Artificial Intelligence & Soft Computing

CSC 703

Subject In-charge

Safa Hamdare

Assistant Professor

Room No. 407

email: safahamdare@sfit.ac.in

Chapter 5

Artificial Neural Network

Based on CO4: Construct supervised and unsupervised ANN for real world applications.

Outline:

- **Introduction**
  - Fundamental concept
  - Basic Models of Artificial Neural Networks
  - Important Terminologies of ANNs
  - McCulloch-Pitts Neuron
- **Neural Network Architecture**
  - Perceptron
  - Single layer Feed Forward ANN,
  - Multilayer Feed Forward ANN,
  - Activation functions
- **Supervised Learning**
  - Delta learning rule, Back Propagation algorithm.
- **Un-Supervised Learning algorithm**
  - Self Organizing Maps

Introduction

Introduction- Biological Neuron

Biological Neurons

Ramón-y-Cajal (Spanish Scientist, 1852~1934):

1. Brain is composed of individual cells called ***neurons***.
2. Neurons are connected to each others by ***synopses***.

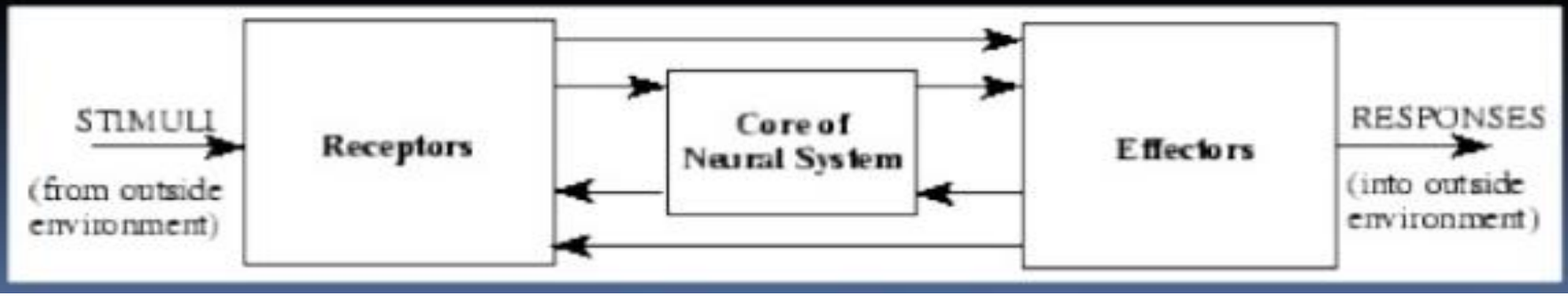
Introduction- ANN

- Hopes to reproduce human brain by artificial means.
- It mimics how our nervous system process information.
- **ANN** is composed of a large number of highly interconnected processing elements (neurons) working in union to solve specific problems.
- **ANNs,** like people, learn by example.
- Learning in biological systems involves adjustments to the synaptic connections that exist between the neurons. This is true of ANNs as well.

