

RECOMMENDED BOOKS

- 1. Fuzzy Logic with Engineering Applications-** By T.J.Ross, 3rd edition, John Wiley Sons, 2010.
- 2. Principles of Soft Computing-** S.N.Sivanandam and S.N.Deepa, Wiley India edition, 2008.
- 3. Neural Networks, Fuzzy Logic and Genetic Algorithms: Synthesis and Applications-** By S.Rajsekharan and G.A.V. Pai, PHI 2007.
- 4. Genetic Algorithms: Search, Optimization and Machine Learning-** Davis E. Goldberg, Pearson Education, Fourth Impression, 2009.
- 5. Fuzzy Sets and Fuzzy Logic: Theory and Applications-** G.J.Klir and B.Yuan, PHI 1997

SOFT COMPUTING

- Soft Computing: **Coined by L.A.Zadeh in 1994**
- **1. L.A.Zadeh:** Fuzzy logic, Neural Networks and Soft Computing, communications of the ACM, March 1994, pp.77 - 84.
- **2. L.A.Zadeh:** Soft Computing and Fuzzy Logic, IEEE Software, vol.11, no.6, 1994, pp.48-56.

SOFT COMPUTING

- It is a **collection of methodologies** that aim
- to exploit the **tolerance for imprecision and uncertainty**
- to achieve **tractability, robustness and low solution cost.**
- Its principal constituents (**In the beginning**) were
 - Fuzzy logic
 - Neuro-computing and
 - Probabilistic reasoning
- The **role model** for soft computing is the human mind

ELABORATION

- **Tractability:** Being handled easily or controlled
- **Robustness:** Ability to resist disorder
- **Low solution cost:**
 - We live in a world that is pervasively imprecise, uncertain and hard to be categorical
- **Precision and certainty carry a cost**
 - For high precision and low uncertainty we have to pay some price

AN EXAMPLE

- Let us consider the **parking of a car.**
- We find it easy to park a car because the final position of the car is not specified precisely.
- If the final position is specified then the difficult of parking would increase geometrically with the increase in precision.
- **Eventually the parking may become impossible**

THE GUIDING PRINCIPLE

- Exploit the tolerance for imprecision, uncertainty and partial truth to achieve tractability, robustness and low solution cost.
- **HARD COMPUTING:**
 - The traditional computing is called hard computing.
 - In this imprecision of the real world is not accommodated.
- **AIM OF SOFT COMPUTING:** To emulate human mind as closely as possible

EVERYDAY USE OF APPROXIMATION

- We find that because of the exploitation of the tolerance for imprecision and uncertainty human have
- The remarkable ability to understand distorted speech
- Decipher sloppy handwriting
- Comprehend nuances of natural language
- Summarize text
- Recognize and classify images
- Drive a vehicle in dense traffic
- More generally: **Make rational decisions in an environment of uncertainty and imprecision**

HARD COMPUTING	SOFT COMPUTING
Conventional computing techniques those are deterministic and have sharp boundary	Non conventional approaches those are stochastic and have vague boundary
Precise, certain and has two valued (Boolean) logic	Imprecise, uncertain and has multi valued logic
Needs exact input	Can handle ambiguous and noisy data
Not tractable	Tractable solution
High computational cost	Low computational cost
low Intelligence Quotient (MIQ)	High Machine Intelligence Quotient (MIQ)
Precise	Approximate reasoning

SOME APPLICATION AREAS OF SOFT COMPUTING

- **Data clustering**
- Database anonymisation
- **Rule generation**
- Image processing
- **Medical diagnosis**
- Pattern recognition
- **Social networks**
- Distributed computing
- **Parallel processing**
- Machine learning and
- **Granular computing**

NEURAL NETWORKS

- A neural network **is a processing device**
- It is either **an algorithm** or **an actual hardware**
- Its **design is inspired by** the design and functioning of **animal brains** and **components thereof**
- **It has the ability to learn by example**
- **It has made them very flexible and powerful**
- **The networks are also well suited for real-time systems**
- **They have fast response and computational times**
- **They have a parallel architecture**

HUMAN BRAIN

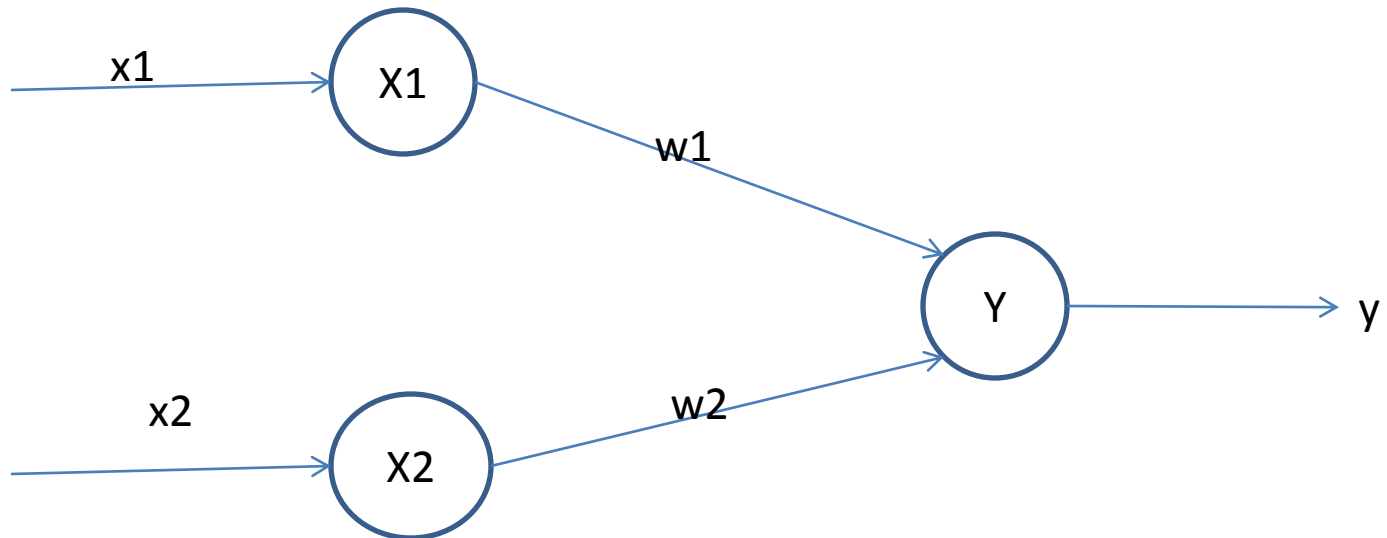
- It is an amazing processor
- Its exact working is still a mystery
- The basic elements of the human brain are **specific type of cells**, known as **neurons**
- Neurons do not regenerate
- Neurons provide us with **abilities to remember, think and apply previous experiences** to our every action
- Human brain comprises of about **100 billion neurons (10^{11})**
- Each neuron is connected to **200,000** other neurons
- **THE POWER OF HUMAN MIND COMES FROM THE SHEER NUMBER OF NEURONS AND THEIR MULTIPLE INTERCONNECTIONS**

ARTIFICIAL NEURAL NETWORKS (ANN)

- Artificial Neural Networks are **information processing systems**
- Constructed and implemented **to model the human brain**
- The **main objective** of the neural network research is
- To **develop a computational device**
- For **modeling the brain**
- To **perform various computational tasks**
- At a faster rate **than the traditional systems**

ARCHITECTURE OF A SIMPLE ANN

- Fig. $y_{in} = x_1 w_1 + x_2 w_2$ $y = f(y_{in})$ f : Activation function
 X_1, X_2 : Input neurons Y : Output neuron
 X_1, X_2 transmit signals, Y receives signal



x_1, x_2 : activations of the input neurons : output of input signals

w_1, w_2 : associated weights, which contain information about the input signals

TASKS PERFORMED BY ANN

- **Pattern-matching**
- **Classification**
- **Optimization of functions**
- **Approximation**
- **Vector quantization**
- **Data Clustering**

ANN CONTD...

- ANN is defined as an **information-processing model**
- Inspired by the way **biological nervous systems** such as the brain processes information
- It tries to replicate the **most basic functions of the brain**
- It comprises of a large number of **highly interconnected processing elements**(neurons) working in unison to solve specific problem
- An ANN is **configured for a specific application**, such as pattern recognition or data classification through a learning process

COMPONENTS OF BIOLOGICAL NEURAL NETWORKS

- **SOMA or CELL BODY:**
 - Inside the cell body we have the cell nucleus
- **DENDRITES:**
 - Where the nerve is connected to the cell body
 - Are tree like networks made of nerve fiber connected to the cell body
- **AXON:**
 - Which carries the impulses of the neuron
 - A single long connection extending from the cell body and carrying signals from the neuron

COMPONENTS OF BIOLOGICAL NN CONTD...

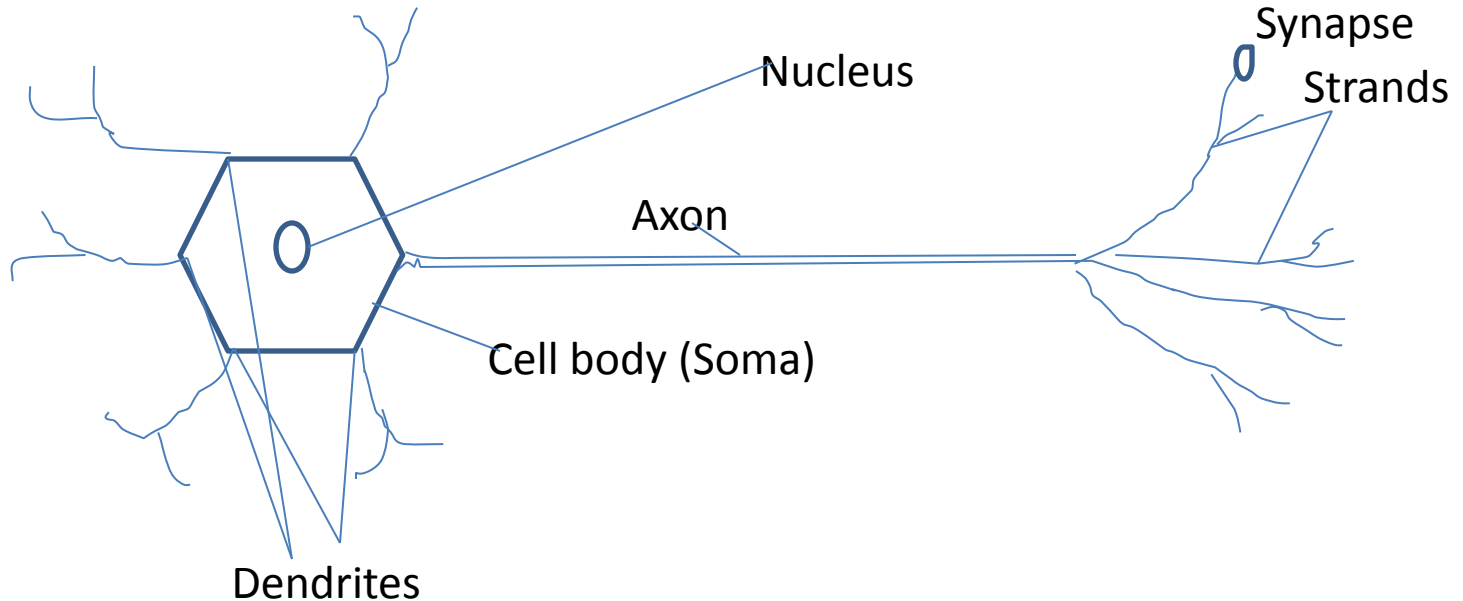
- **STRANDS:**
 - The components into which an axon splits into at its end
- **SYNAPSE:**
 - A small bulb-like organ into which each strand terminates
- **THROUGH SYNAPSES THE NEURON INTRODUCES ITS SIGNALS TO OTHER NEARBY NEURONS**
- **THE RECEIVING ENDS OF THESE SYNAPSES ON THE NEARBY NEURONS CAN BE FOUND BOTH ON THE DENDRITES AND ON THE CELL BODY**

SCHEMATIC DIAGRAM OF A BIOLOGICAL NEURON

- dia

Synapse : bulb like organ at the end of strands

Strands : The splits at the end of axons



Dendrites : Where the nerve is connected to the cell body

Axon : Which carries the impulses of the neuron

BIOLOGICAL NN CONTD...

- In the human brain there are approximately **10000** synapses per neuron
- **Mathematical representation of the above process in ANN is:**
- Suppose there are n inputs from n neurons X_1, X_2, \dots, X_n with activations x_1, x_2, \dots, x_n respectively
- Let the weights of the interconnections between X_1, X_2, \dots, X_n and the connecting neuron **Y** be w_1, w_2, \dots, w_n respectively

ARTIFICIAL NEURAL NETWORKS CONTD...

