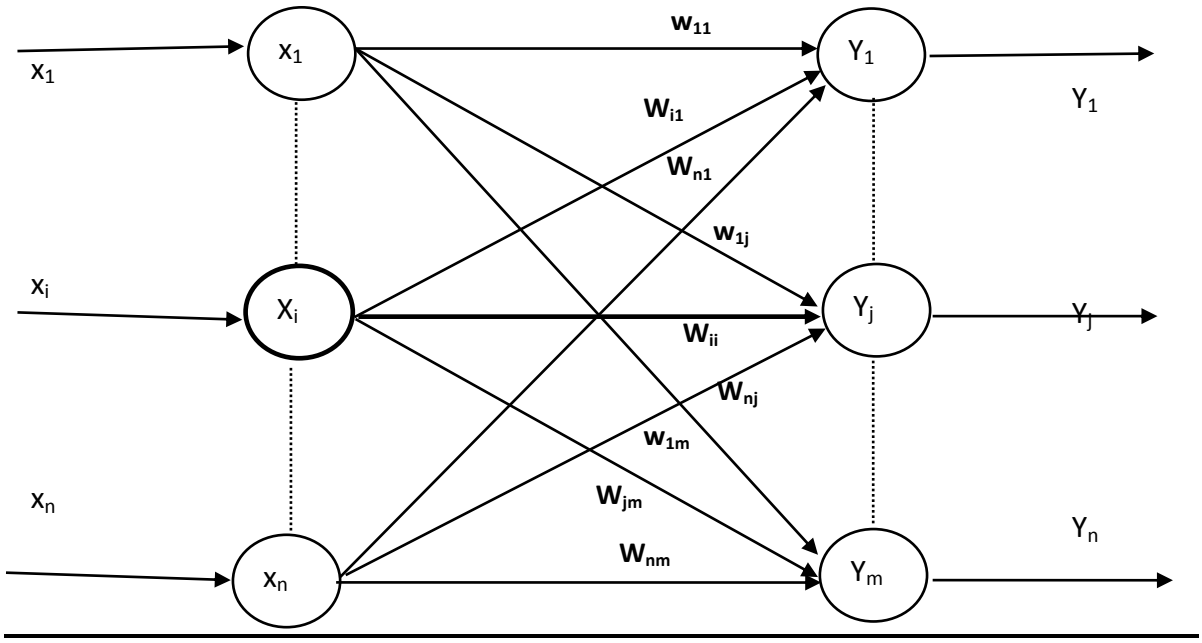

Figure 9 a. Heteroassociative Memory Architecture

Solved Numerical on Hetroassociative Network

Train a hetroassociative network using Hebb rule to store vector s= (s_1, s_2, s_3, s_4) to the output row vector t = (t_1, t_2) the vectors pairs as gven below in table. Also test the performance of network using training pairs.

S1	S2	S3	S4	T1	T2
1	0	0	0	1	0
1	1	0	0	1	0
0	0	0	0	0	1
0	0	1	1	0	1

Solution Given net is as in table above

Step 0: Initialize weights to zero.

Step 1: For training pair (1 0 0 0) : (1 0)

Step 2: Set the activation of input units

$x_1=1$ $x_2=0$ $x_3=0$ $x_4=0$

Step 3: Set the activation of output unit

$y_1=1$ $y_2=0$

Step 4: Update weights.

W_{ij} (new) $=w_{ij}$ (old) $+xi_1y_j$

w_{11} (new) $=w_{11}$ (old) $+x_1y_1$ $=0+1*1=1$

w_{21} (new) = w_21 (old) + x2y1 = 0+0*1=0

w_{31} (new) = w_31 (old) + x_3y_1 = 0+0*1=0

w_{41} (new) = w_{41}(old) + x_4y_1 = 0+0*1=0

W_{12} (new) = w21 (old) + x_1y_2 = 0+1*0=0

w_{22} (new) = w_{22} (old) + x_2y_2 = 0+0*0=0

w_{32} (new) = w_{32} (old) + x_3y_2 = 0+0*0=0

w_{42} (new) = w_{42} (old) + x_4y_2 = 0+0*0=0

For 2nd Input/output Pattern.