- The net input to the neuron Y is given by the formula:

$$y_{in} = x_1 w_1 + x_2 w_2 + \dots x_n w_n$$

- The activation function is applied to y_{in} to compute the output
- The **weight represents the strength of synapse** connecting the input and the output neurons
- The weights may be positive or negative
- **+ve** weight means the **synapse is excitatory**
- **-ve** weight means the **synapse is inhibitory**

TERMINOLOGICAL RELATIONSHIP BETWEEN BIOLOGICAL NN AND ANN

Biological Neuron	Artificial Neuron
Cell	Neuron
Dendrites	Weights or Interconnections
Soma	Net input
Axon	Output

COMPARISON BETWEEN BIOLOGICAL NEURON AND ARTIFICIAL NEURON

- **SPEED:**
 - The **cycle time** of execution in the **AN** is of **few nanoseconds**
 - In **biological neurons** the **cycle time** is of a **few milliseconds**
- **PROCESSING:**
 - **Biological neuron** can perform **massive parallel operations** simultaneously
 - **AN** can also perform **several parallel operations** simultaneously.
- **PROCESSING IN ANN IS FASTER THAN BIOLOGICAL NN**

COMPARISON BETWEEN BIOLOGICAL NEURON AND ARTIFICIAL NEURON

- **SIZE AND COMPLEXITY:**
- **IN BRAIN:**
 - The total no. of neurons is 10^{11}
 - The total no. of interconnections is 10^{15}
 - The **complexity** of a biological neuron is **high**
 - The **size** of a biological neuron is **high**
 - The **computational work** is carried not only **inside the cell body** but also **in axon, synapse** etc.
- **IN ANN:**
 - The size and complexity depends upon the **chosen application** and the **network designer**

COMPARISON BETWEEN BIOLOGICAL NEURON AND ARTIFICIAL NEURON

- **STORAGE CAPACITY (MEMORY)**
- The biological neuron stores the information in its interconnections or in synapse strength
- **AN** stores in its contiguous memory locations
- In **AN** the loading may sometimes overload the memory
- Some of the information in the contiguous memory may be lost
- In biological neuron the new information can be added in the interconnections by adjusting the strength without the information being lost
- In Biological **N**, the recalling may fail, but in **AN** it never fails

COMPARISON BETWEEN BIOLOGICAL NEURON AND ARTIFICIAL NEURON

- **TOLERANCE**
- The biological N has fault tolerance capability where as the AN has no fault tolerance
- In biological N the storage and retrieval continues even if the interconnections get disconnected
- In AN the information gets corrupted if the interconnections are disconnected
- Biological N accepts redundancies
- This is not possible for AN

COMPARISON BETWEEN BIOLOGICAL NEURON AND ARTIFICIAL NEURON

- **CONTROL MECHANISM**
- In AN there is a control unit in CPU, which can transfer and control precise scalar values from unit to unit
- There is no such control unit in the brain
- The **strength** of the N in the **brain** depends on the **active chemicals present** and whether neuron connections are **strong or weak**
- So in turn **on the structure of the layer** rather than the individual synapses
- **ANs** possess simpler interconnections and is **free from chemical actions** similar to those taking place in brain
- **Control mechanism in AN is simpler than that in brain**

BASIC MODELS OF ARTIFICIAL NEURAL NETWORK

- The models are specified by the **three basic entities**
 - The model's **synaptic** (through synapses) **interconnections**
 - The **training or learning rules** adopted for updating and adjusting the connection weights
 - The **activation functions**

CONNECTIONS

- The neurons can be visualised for their arrangements in layers
- An ANN consists of a **set of highly interconnected processing elements** called neurons
- **Output** of each processing element is found to be **connected through weights** to the other processing elements or itself
- Delay lead and lag-free connections are allowed
- **Arrangement of these processing elements** and the geometry of their interconnections are **essential** for an ANN
- The point where the connection originates and terminates should be noted
- The **function** of each processing element in an ANN **should be specified**

BASIC NEURON CONNECTION ARCHITECTURES

- There are **five** types of basic connections:

1. SINGLE-LAYER FEED FORWARD NETWORK

2. MULTI-LAYER FEED FORWARD NETWORK

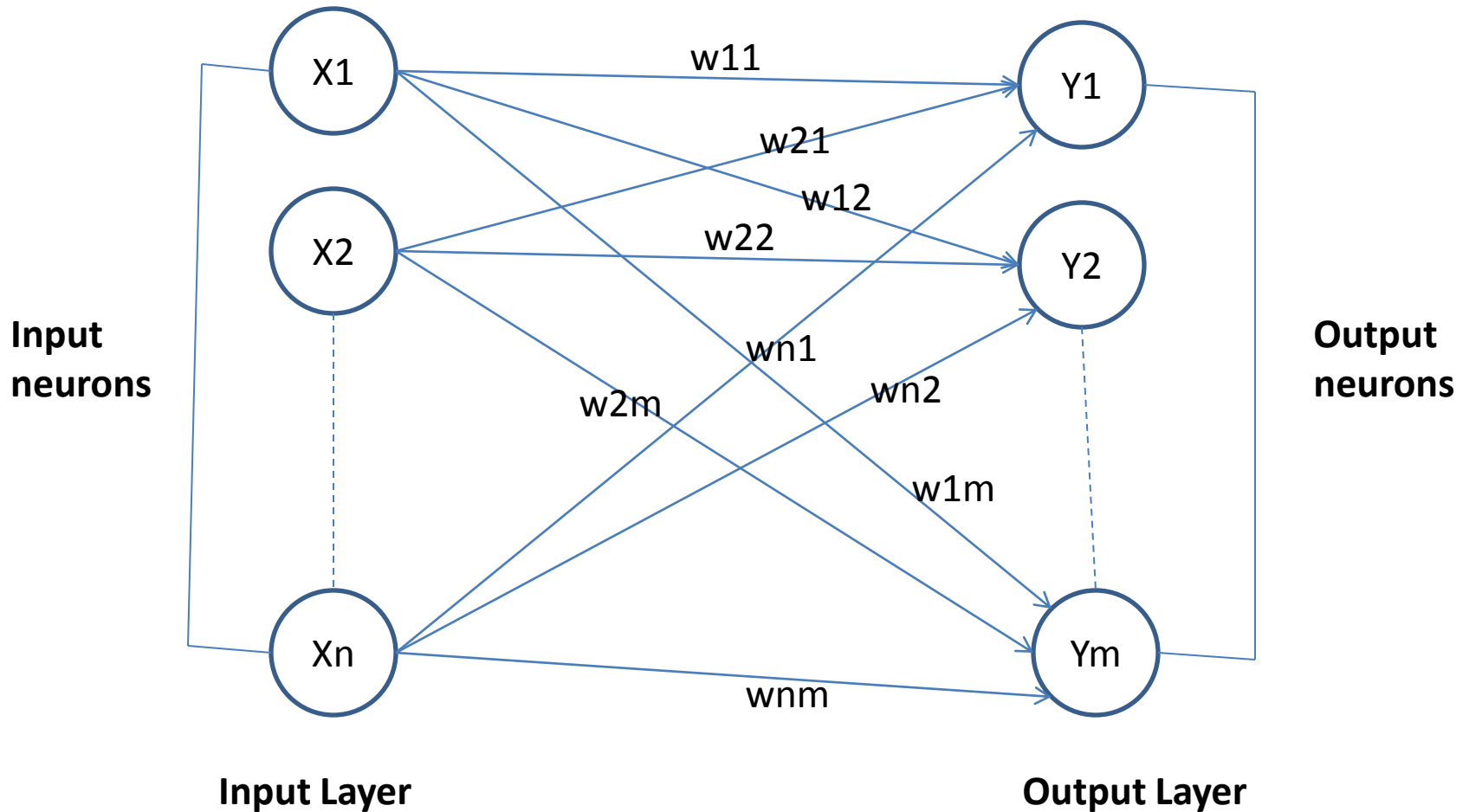
3. SINGLE NODE WITH ITS OWN FEEDBACK

4. SINGLE-LAYER RECURRENT NETWORK

5. MULTI-LAYER RECURRENT NETWORK

SINGLE LAYER FEED FORWARD NETWORK

- Architecture



DETAILED DESCRIPTIONS

- Neural nets are classified into **single-layer** or **multi-layer** neural nets
- A layer is formed by taking a **processing element** and **combining it** with other processing elements
- Practically a **layer implies a stage**, going stage by stage
- This means the **input stage** and **output stage** are linked with each other
- These **linked interconnections** lead to various network architectures

SINGLE LAYER FEED FORWARD NETWORK

- From **each node in the input layer** we have **connections** to **each of the output nodes** with various **weights**
- This architecture is called the single-layer feed forward network
- The input nodes are say $X_i, i = 1, 2, \dots, n$
- The output nodes be say $Y_j, j = 1, 2, \dots, m$
- The **connections** from 'n' input nodes to the 'm' output nodes may be **assigned with weights** be given by

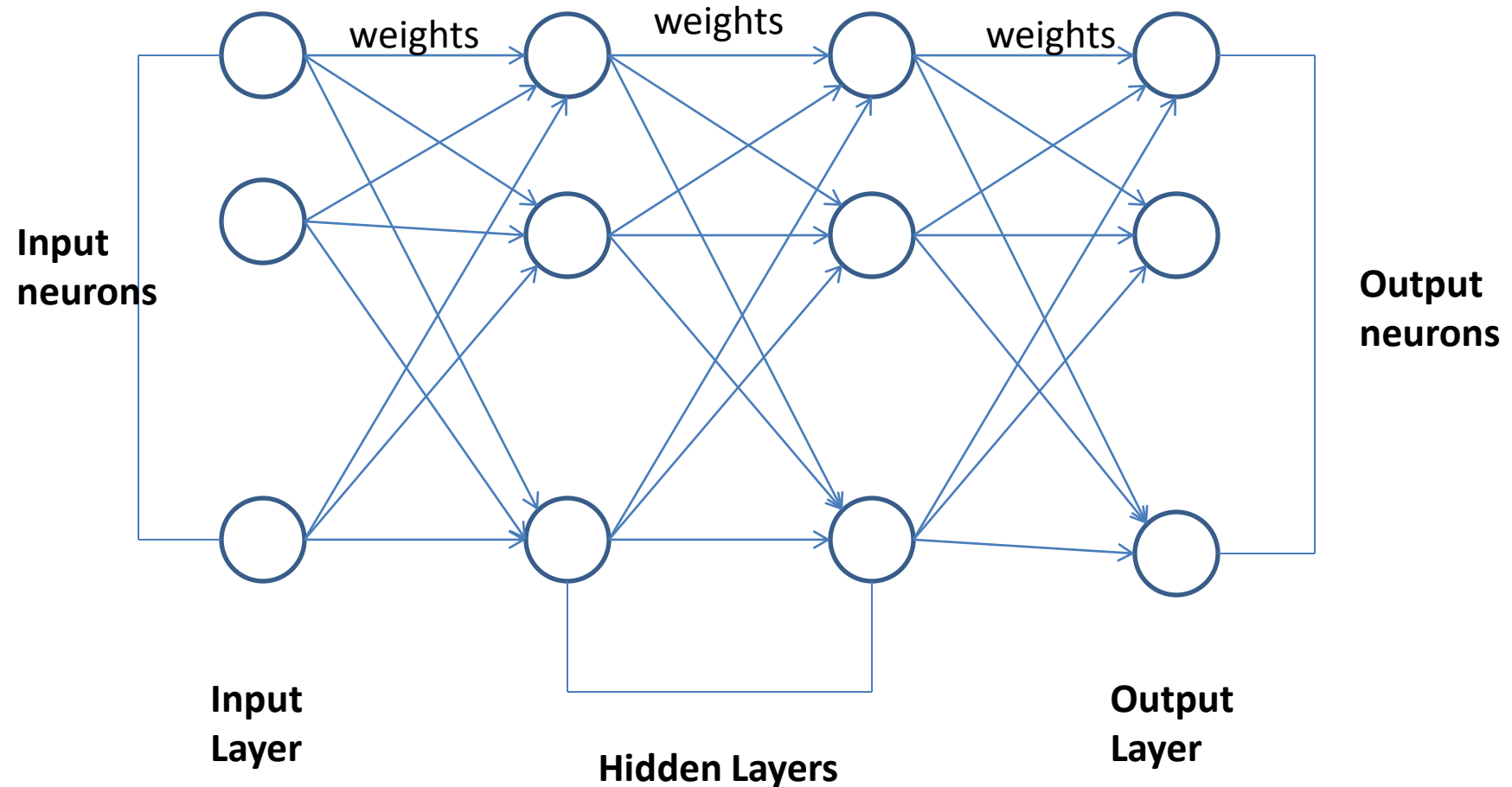
$$w_{ij}, i = 1, 2, \dots, n; j = 1, 2, \dots, m$$

MULTI-LAYER FEED FORWARD NETWORK

- It is formed by the interconnection of several layers
- Input layer is the one which receives the input
- It simply buffers the input signal
- The output layer generates the output of the network
- Any layer that is formed between the input and the output layer are called the hidden layers
- The hidden layers are internal to the network and do not have any connection with the external environment
- More the number of hidden layers more is the complexity of the network
- This may provide an efficient output response
- Every output from one layer is connected to each and every node in the next layer

MULTI-LAYER FEED FORWARD NETWORK

- Architecture

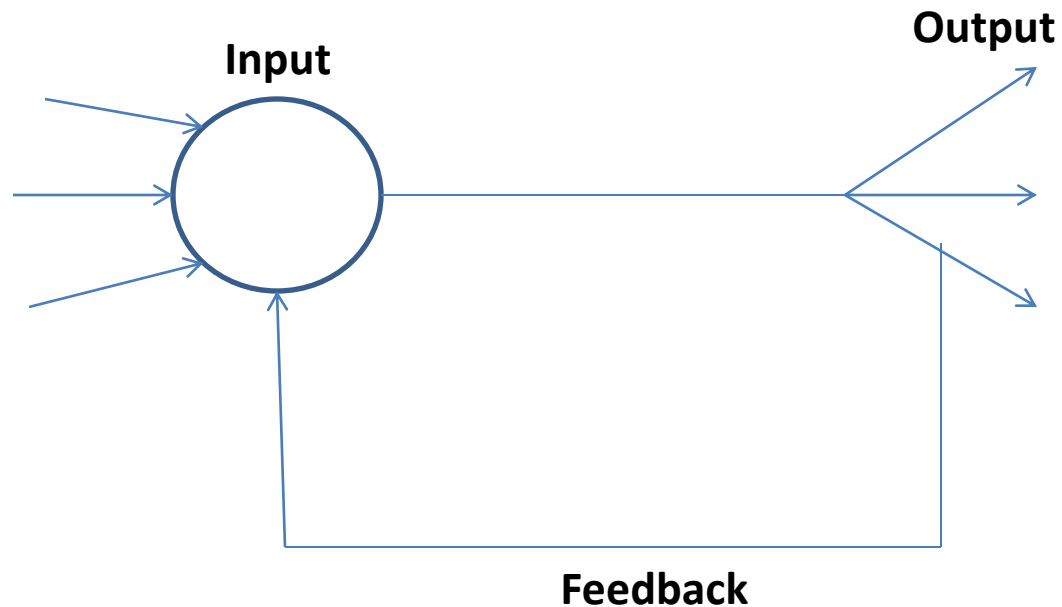


FEEDBACK NETWORKS

- Networks are said to be **feed forward** if **no neuron** in the **output layer is an input** to a node in the **same layer** or in the **preceding layer**
- When the outputs can be directed back as inputs to same or preceding layer nodes then it results in the formation of **feedback networks**
- When the feedback is directed as input to the **nodes in the same layer** it is called **lateral feedback**
- When each node gets two types of inputs; **excitatory** (from nearby processing elements) and **inhibitory** (from most distinctly located processing elements) it is called **lateral inhibition structure network**

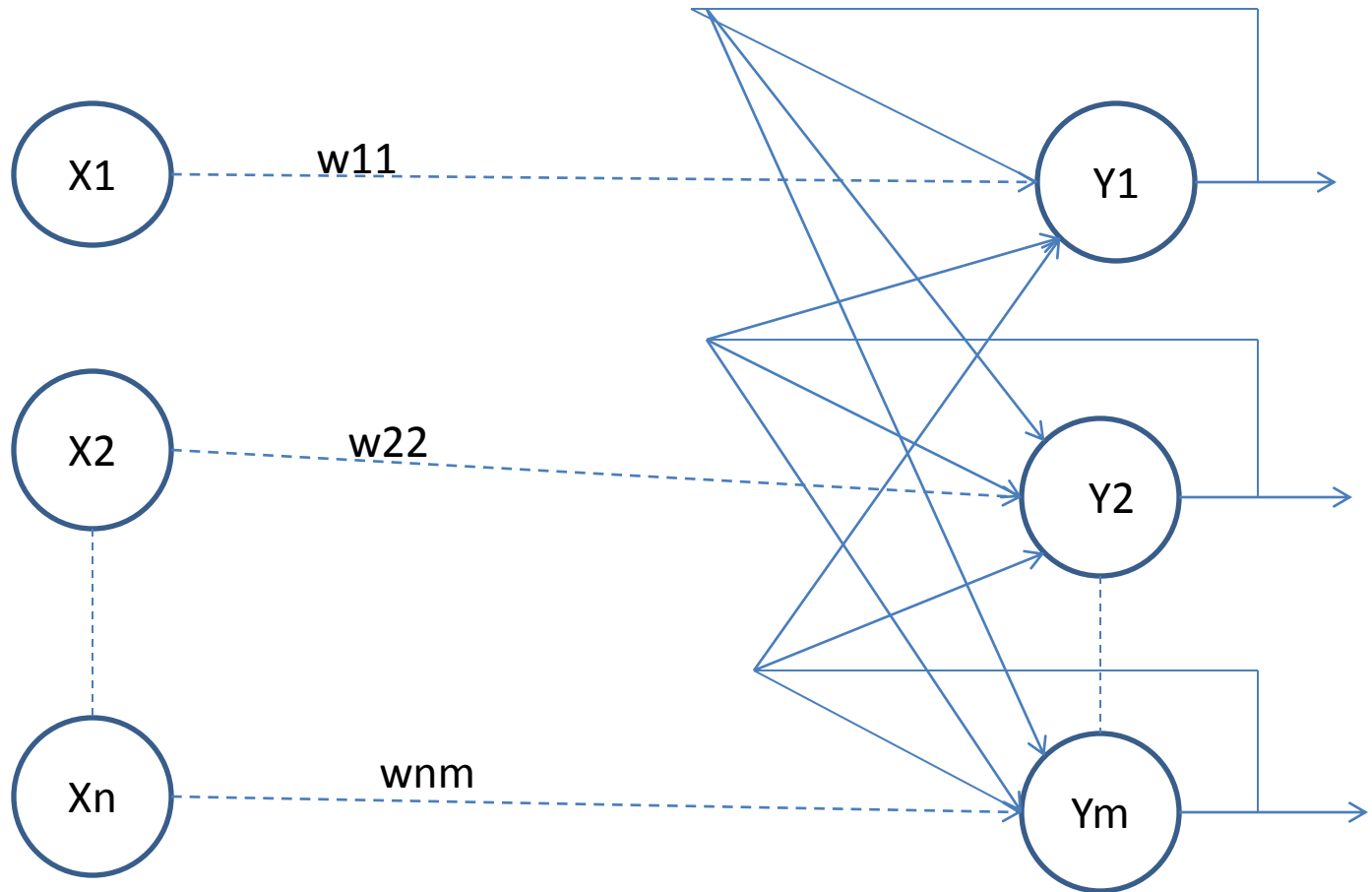
SINGLE NODE WITH ITS OWN FEEDBACK

- Sn.



SINGLE LAYER RECURRENT NETWORK

- slr



LEARNING

- Main property of ANN is learning
- There are **two kinds of learning** in ANNs
- **Parameter learning**
 - It **updates the connecting weights** in a neural network
- **Structure Learning**
 - It focuses on the **change in structure** in the network
 - Structure:**
 - Number of processing elements
 - The types of connection between nodes

LEARNING CATEGORIES

- **Supervised learning:**

Learning **through** a teacher/ supervisor

- **Unsupervised learning:**

Learning **without** a teacher /supervisor

- **Reinforcement learning:**

Learning basing upon a **critic information**

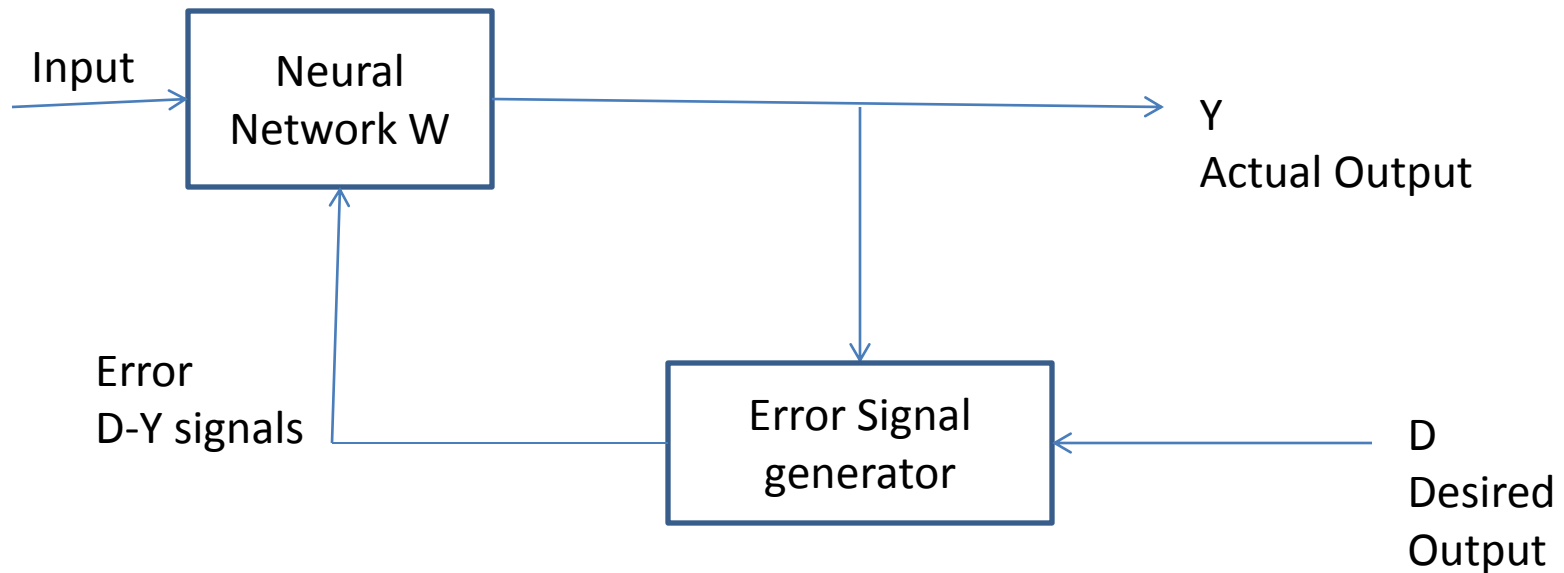
It is similar to supervised learning

SUPERVISED LEARNING

- **Example:** A child learning with the help of a teacher or parents
- Here each and every action of the child is supervised by a teacher
- The child works on the basis of the output that he/she has to produce
- In ANN context, each input vector requires a corresponding target vector, which represents the output vector
- **Training pair:**
- A pair of input vector and the output vector

SUPERVISED LEARNING CONTD...

- During training an input vector generates a actual output vector
- This actual output vector is compared with the desired output vector



SUPERVISED LEARNING CONTD...

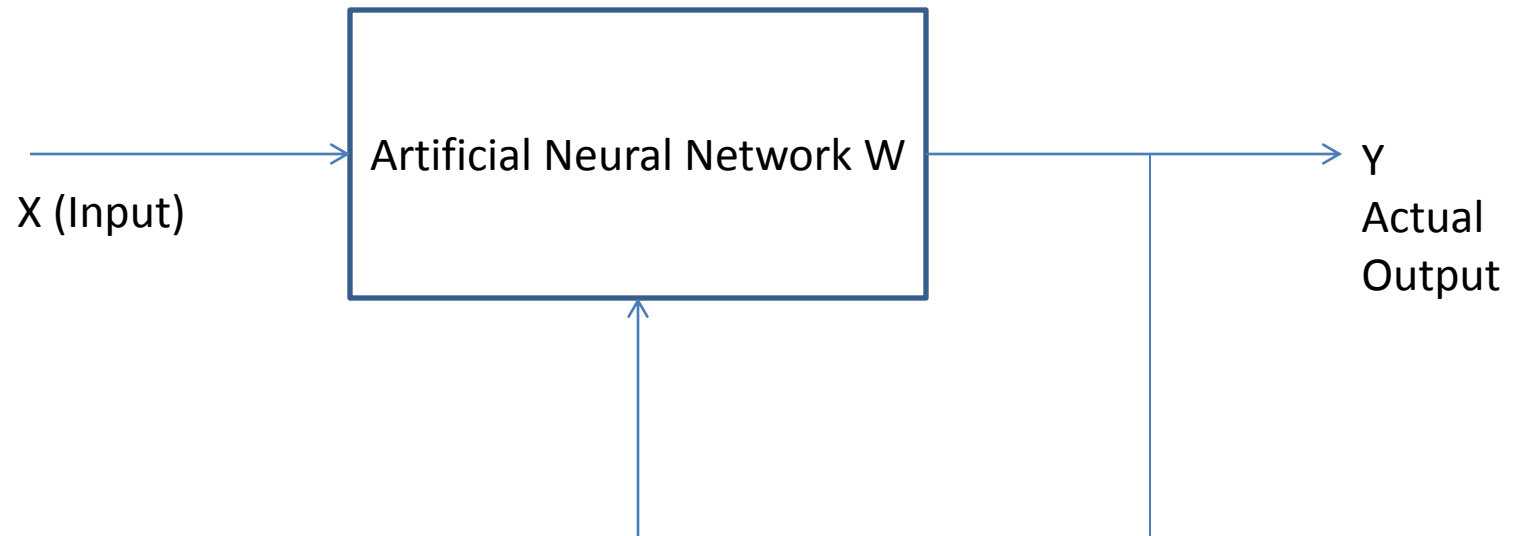
- If there is a difference between the two vectors, then an error signal is generated by the network
- This error signal is used for adjustment of weights until the output matched the desired output
- NOTE: It is assumed that correct target output values are known for each input pattern

UNSUPERVISED LEARNING

- **Example:** A fish learns swimming by itself, not taught by its mother
- In the context of ANN
- The input vectors of similar type are grouped without the use of training data to specify how a member of each group looks or to which group a member belongs
- In the training process the network receives the input patterns and organises these patterns into clusters
- When a new pattern arrives the ANN gives an output response indicating the class to which the pattern belongs

UNSUPERVISED LEARNING CONTD...