(1 1 0 0): (1 0)

Step 2. Set the activation of input units

$x_1=1$ $x_2=1$ $x_3=0$ $x_4=0$

Step 3. Set the activation of output unit

$y_1=_1$ $y_2=0$

Step 4. Update weights.

w_{ij} (new) = w_{ij}(old) + x_iy_j

w_{11}(new) = $w_1$1(old) + x1y1 =1+1*1=2

w_{21}(new) = w_{21}(old) + x_2y_1 =0+1*1=1

w_{31}(new) =w $_{31}$(old)+ x_3y_1 =0+0*1=0

w_{41}(new) = w_{41}(old) + x_4y_1 =0+0*1=0

$W_{12}(new) = w_{21}(old) + x_1y_2 = 0+1*0=0$

$w_{22}(new) = w_{22}(old) + x_2y_2 = 0+1*0=0$

$w_{32}(new) = w_{32}(old) + x_3y_2 = 0+0*0=0$

$w_{42}(new) = w_{42}(old) + x_4y_2 = 0+0*0=0$

For 3nd Input/Output Pattern.

(0 0 0 0): (0 1)

Step 2. Set the activation of input units

x1=0 x2=0 x3=0 x4 =0

Step 3. Set the activation of output unit

y1 =0 y2=1

Step 4 . Update weights.

wij(new) = wij(old) +xiyj

w11(new) = w11(old) + x1y1 =2+0*0=2

w21(new) = w21(old) + x2y1 =1+0*0=1

w31(new) = w31(old) + x3y1 =0+0*0=0

w41(new) = w41(old)+ x4y1 =0+0*0=0

W12(new) = w21(old) + x1y2 =0+0*1=0

w22(new) = w22(old) + x2y2 =0+0*1=0

w32(new) = w32(old) + x3y2 =0+0*1=0

w42(new) = w42 (old) + x4y2 =0+0*1=0

For 4th Input/Output Pattern.

(0 0 1 1): (0 1)

Step 2. Set the activation of input units

x1=0 x2=0 x3=1 x4 =1

Step 3. Set the activation of output unit

y1 =0 y2=1

Step 4 . Update weights.

wij (new) =	wij(old)+xiyj	
w11(new) =	w11(old)+x1y1	=2+0*0=2
w21(new) =	w21(old)+x2y1	=1+0*0=1
w31(new) =	w31(old)+x3y1	=0+1*0=0
w41(new) =	w41(old)+x4y1	=0+1*0=0
W12(new) =	w21(old)+x1y2	=0+0*1=0
w22(new) =	w22(old)+x2y2	=0+0*1=0
w32(new) =	w32(old)+x3y2	=0+1*1=1
w42(new) =	w42 (old)+x4y2	=0+1*1=1

Final Weights are W= $\begin{bmatrix} w_{11} & w_{12} \\ w_{12} & w_{22} \\ w_{13} & w_{23} \\ w_{14} & w_{24} \end{bmatrix} = \begin{bmatrix} \mathbf{2} & \mathbf{0} \\ \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} \\ \mathbf{0} & \mathbf{1} \end{bmatrix}$

IInd Method. (Use any one of them)

For first pair$(1 \quad 0 \quad 0 \quad 0)$: $(1 \quad 0)$

$W1=s^T.t = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} [1 \quad 0] = \begin{bmatrix} \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}$

For Second training t pair$(1 \quad 1 \quad 0 \quad 0)$: $(1 \quad 0)$

$W2=s^T.t= \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} [1 \quad 0] = \begin{bmatrix} \mathbf{1} & \mathbf{0} \\ \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}$

For the third training pair $(0 \quad 0 \quad 0 \quad 0)$: $(0 \quad 1)$

$W3=s^T.t= \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} [0 \quad 1] = \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}$