- If for a pattern no class could be found then a new class is formulated

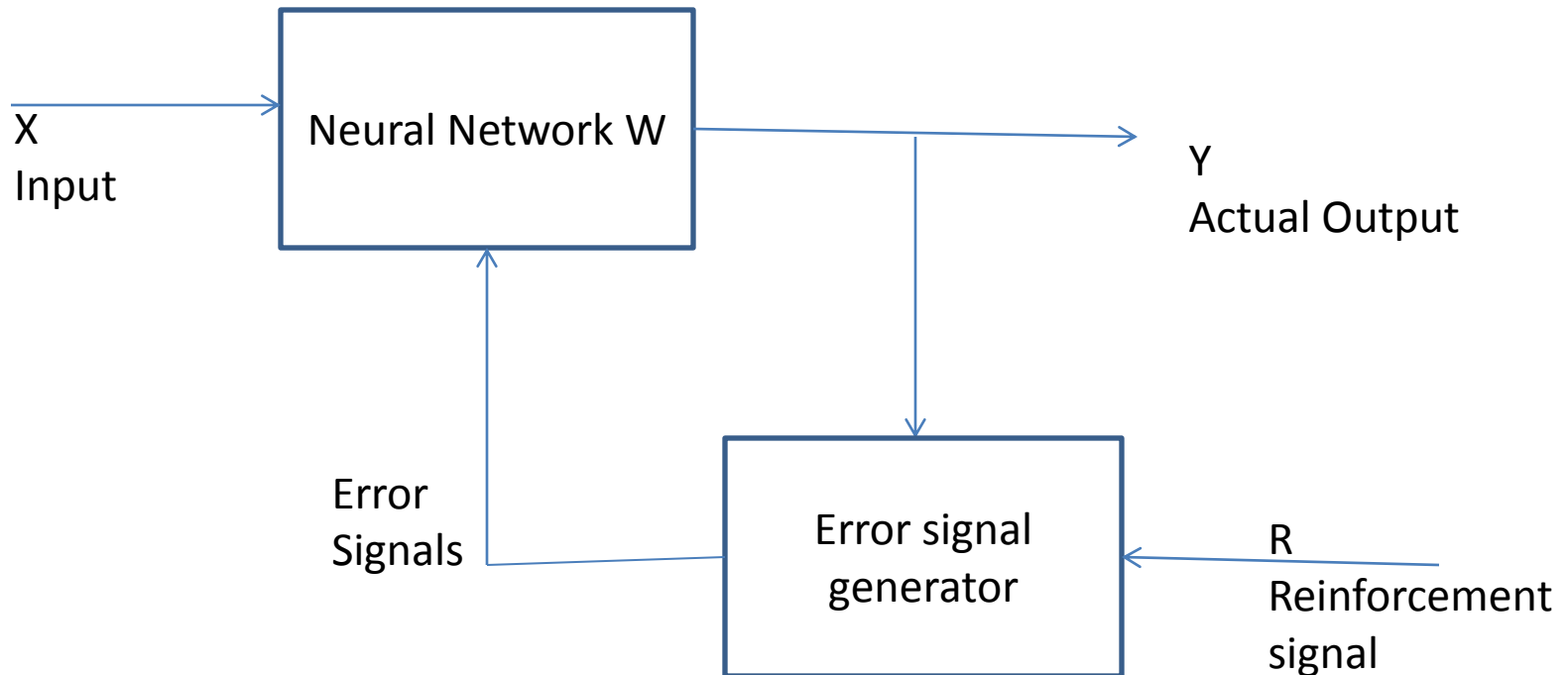


REINFORCEMENT LEARNING

- Similar to supervised learning
- In some cases instead of the exact output vector less information may be available
- This information is called “critic information”
- The learning based upon critic information is called reinforcement learning and the feedback sent is called “reinforcement signal”
- The external reinforcement signals are processed in the critic signal generator and the obtained critic signals are sent to the ANN for adjustment of weights properly

REINFORCEMENT LEARNING CONTD...

- Adjustment of weights are done by the ANN so as to get better critic feedback in future



ACTIVATION FUNCTIONS

- It helps in achieving the exact output
- The **activation function is applied over the net input** to calculate the output of an ANN
- In a processing element in an ANN comprises of **two major parts**: the input and the output
- Mostly non-linear functions are used to achieve the advantages of a multilayer network over a single layer network
- If a linear activation function is used in a multilayer ANN then the output remains same as that obtained from a single layer network

TYPES OF ACTIVATION FUNCTIONS

- **Identity function:** It is a linear function $f(x) = x$ for all x
- : The output remains the same as the input
- **Binary step function:** This function is mostly used in single layer ANN to convert the net input into a binary output

- $$f(x) = \begin{cases} 1, & \text{if } x \geq \theta; \\ 0, & \text{if } x < \theta. \end{cases}$$

- Here θ represents the threshold value
- Bipolar step function: This function is also used in single layer ANN to convert the net input to an output that is bipolar (+1 or -1)

$$f(x) = \begin{cases} 1, & \text{if } x \geq \theta; \\ -1, & \text{if } x < \theta. \end{cases}$$

TYPES OF ACTIVATION FUNCTIONS CONTD...

- **SIGMOIDAL FUNCTIONS:** These are used in back propagation ANN because of the relationship between the value of the functions at a point and the value of the derivative at that point which reduces the computational burden during training
- **Binary sigmoid function:** (Also called as **logistic sigmoid function** or **unipolar sigmoid function**)

- $$f(x) = \frac{1}{1 + e^{-\lambda x}}, \quad \text{where } \lambda \text{ is the steepness parameter}$$

- Derivative is $f'(x) = \lambda \cdot f(x)[1 - f(x)]$
- The range of this function is 0 to 1.

BIPOLAR SIGMOID FUNCTION

- This function is defined as
- $f(x) = \frac{1 - e^{-\lambda x}}{1 + e^{-\lambda x}}$, where λ is the steepness parameter
- The value of this function lies between -1 and +1
- $f'(x) = \lambda / 2[1 + f(x)][1 - f(x)]$

HYPERBOLIC TANGENT FUNCTION

- It is closely related to the bipolar sigmoid function

- $$h(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$$h(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$

- The derivative of this function is given by

$$h'(x) = [1 + h(x)][1 - h(x)]$$

-

RAMP FUNCTION

- The ramp function is given by

$$f(x) = \begin{cases} 1, & \text{if } x > 1; \\ x, & \text{if } 0 \leq x \leq 1; \\ 0, & \text{if } x < 0. \end{cases}$$

IMPORTANT TERMINOLOGIES

- **Weights:**
- In ANN architecture each neuron is connected to other neurons by means of directed communication links
- Each communication link is associated with a weight
- The weights contain information about input signal
- This information is used by the ANN to solve a problem
- The weight can be represented by a matrix
- This matrix is called the **connection matrix**

THE CONNECTION MATRIX

- Suppose there are 'n' processing elements in an ANN and each processing element has exactly 'm' adaptive weights
- The weight matrix W is defined by

$$W = \begin{bmatrix} w_1^T \\ w_2^T \\ . \\ . \\ w_n^T \end{bmatrix} \begin{bmatrix} w_{11} & w_{12} & . & . & w_{1m} \\ w_{21} & w_{22} & . & . & w_{2m} \\ . & . & . & . & . \\ . & . & . & . & . \\ w_{n1} & w_{n2} & . & . & w_{nm} \end{bmatrix}$$

WEIGHTS CONTD...

- The vector $w_i = [w_{i1}, w_{i2}, \dots, w_{im}]^T, i = 1, 2, \dots, n$ is the weight vector of the processing element
- w_{ij} is the weight from the processing element 'i' to the processing element 'j'
- The set of all weight matrixes will determine the set of all possible information processing configurations for the ANN
- The ANN can be realised by finding an appropriate matrix W
-

BIAS

- The bias included in the ANN has its impact in calculating the net input
- The bias is included by adding a component $x_0 = 1$ to the input vector X
- The vector becomes $X = (1, x_1, x_2, \dots, x_n)$
- The bias is considered like another weight $w_{0j} = b_j$
- Thus

$$(y_{in})_j = \sum_{i=0}^n x_i w_{ij} = b_j + \sum_{i=1}^n x_i w_{ij}$$

THRESHOLD

- Threshold is a pre-defined value based upon which the final output of the ANN is to be computed
- It is used in the activation function
- A comparison is made between the net input and the threshold to obtain the net output
- For each and every application there is a threshold
- The activation functions are defined in terms of the threshold values and the outputs are computed

- Ex.

$$f(\text{input}) = \begin{cases} 1, & \text{if } \text{input} \geq \theta; \\ -1, & \text{if } \text{input} < \theta. \end{cases}$$

LEARNING RATE

- The learning rate is denoted by ' α '
- It is used to control the amount of weight adjustment at each step of training
- It ranges from 0 to 1
- Determines the rate of learning at every step

MOMENTUM FACTOR

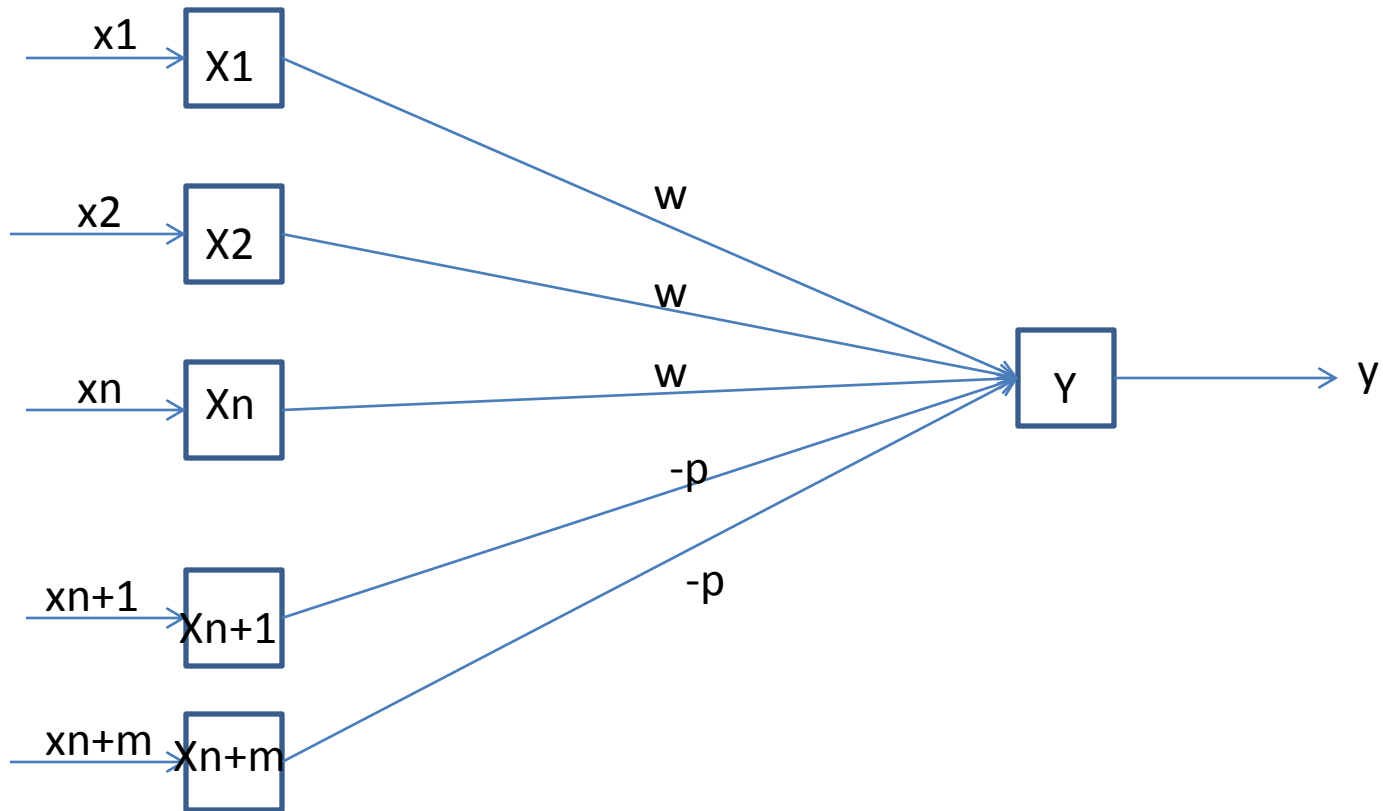
- The **convergence is made faster** if a **momentum factor** is **added to the weight updation process**
- This is **generally done in the back propagation** ANN
- For momentum to be used, **the weights from one or more of previous training patterns must be saved**
- Momentum **helps the net** in reasonably **large weight adjustments** until the **corrections are in the same general direction for several patterns**

McCulloch-Pitts Neuron

- It is the earliest neuron discovered in 1943
- Popularly called M-P neuron
- Here, the neurons are connected by directed weighted paths
- Activation function for M-P neuron is binary
- This means at every time step the neuron may or may not fire
- The weights attached with the communication links may be excitatory (+ve) or inhibitory (-ve)
- All the excitatory connection weights are same in value
- The threshold plays a major role here
- If the net input to the neuron is greater than the threshold then the neuron fires
- Any non-zero inhibitory input would prevent the neuron from firing

THE ARCHITECTURE

- The M-P neuron model architecture



M-P NEURON CONTD...

- It has excitatory weight 'w'
- It has inhibitory weight '-p'
- The activation function is defined by

$$f(y_{in}) = \begin{cases} 1, & \text{if } y_{in} \geq \theta; \\ 0, & \text{if } y_{in} < \theta. \end{cases}$$

- The output will fire if it receives 'k' or more excitatory inputs but no inhibitory inputs, where

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

- It is used as building blocks on which we can model any function or phenomenon, which can be represented as a logic function

LINEAR SEPARABILITY

- An ANN does not provide exact solution for a nonlinear problems
- It provides possible approximation solutions to nonlinear problems
- Linear separability is the concept wherein the separation of the input space into regions is based on whether the network response is positive or negative
- A decision line is drawn to separate positive and negative responses
- Decision line = decision-making line = decision-support line = linear-separable line

LINEAR SEPARABILITY CONTD...

- We have the formula for the net input given by

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

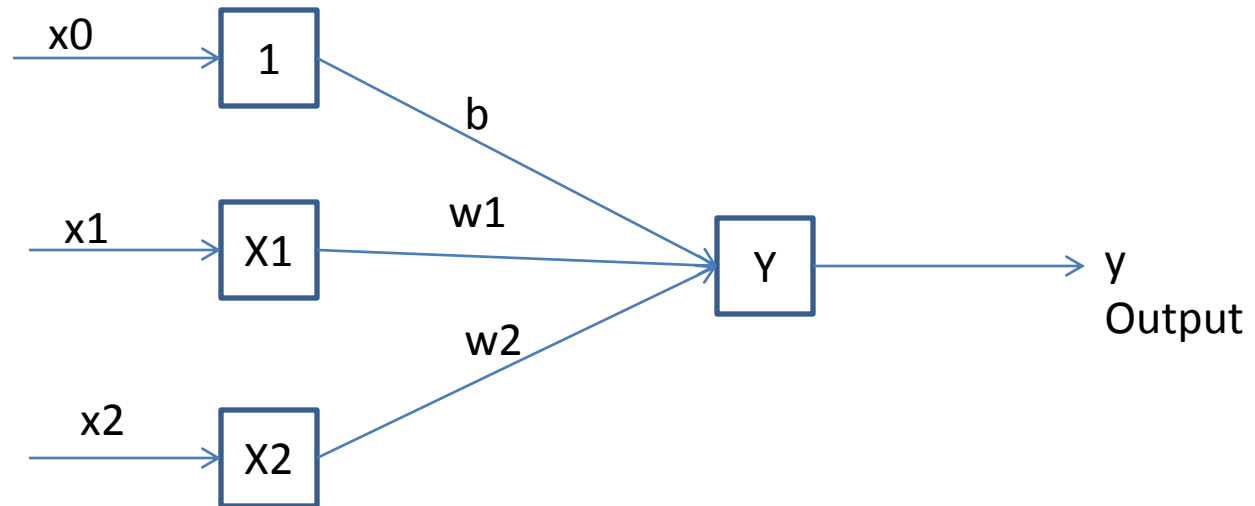
- It is clear that there exists a boundary between the regions where $y_{in} > 0$ and $y_{in} < 0$
- The region may be called as decision boundary and can be determined by the relation
$$b + \sum_{i=1}^n x_i w_i = 0$$
- Basing upon the number of input units in the ANN the above equation may represent a **line** or **plane** or a **hyperplane**

LINEAR SEPARABILITY CONTD...

- The **linear separability** of the network is **based on the decision boundary line**
- If there exist weights for which the training input vectors having positive response, +1, **lie on one side of the decision boundary** and all the other vectors having negative response, -1, **on the other side of the decision boundary** then we can conclude that the problem is **“linearly separable”**
- Let us consider the ANN in the next slide

EXAMPLE OF LINEAR SEPARABILITY

- Consider the single layer neural network



- The net input is $y_{in} = b + x_1 \cdot w_1 + x_2 \cdot w_2$
- The separating line is $b + x_1 \cdot w_1 + x_2 \cdot w_2 = 0$

EXAMPLE OF LINEAR SEPARABILITY

- The requirements for positive response of the ANN is

$$b + x_1.w_1 + x_2.w_2 > 0$$

- During the training process, the values of w_1, w_2, b are determined so that the ANN will produce a positive response for the training data
-

HEBB NETWORK

- The Hebb learning rule was put forth by Donald Hebb in 1949
- Statement:
- **When an axon of cell A is near enough to excite cell B and repeatedly or permanently takes place in firing it, then some growth process or metabolic change takes place in one or both the cells such that A's efficiency, as one of the cells firing B is increased**
- The weight update in Hebb rule is given by
$$w_i(new) = w_i(old) + x_i \cdot y$$
- The Hebb rule is more suited for bipolar data than binary data

HEBB NETWORK

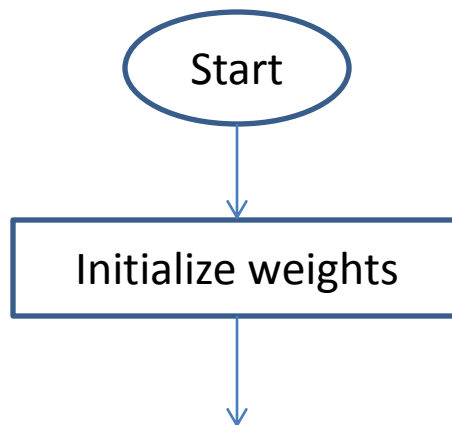
- If binary data is used then the two following cases cannot be distinguished
- Case-1: A training pair in which an input unit is 'on' and the target value is 'off'
- Case-2: A training pair in which both the input and the target value are 'off'
- **The Training Algorithm for Hebb ANN:**
- **STEP-1:** first initialize the weights. In this network they may be set to 0. ($w_i = 0$, $i = 1, 2, \dots, n$)
- **STEP-2:** Steps 2 to 4 have to be performed for each input training vector and target output pair (s:t)

TRAINING ALGORITHM CONTD...

- **STEP-2:** Input units activations are set. Generally the activation function of input layer is identity function
 - $x_i = s_i, \text{ for } i = 1, 2, \dots, n$
 - **STEP-3:** Output activations are set: $y = t$
 - **STEP-4:** Weight adjustments and bias adjustments are performed
- $$w_i(\text{new}) = w_i(\text{old}) + x_i \cdot y$$
- $B(\text{new}) = b(\text{old}) + y$

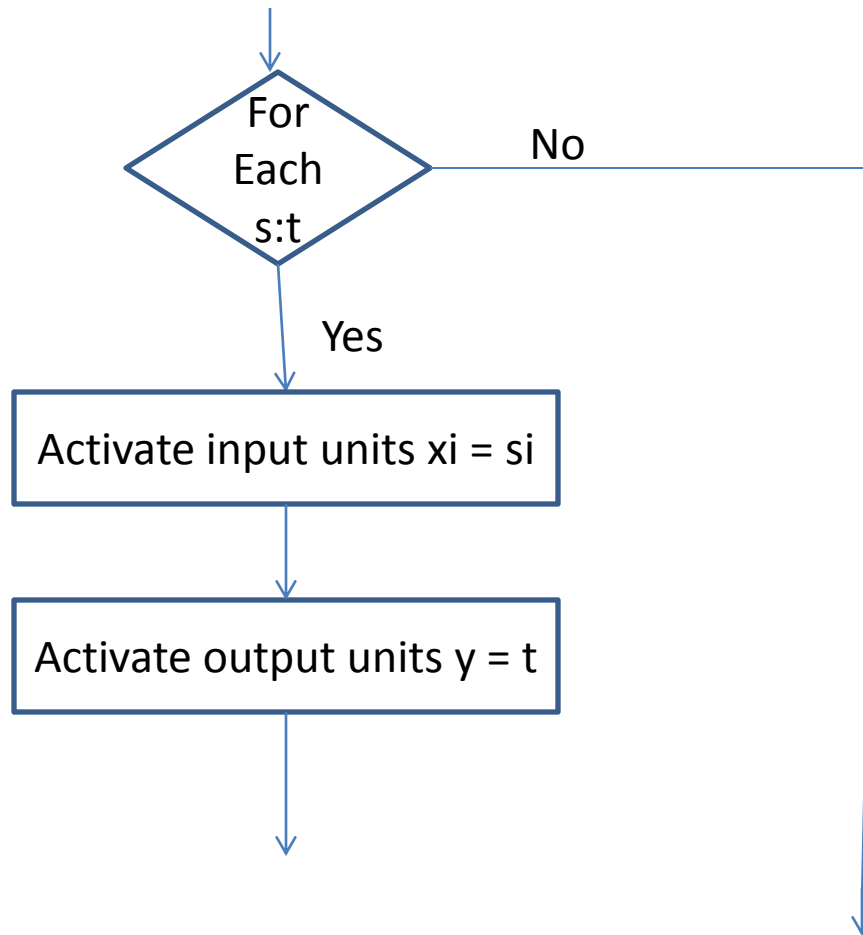
TRAINING ALGORITHM CONTD...

- Weight formula can be given in the vector form as
- $w(\text{new}) = w(\text{old}) + x.y$
- Change in weight can be represented as $\Delta w = x.y$
- So, the vector form of the weight formula can be written as
- $W(\text{new}) = w(\text{old}) + \Delta w$
- The flow chart is



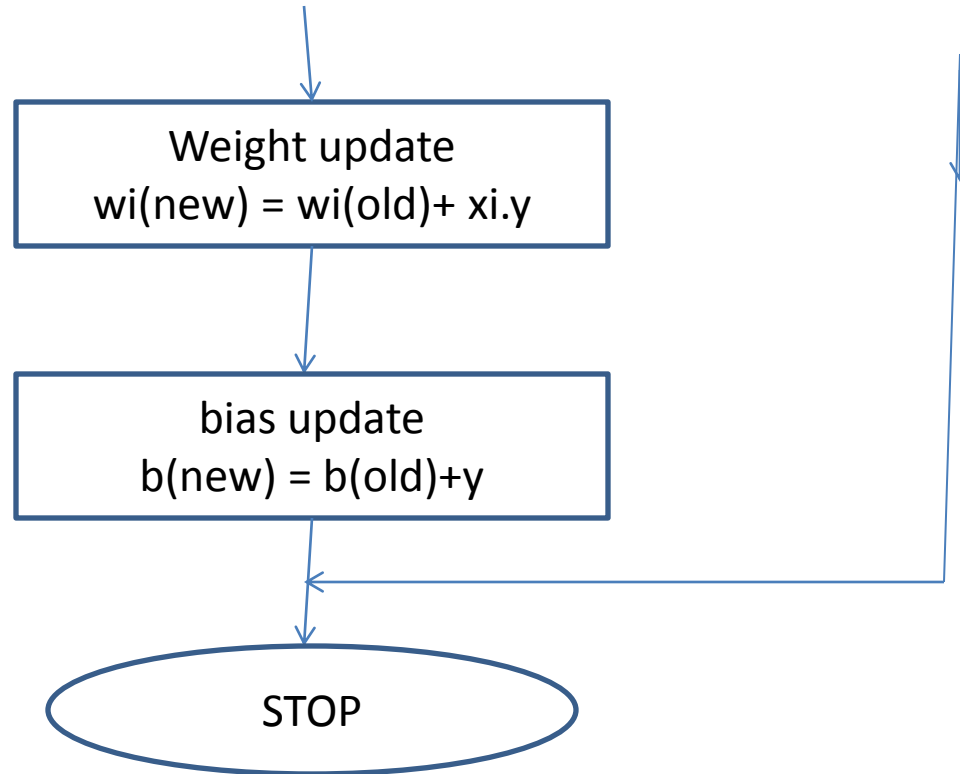
FLOW CHART CONTD...

- Contd.



FLOW CHART CONTD...

- Contd..



EXAMPLE FOR M-P ANN

- Implement AND function using M-P Neuron
- We have the truth table for the AND function given by

x1	x2	y
1	1	1
1	0	0
0	1	0
0	0	0

EXAMPLE CONTD...

- Let us assume that the two weights be $w_1=1$, $w_2=1$
- Here

$$y_{in}(1,1) = x_1 \cdot w_1 + x_2 \cdot w_2 = 2$$

$$y_{in}(1,0) = x_1 \cdot w_1 + x_2 \cdot w_2 = 1$$

$$y_{in}(0,1) = x_1 \cdot w_1 + x_2 \cdot w_2 = 1$$

$$y_{in}(0,0) = x_1 \cdot w_1 + x_2 \cdot w_2 = 0$$

- For an AND input the output is high if both the inputs are high
- We set the threshold basing upon the net input
- So, $\theta = 2$. Thus, the output of neuron Y can be written as

$$y = f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} \geq 2; \\ 0 & \text{if } y_{in} < 2. \end{cases}$$

EXAMPLE CONTD...

- The computation of the value of θ can be seen as follows:
- If there are 'n' excitatory weights of value 'w' and 'm' inhibitory weights of value 'p' then we should have

$$\theta \geq n.w - m.p$$

- Here, $n = 2$, $w = 1$ and $m = 0$. So, $\theta \geq 2 \times 1 - 0 \times p$ or $\theta \geq 2$
- So, we take the threshold to be 2

EXAMPLE FOR HEBB NETWORK

- Design a Hebb net to implement logical AND function by using bipolar inputs and targets
- The training data for the logical AND function is

Inputs-----→	->	->	Target
x1	x2	b	y
1	1	1	1
1	-1	1	-1
-1	1	1	-1
-1	-1	1	-1

EXAMPLE CONTD...

- Using the Hebb Network training algorithm:
- Initially the weights and bias are zeros. So, we have

$$w_1 = w_2 = b = 0$$

- **First Input:** $[x_1 \ x_2 \ b] = [1 \ 1 \ 1]$ Target: 1 (i.e. $y = 1$)
- Using Hebb Rule: $w_i(\text{new}) = w_i(\text{old}) + x_i \cdot y$
- We have
$$w_1(\text{new}) = w_1(\text{old}) + x_1 \cdot y = 0 + 1 \times 1 = 1$$
$$w_2(\text{new}) = w_2(\text{old}) + x_2 \cdot y = 0 + 1 \times 1 = 1$$
$$b(\text{new}) = b(\text{old}) + y = 0 + 1 = 1$$
- These inputs are used in the next iteration
- Here, $\Delta w_1 = x_1 \cdot y = 1 \times 1 = 1, \Delta w_2 = x_2 \cdot y = 1 \times 1 = 1, \Delta b = y = 1$

EXAMPLE CONTD...

- **Second Input:** $[x_1 \ x_2 \ b] = [1 \ -1 \ 1]$ target: $y = -1$

- The weight changes are:

$$\Delta w_1 = x_1 \cdot y = 1 \times -1 = -1, \Delta w_2 = x_2 \cdot y = -1 \times -1 = 1, \Delta b = y = -1$$

- The new weights here are:

$$w_1(\text{new}) = w_1(\text{old}) + \Delta w_1 = 1 - 1 = 0$$

$$w_2(\text{new}) = w_2(\text{old}) + \Delta w_2 = 1 + 1 = 2$$

$$b(\text{new}) = b(\text{old}) + \Delta b = 1 - 1 = 0$$

- **Third Input:** $[x_1 \ x_2 \ b] = [-1 \ 1 \ 1]$ target: $y = -1$

- The weight changes are:

$$\Delta w_1 = x_1 \cdot y = -1 \times -1 = 1, \Delta w_2 = x_2 \cdot y = 1 \times -1 = -1, \Delta b = y = -1$$

EXAMPLE CONTD...