For the fourth training pair$(0 \quad 0 \quad 1 \quad 1)$: $(0 \quad 1)$

$$W4=s^T.t=\begin{bmatrix}0\\0\\1\\1\end{bmatrix}\ [0 \quad 1] = \begin{bmatrix}\mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{1}\\ \mathbf{0} & \mathbf{1}\end{bmatrix}$$

W= W1+W2+W3+W4

$$= \begin{bmatrix}\mathbf{1} & \mathbf{0}\\ \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{0}\end{bmatrix} + \begin{bmatrix}\mathbf{1} & \mathbf{0}\\ \mathbf{1} & \mathbf{0}\\ \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{0}\end{bmatrix} + \begin{bmatrix}\mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{0}\end{bmatrix} + \begin{bmatrix}\mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{1}\\ \mathbf{0} & \mathbf{1}\end{bmatrix} = \begin{bmatrix}\mathbf{2} & \mathbf{0}\\ \mathbf{1} & \mathbf{0}\\ \mathbf{0} & \mathbf{1}\\ \mathbf{0} & \mathbf{1}\end{bmatrix}$$

Performance Testing

Training rule used in the solution is Binary Activation is used.

$$Y = \begin{cases}1 & net\ input > 1\\ 0 & net\ input \leq 0\end{cases}$$

(a) For Input pattern $(1 \quad 0 \quad 0 \quad 0)$

$$[y_{in1}, y_{in2}] = [1 \quad 0 \quad 0 \quad 0] .W = [1 \quad 0 \quad 0 \quad 0] \begin{bmatrix}\mathbf{2} & \mathbf{0}\\ \mathbf{1} & \mathbf{0}\\ \mathbf{0} & \mathbf{1}\\ \mathbf{0} & \mathbf{1}\end{bmatrix}$$

$= [2 \quad 0]$

Applying activation function corresponding output vector will be $[y_1, y_2]$ =$[1 \quad 0]$ hence the correct response.

(b) For Input pattern $(1 \quad 1 \quad 0 \quad 0)$

$[y_{in1}, y_{in2}]$ = [1 1 0 0] .W = [1 1 0 0] $\begin{bmatrix} \mathbf{2} & \mathbf{0} \\ \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} \\ \mathbf{0} & \mathbf{1} \end{bmatrix}$

= [3 0]

Applying activation function corresponding output vector will be $[y_1, y_2]$ =[1 0] hence the correct response.

(c)For Input pattern (0 0 0 0)

$[y_{in1}, y_{in2}]$ = [0 0 0 0] .W = [0 0 0 0] $\begin{bmatrix} \mathbf{2} & \mathbf{0} \\ \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} \\ \mathbf{0} & \mathbf{1} \end{bmatrix}$

= [0 0]

Applying activation function corresponding output vector will be $[y_1, y_2]$ =[0 0] hence **no** correct response.

(c) For Input pattern (0 0 0 0)

$[y_{in1}, y_{in2}]$ = [0 0 1 1] .W = [0 0 1 1] $\begin{bmatrix} \mathbf{2} & \mathbf{0} \\ \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} \\ \mathbf{0} & \mathbf{1} \end{bmatrix}$

= [0 2]

Applying activation function corresponding output vector will be $[y_1, y_2]$ =[0 1] hence correct response.

Unit 1 Assignment

1. **Give comparison between artificial and biological neuron.**
2. **Define neuron, nerve structure and synapse.**
3. **Discuss various activation functions.**
4. **Define artificial neural network. State characteristics of an artificial neural network.**
5. **Briefly explain application domain of artificial neural network.**
6. **Define learning. Discuss various learning methods.**
7. **Construct recurrent with four input nodes, three hidden nodes and five out nodes with later feedback in output layer.**
8. **Give a comparison between Single layer feed forward network, multilayer feedforward network and recurrent network.**
9. **What is associative memory? Find weight matrix to store S1=(0 1 1 0), s2= (0 10 0) and t1=(1 0) t2 =(1 0) in a heteroassociative memory.**
10. **Define Auto and hetero associative memories.**