- The new weights are:

$$w_1(new) = w_1(old) + \Delta w_1 = 0 + 1 = 1$$

$$w_2(new) = w_2(old) + \Delta w_2 = 2 - 1 = 1$$

$$b(new) = b(old) + \Delta b = 0 - 1 = -1$$

- **Fourth input:** $[x_1 \ x_2 \ b] = [-1 \ -1 \ 1]$ target: $y = -1$

- The weight changes are:

$$\Delta w_1 = x_1 \cdot y = -1 \times -1 = 1, \Delta w_2 = x_2 \cdot y = -1 \times -1 = 1, \Delta b = y = -1$$

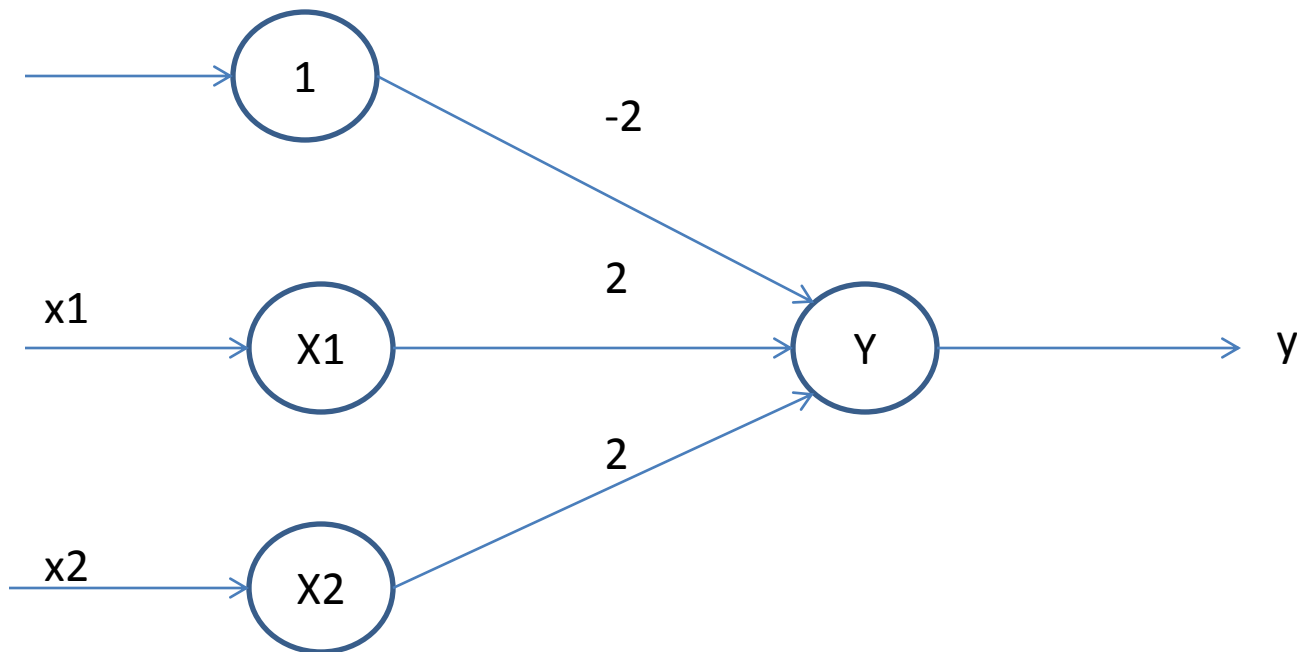
- The new weights are: $w_1(new) = w_1(old) + \Delta w_1 = 1 + 1 = 2$

$$w_2(new) = w_2(old) + \Delta w_2 = 1 + 1 = 2$$

$$b(new) = b(old) + \Delta b = -1 - 1 = -2$$

EXAMPLE CONTD...

- Thus the Hebb net for the AND function is:



SUPERVISED LEARNING NETWORK (PERCEPTRON NETWORKS)

- **Perceptron Networks** come under single-layer feed forward networks
- These are also called **simple perceptrons**
- **PERCPTRONS ARE THE PATTERN RECOGNISERS**
- Designed by many researchers:
- **Rosenblatt (1962)**
- **Minsky- Papert (1969, 1988)**
- **Block (1962)**
- The design of **Block** is the **simplest**

PERCEPTRON NETWORK ARCHITECTURE

- Consists of 3 units
- **Sensory unit** (input unit)
- **Associator unit** (hidden unit)
- **Response unit** (output unit)
- The **sensory units** are connected to **associator units** with **fixed weights having values 1, 0 or -1**, assigned at random
- The binary activation function is used in sensory unit and associator unit
- The **response unit has an activation** of 1, 0 or -1
- The binary step with **fixed threshold θ** is used as an **activation for the associator**

PERCEPTRON NETWORK ARCHITECTURE

- The **output signals** that are sent from the **associator unit** to the **response unit** are **only binary**
- The **output** of the perceptron network is given by $y = f(y_{in})$
- The **activation function** $f(y_{in})$ is given by

$$f(y_{in}) = \begin{cases} 1, & \text{if } y_{in} > \theta; \\ 0, & \text{if } -\theta \leq y_{in} \leq \theta; \\ -1 & \text{if } y_{in} < -\theta. \end{cases}$$

- **Perceptron learning rule** is used in the weight updation between the associator unit and the response unit .
- For each training input, the net will calculate the response and it will **determine whether or not an error has occurred**

EXAMPLE CONTD...

- Using the Hebb Network training algorithm:
- Initially the weights and bias are zeros. So, we have

$$w_1 = w_2 = b = 0$$

- First Input: $[x_1 \ x_2 \ b] = [1 \ 1 \ 1]$ Target: 1 (i.e. $y = 1$)
- Using Hebb Rule: $w_i(\text{new}) = w_i(\text{old}) + x_i \cdot y$
- We have
$$w_1(\text{new}) = w_1(\text{old}) + x_1 \cdot y = 0 + 1 \times 1 = 1$$
$$w_2(\text{new}) = w_2(\text{old}) + x_2 \cdot y = 0 + 1 \times 1 = 1$$
$$b(\text{new}) = b(\text{old}) + y = 0 + 1 = 1$$
- These inputs are used in the next iteration
- Here, $\Delta w_1 = x_1 \cdot y = 1 \times 1 = 1, \Delta w_2 = x_2 \cdot y = 1 \times 1 = 1, \Delta b = y = 1$

PERCEPTRON NETWORK ARCHITECTURE

- The **error calculation** is based on the comparison of the values of targets with those of the calculated outputs
- The **weights on the connections** from the units that send the nonzero signal will get adjusted suitably
- The **weights will be adjusted** on the basis of the learning rule if an error has occurred for a particular training pattern;

$$w_i(\text{new}) = w_i(\text{old}) + \alpha \cdot t \cdot x_i$$

$$b(\text{new}) = b(\text{old}) + \alpha \cdot t$$

- Here, the **target value 't'** is equal to +1 or -1
- α is the **learning rate**
- **If there are no errors then the training process is stopped**

PERCEPTRON LEARNING RULE

- The **learning signal** is the difference between the desired and actual response of a neuron (i.e. the **error signal**)
- A **sensory unit generates** signals for the **associator unit**
- The weights between sensory unit and associator unit are fixed (**cannot be changed**)
- These weights are chosen arbitrarily from 1, 0 and -1
- Only the **weights between associator unit and response unit can be changed**
- Consider a finite number 'N' of pairs of input training vectors and associated target vectors, represented by $x(n)$ and $t(n)$, $n = 1, 2, \dots, N$

PERCEPTRON LEARNING RULE CONTD...

- The target is bipolar with values +1 or -1
- The output 'y' is obtained on the basis of the net input y_{in} computed, the threshold value θ and the activation function 'f' as follows:

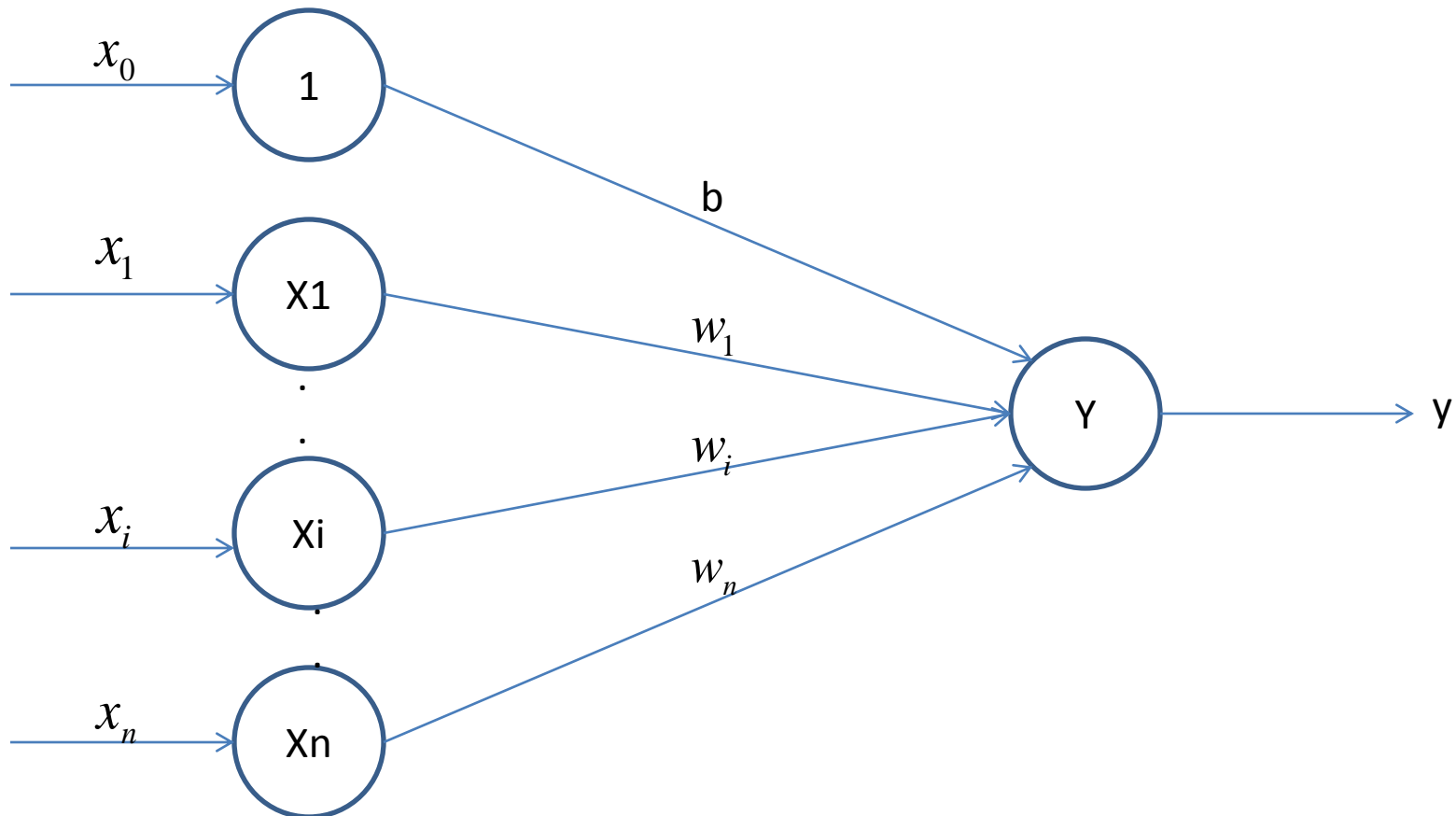
$$y = f(y_{in}) = \begin{cases} 1, & \text{if } y_{in} > \theta; \\ 0, & \text{if } -\theta \leq y_{in} \leq \theta; \\ -1, & \text{if } y_{in} < -\theta \end{cases}$$

- The weight updation is done as follows:

$$w(new) = \begin{cases} w(old) + \alpha.t.x, & \text{if } y \neq t; \\ w(old), & \text{otherwise.} \end{cases}$$

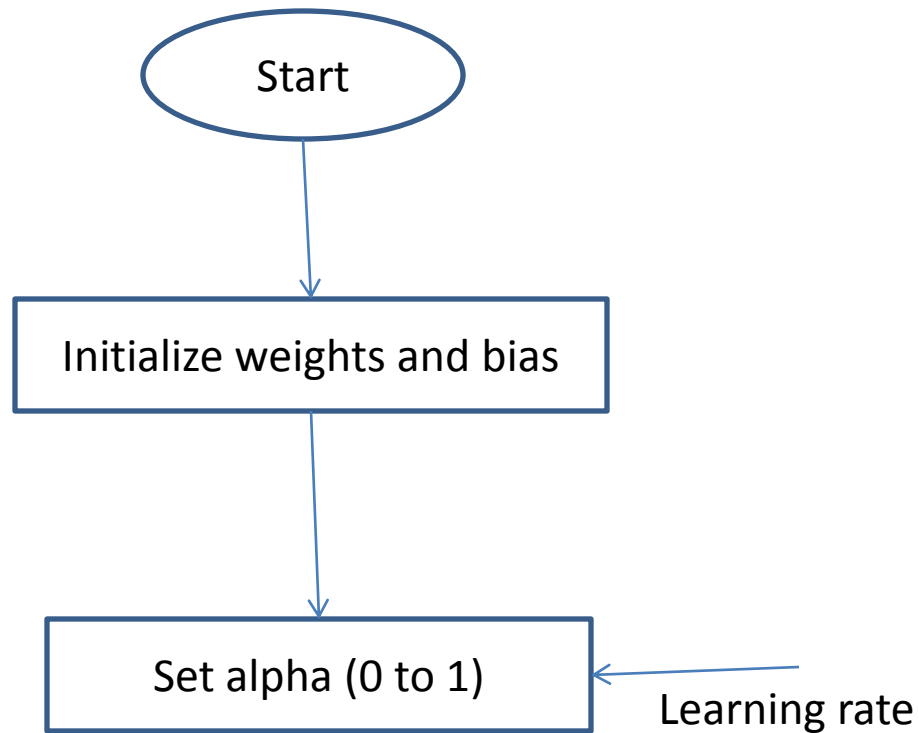
SINGLE CLASSIFICATION PERCEPTRON NETWORK ARCHITECTURE

- arc



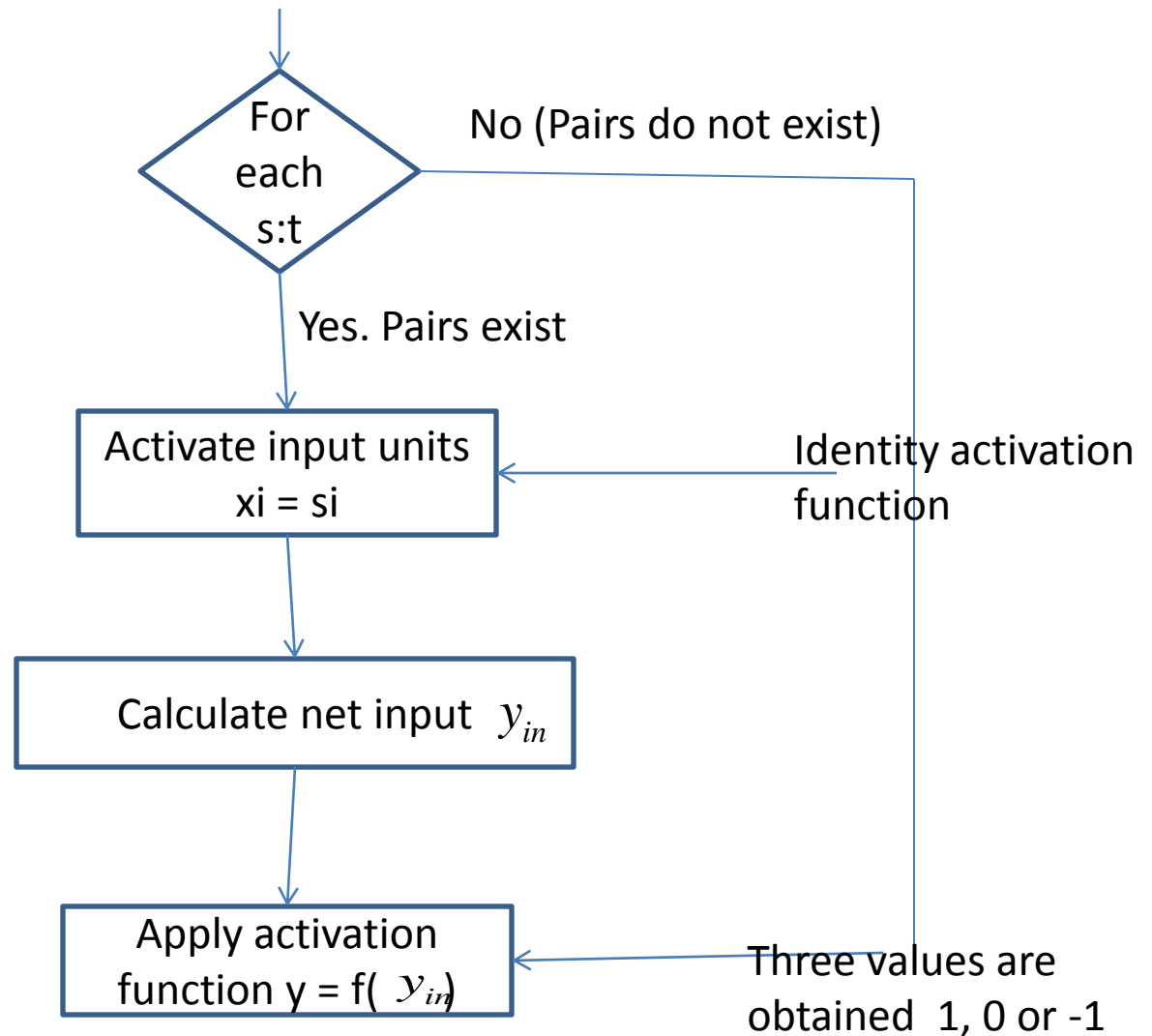
FLOW CHART FOR TRAINING PROCESS

- flow



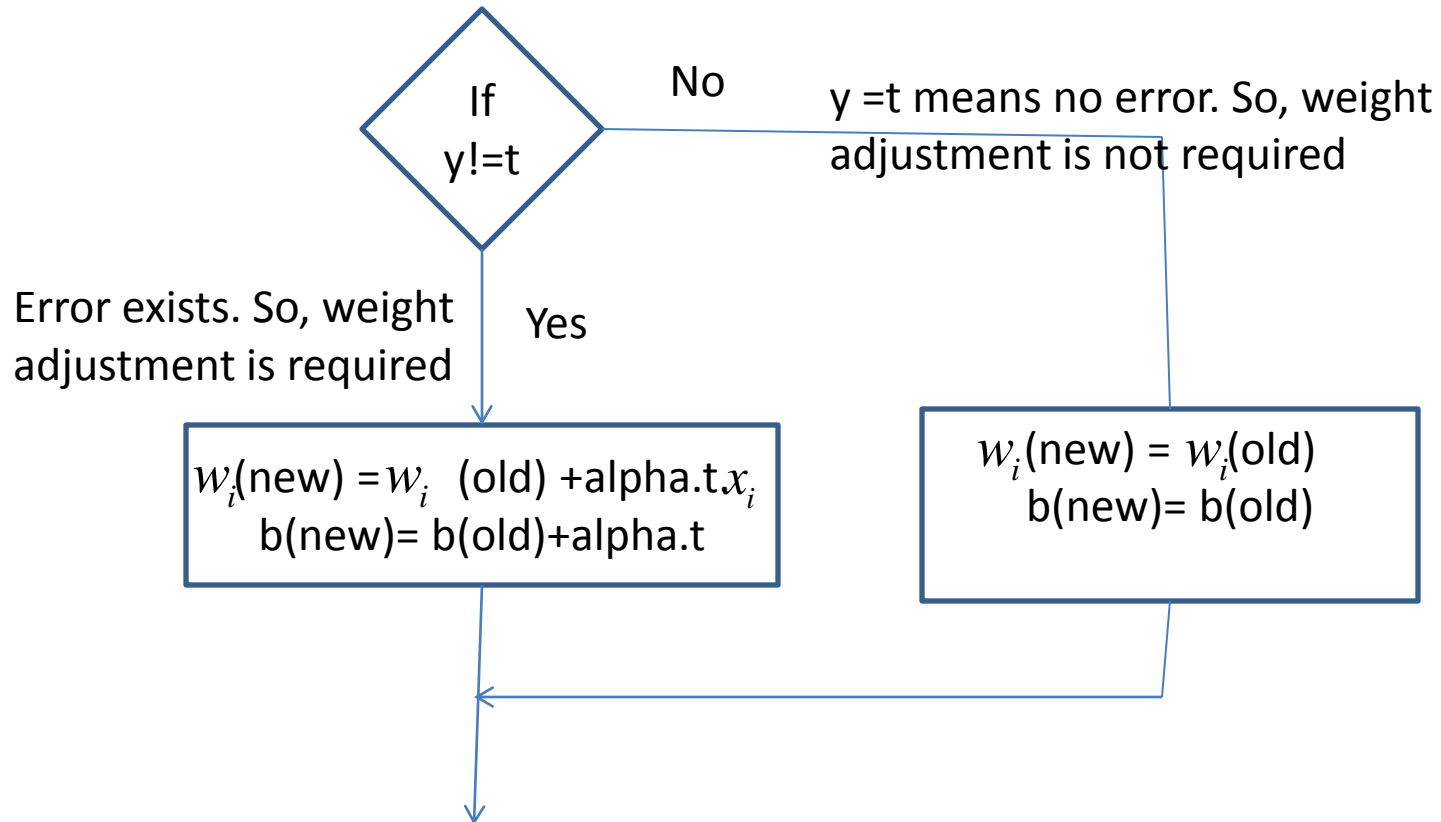
FLOW CHART FOR TRAINING PROCESS

- flow



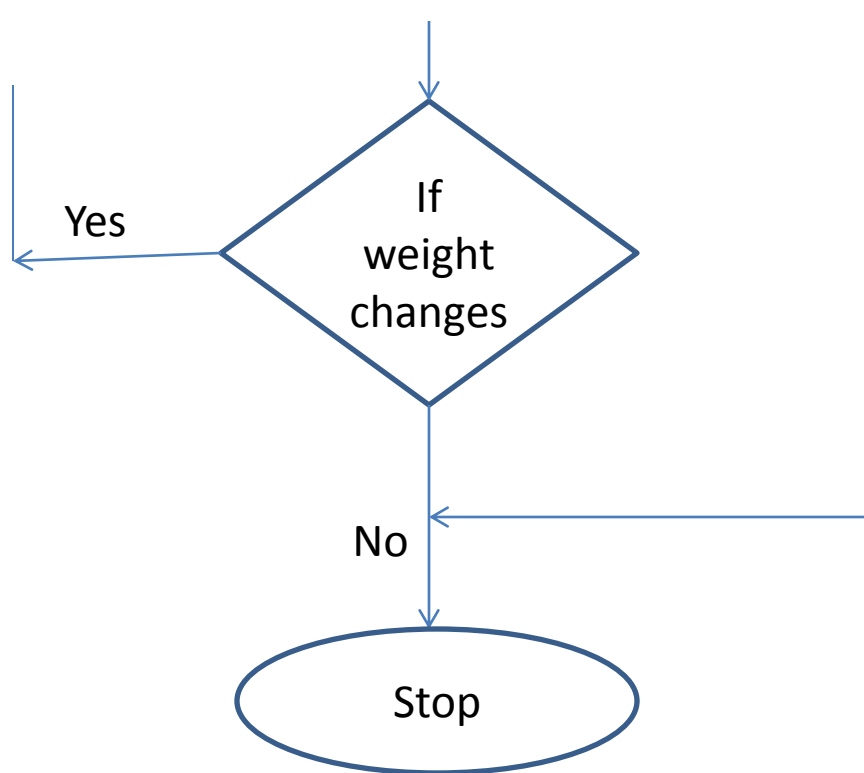
FLOW CHART FOR TRAINING PROCESS

- Flow



FLOW CHART FOR TRAINING PROCESS

- flow



PERCEPTRON TRAINING ALGORITHM FOR SINGLE OUTPUT CLASSES

- **STEP 0:** (i) Initialize the weights and the bias (for easy calculation they can be set to 0)
- (ii) Initialize the learning rate α (For simplicity it is set to 1)
- **STEP 1:** Perform steps 2 – 6 until the final stopping condition is false
- **STEP 2:** Perform steps 3 – 5 for each training pair indicated by $s:t$
- **STEP 3:** The input layer containing input units is applied with identity activation function $x_i = s_i$

PERCEPTRON TRAINING ALGORITHM FOR SINGLE OUTPUT CLASSES

- **STEP 4:** Calculate the output of the network.
- First obtain the net input using $y_{in} = b + \sum_{i=1}^n x_i \cdot w_i$
- Then apply activations over the net input to obtain
- the outputs:

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

PERCEPTRON TRAINING ALGORITHM FOR SINGLE OUTPUT CLASSES

- **STEP 5:** Weights and bias adjustment:
 - Compare the values of the actual output 'y' and the
 - desired output 't'.
 - If $y \neq t$ then
$$w_i(new) = w_i(old) + \alpha \cdot t \cdot x_i$$
$$b(new) = b(old) + \alpha \cdot t$$
 - Else
$$w_i(new) = w_i(old)$$
 - $$b(new) = b(old)$$
 -

PERCEPTRON TRAINING ALGORITHM FOR SINGLE OUTPUT CLASSES

- **STEP 6:** Train the network until there is no change. This is the stopping condition for the network. If this condition is not satisfied then start from step 2.

PERCEPTRON TRAINING ALGORITHM FOR MULTIPLE OUTPUT CLASSES

- Here instead of a single output 'y' there are multiple ('m' number of) output units
- **STEPS 0 to 3** remain same
- **STEP 4:** For each of the output units y_j , $j = 1, \dots, m$, calculate the net input by using:

$$(y_{in})_j = b_j + \sum_{i=1}^n x_i \cdot w_{ij}$$

Then activations are applied to calculate the outputs:

$$y_j = f((y_{in})_j) = \begin{cases} 1, & \text{if } (y_{in})_j > \theta; \\ 0, & \text{if } -\theta \leq (y_{in})_j \leq \theta; \\ -1, & \text{if } (y_{in})_j < -\theta. \end{cases}$$

PERCEPTRON TRAINING ALGORITHM FOR MULTIPLE OUTPUT CLASSES

- **STEP 5:** Make weight and bias adjustments for $j = 1, \dots, m$ and $i = 1, \dots, n$ as:

If $t_j \neq y_j$ then

$$w_{ij}(new) = w_{ij}(old) + \alpha \cdot t_j \cdot x_i$$

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

Else

$$w_{ij}(new) = w_{ij}(old)$$

$$b_j(new) = b_j(old)$$

PERCEPTRON TRAINING ALGORITHM FOR MULTIPLE OUTPUT CLASSES

- **STEP 6:** Test for stopping condition. That is, if there is no
- change in weights then stop the training process, else
- start from step 2.

PERCEPTRON NETWORK TESTING ALGORITHM

- The network performance is tested, once the training is over. For efficient performance of the network it should be trained with more data
- Testing algorithm:
- **STEP 0:** The final weights obtained after the training stage are taken as the initial weights
- **STEP 1:** For each input vector to be classified, perform steps 2 to 3
- **STEP 2:** Set activations of the input unit

PERCEPTRON NETWORK TESTING ALGORITHM

- **STEP 3:** Calculate the net input to the output unit using:

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

- **STEP 4:** Apply the activation function over the net input calculated:

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

- The testing of the network is done by classifying known inputs

BACK PROPOGATION NETWORK

- **Back propagation learning algorithm** is one of the most important developments in neural networks
- This learning algorithm is applied to **multilayer feed-forward networks** consisting of processing elements with **continuous differentiable activation functions**
- The networks using the back propagation learning algorithm are also called **back propagation networks** (BPN)
- Given a set of input, output pairs this algorithm **provides a procedure for changing the weights in a BPN**
- The **basic concept used** for weight updation is the **gradient-descent method** as used in simple perceptron network

GRADIENT DESCENT METHOD

- It is a method **used to find a local minimum of functions**
- It starts with **an initial guess of the solution**
- Takes the gradient of the function at that point
- **The solution is stepped in the negative direction of the gradient**
- Repeat the process. Suppose, x_k and x_{k+1} are two successive values in the sequel
- Then we have $x_{k+1} = x_k - \theta \cdot f'(x_k)$, $\theta > 0$ is a small number that forces the algorithm to make small jumps
- For suitable choices of θ it is guaranteed that $f(x_{k+1}) \leq f(x_k)$

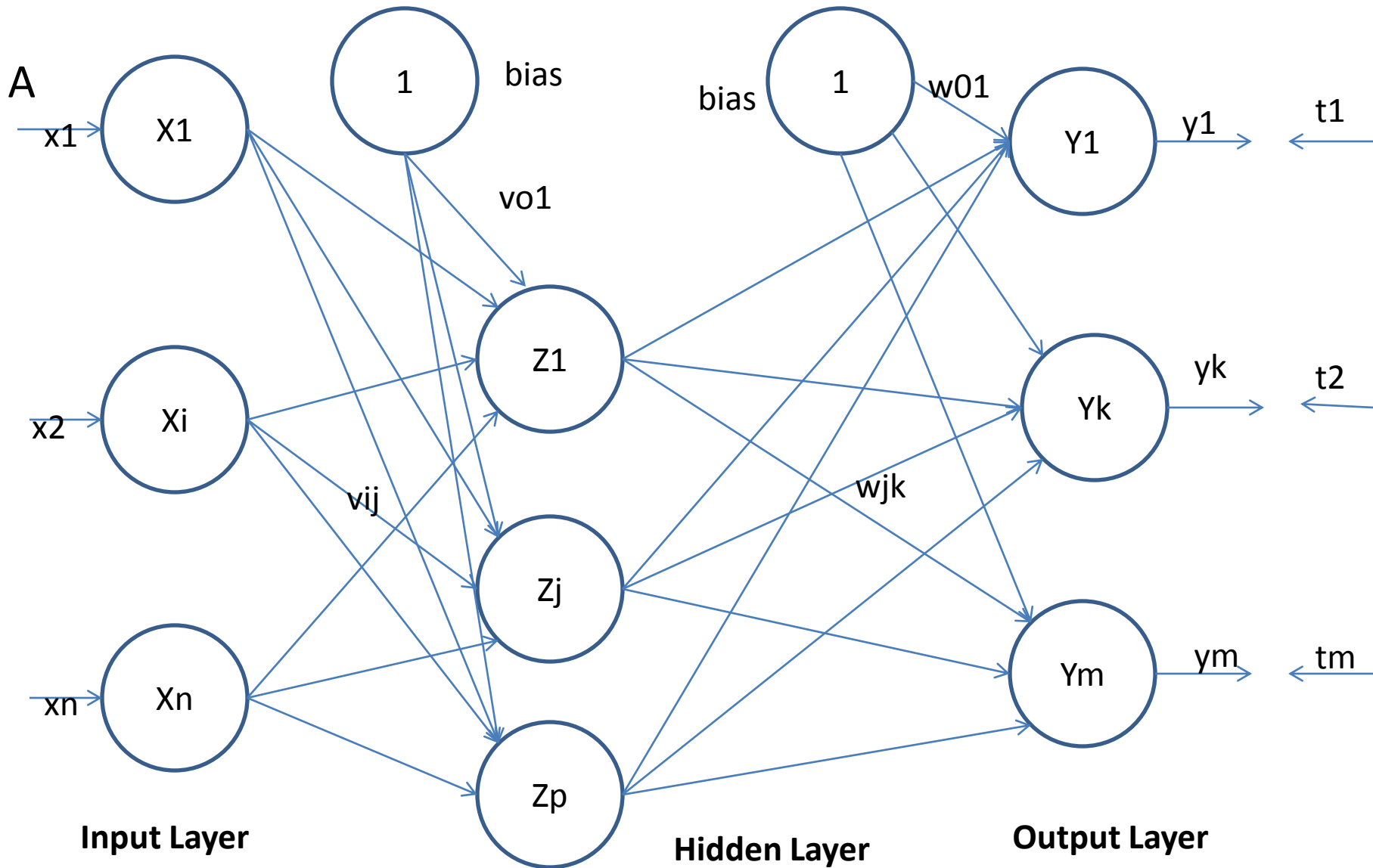
BACK PROPOGATION NETWORK

- In this method the **error is propagated back to the hidden unit**
- **It differs from other networks in respect to the process by which the weights are calculated during the learning period**
- When the number of hidden layers is increased the complexity increases
- The error is **usually measured at the output layer**
- At the **hidden layers there is no information about the errors**
- So, **other techniques are needed** to be followed to **calculate the error at the hidden layers** so that the ultimate error is minimised

ARCHITECTURE OF BACK PROPOGATION NETWORK

- It is a **multi layer**, **feed forward** neural network
- It has **one input layer**, **one hidden layer** and **one output layer**
- The neurons in the hidden and output layer have biases
- During the **back-propagation phase of learning**, **signals are sent in the reverse direction**
- The inputs are sent to the BPN and the **output** obtained from the net could be
 - **Binary** {0, 1} or
 - **Bipolar** {-1, 1}
- The **activation function** could be any function which **increases monotonically** and is also **differentiable**

DIAGRAMATIC REPRESENTATION OF THE ARCHITECTURE OF BPN



FLOWCHART DESCRIPTION FOR TRAINING PROCESS

- x = input training vector (x_1, x_2, \dots, x_n)
- t = target output vector (t_1, t_2, \dots, t_m)
- α = learning rate parameter
- x_i = input unit i
- v_{0j} = bias on j th hidden unit
- w_{ok} = bias on k th output unit
- z_j = j th hidden unit
- Then we have

$$(z_j)_{in} = v_{0j} + \sum_{i=1}^n x_i \cdot v_{ij}, j = 1, 2, \dots, p$$

FLOWCHART DESCRIPTION FOR TRAINING PROCESS CONTD...

- The output from the j th hidden unit is

$$z_j = f((z_j)_{in})$$

- Let y_k be the k th output unit. The net input to it is

$$(y_{in})_k = w_{ok} + \sum_{j=1}^p z_j \cdot w_{jk}, k = 1, 2, \dots, m$$


- The output from the k th output unit is


-

$$y_k = f((y_{in})_k)$$

ACTIVATION FUNCTIONS USED

- The commonly used **activation functions** are **binary sigmoidal** and **bipolar sigmoidal functions** have the properties:
- Continuous
- Differentiable
- Monotonously Non-decreasing
- So, **these functions are used as the activation functions** here
- δ_k =Error correction weight adjustment for w_{jk}
- This error is at the output unit y_k
- **This is back-propagated to the hidden units** which have fed into y_k


$$f(x) = \frac{1 - e^{-\lambda x}}{1 + e^{-\lambda x}},$$


$$f(x) = \frac{1}{1 + e^{-\lambda x}},$$

ERROR CORRECTION CONTD...

- δ_j is the **error correction weight adjustment** for v_{ij}
- This **error correction has occurred** due to the **back propagation of error to the hidden unit** z_j

TRAINING ALGORITHM

- **STEP 0:** Initialize **weights** and **learning rate** (some **random small values** are taken)
- **STEP 1:** Perform Steps 2 -9 when stopping condition is false
- **STEP 2:** Perform steps 3 – 8 for **each training pair**
- **STEP 3:** Each **input unit receives** input signal x_i and **sends it to the hidden unit** , $i = 1, \dots, n$
- **STEP 4:** Each **hidden unit** z_j , $j = 1, \dots, p$ **sums its weighted input signals to calculate net input:**

$$(z_j)_{in} = z_{oj} + \sum_{i=1}^n x_i \cdot v_{ij}$$

TRAINING ALGORITHM

- Calculate the outputs from the hidden layer by applying activation functions over $(z_{in})_j$, i.e. $z_j = f((z_{in})_j)$
- **These signals are sent as input signals for the output units**
- For **each output unit** $y_k, k = 1, \dots, m$, **calculate the net input**

$$(y_{in})_k = w_{0k} + \sum_{j=1}^p z_j \cdot w_{jk}$$

- **Apply the activation function** to compute the output signal

$$y_k = f((y_{in})_k), k = 1, 2, \dots, m.$$

TRAINING ALGORITHM (BACK PROPAGATION OF ERROR)

- STEP 6: **Each output unit** $y_k, k = 1, 2 \dots m$ **receives a target pattern to the input training pattern** and **computes the error** correction using:

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

- On the basis of the calculated error correction term, update the change in weights and bias:

$$\Delta w_{jk} = \alpha \cdot \delta_k \cdot z_j$$

$$\Delta w_{0k} = \alpha \cdot \delta_k$$

- **Also, send δ_k to the hidden layer backwards**

TRAINING ALGORITHM (BACK PROPAGATION OF ERROR)

- STEP 7: Each hidden unit $z_j, j = 1, 2 \dots p$ sums its delta inputs from the output units:

$$(\delta_{in})_j = \sum_{k=1}^m \delta_k \cdot w_{jk}$$

- The term $(\delta_{in})_j$ gets multiplied with the derivative of $f((z_{in})_j)$ to calculate the error term:

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

- On the basis of the calculated δ_j , update the **change in the weights and the bias:**

- $$\Delta v_{ij} = \alpha \cdot \delta_j \cdot x_i$$

$$\Delta v_{oj} = \alpha \cdot \delta_j$$

TRAINING ALGORITHM (WEIGHTS AND BIAS UPDATE PHASE)

- **Each output unit**, $y_k, k = 1, 2, \dots, m$ **updates the bias and weights as:**

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

$$w_{0k}(\text{new}) = w_{0k}(\text{old}) + \Delta w_{0k}$$

- **Each hidden unit**, $z_j, j = 1, \dots, p$ **updates its bias and weights as:**

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

- $$v_{0j}(\text{new}) = v_{0j}(\text{old}) + \Delta v_{0j}$$
-

- STEP 9: Check for the **stopping condition**. The stopping condition **may be a certain number of cycles** reached or when **actual output is equal to the target output** (That is error is zero)

LEARNING FACTORS OF BACK PROPAGATION ALGORITHM

- **Convergence** of the BPN is based upon several important factors
- **Initial Weights**
- **Learning rate**
- **Up gradation rule**
- **Size and nature of the training set**
- **Architecture** (Number of layers and number of neurons per layer)

INITIAL WEIGHTS

- **Ultimate solution may be affected by the initial weights**
- These are **initialized by small random values**
- The choice of initial weights **determines the speed at which the network converges**
- **Higher initial weights** may lead to **saturation of activation functions** from the beginning by stocking up **at a local minimum**
- One method to select the weights $w_{ij} \in \left[\frac{-3}{\sqrt{o_i}}, \frac{3}{\sqrt{o_i}} \right]$
- O_i = The number of processing elements j that feed forward to the ith processing element

INITIAL WEIGHTS CONTD...

- |

$$v_{ij}(new) = \gamma \cdot \frac{v_{ij}(old)}{\overline{v_j}(old)}$$

- $\overline{v_j}$ is the average weight calculated for all i, that is

$$\overline{v_j} = \frac{\sum_{i=1}^n v_{ij}}{n}$$

- γ is the scale factor . $\gamma = 0.7(p)^{1/n}$
- Here, p is the number of input neurons and p is the number of hidden layer neurons

LEARNING RATE

- Learning rate α also affects the convergence of the network
- Larger value of α May speed up the convergence but may lead to overshooting
- The range of α is 10^{-3} to 10
- Large learning rate leads to rapid learning but there will be oscillation of weights
- Lower learning rate leads to slow learning\

MOMENTUM FACTOR

- To overcome the problems stated above (in the previous slide)
- We add a factor called momentum factor to the usual gradient descent method
- Momentum factor η is in the interval $[0, 1]$
- η is normally taken to be 0.9

$$w_{jk}(t+1) = w_{jk}(t) + \alpha \delta_k z_j + \eta[w_{jk}(t) - w_{jk}(t-1)]$$

$$v_{ij}(t+1) = w_{ij}(t) + \alpha \delta_j x_i + \eta[v_{ij}(t) - v_{ij}(t-1)]$$

NUMBER OF TRAINING DATA

- The training data should be sufficient and proper
- Training vectors should be taken randomly from the training set
- The data set should be representing the input space
- Suppose the input space is linearly separable with L disjoint regions
- Let T be the lower bound on the number of training patterns
- Then T should be selected such that $T/L \gg 1$

NUMBER OF HIDDEN LAYER NODES

- If the number of hidden layers is more than 1 in a BPN then
- The calculations performed for a single layer are repeated
- The size of a layer is very important
- It is determined experimentally
- If the network does not converge then the number of hidden nodes are to be increased
- If the network converges then the user may try with a few hidden nodes and settle for a size based on the performance
- In general the size of hidden nodes should be a relatively small fraction of the input layer

TESTING ALGORITHM FOR BPN

- STEP 0: Initialize the weights. (The weights are taken from the training phase)
- STEP 1: Perform steps from 2 – 4 for each input vector
- STEP 2: Set the activation of input for x_i , $i = 1, 2, \dots, n$
- STEP 3: Calculate the net input to hidden unit z and its output

- $$(z_{in})_j = v_{oj} + \sum_{i=1}^n x_i \cdot v_{ij} \quad \text{and} \quad z_j = f((z_{in})_j)$$

- STEP 4: Compute the net input and the output of the output layer

$$(y_{in})_k = w_{0k} + \sum_{j=1}^p z_j \cdot w_{jk}, \quad y_k = f((y_{in})_k)$$

- **USE SIGMOIDAL FUNCTIONS AS ACTIVATION FUNCTIONS**