Introduction- Biological Neuron

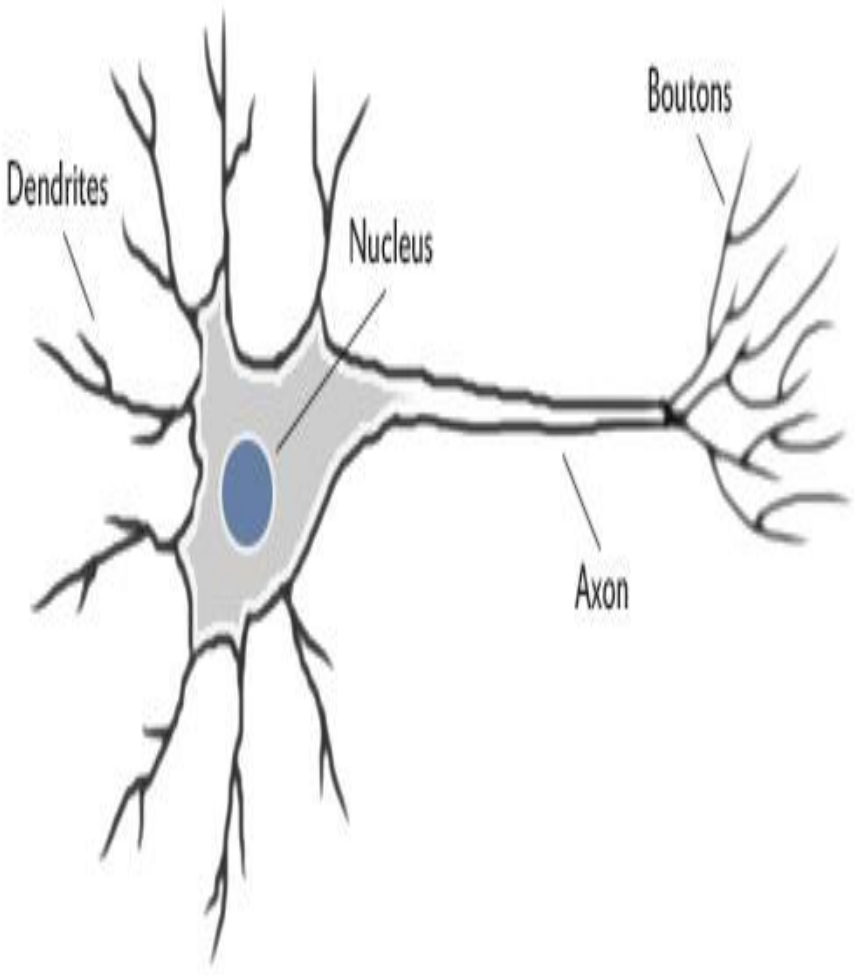
Stages of Biological Neural System

- The neural system of the human body consists of three stages: receptors, a neural network, and effectors. The receptors receive the stimuli either internally or from the external world, then pass the information into the neurons in a form of electrical impulses. The neural network then processes the inputs then makes proper decision of outputs. Finally, the effectors translate electrical impulses from the neural network into responses to the outside environment. Figure shows the bidirectional communication between stages for feedback



Introduction- Biological Neuron working

- Neuron collects inputs using dendrites
- Then it sums up all inputs from dendrites
- Nucleus process the inputs
- If the resulting value is greater than its firing threshold, the neuron fires.
- Firing neuron sends an electrical impulse through the neuron's axon to its boutons.
- Boutons connect to other neurons via synapses.



Introduction- Biological Neuron working

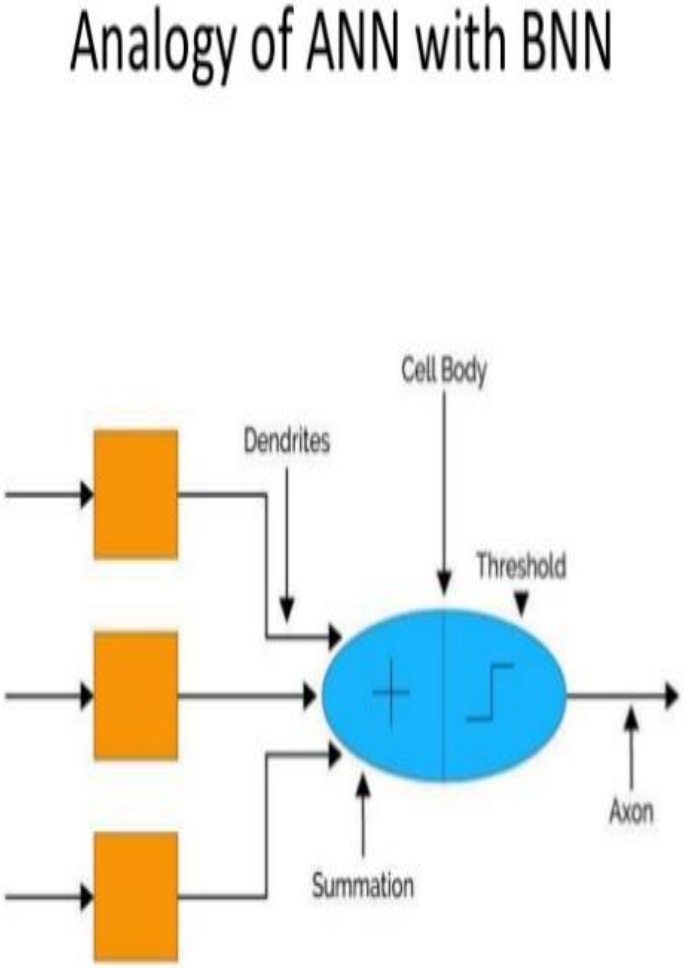
Neurons Function (Biology)

1. Dendrites receive signal from other neurons.
2. Neurons can process (or transfer) the received signals.
3. Axon terminals deliverer the processed signal to other tissues.

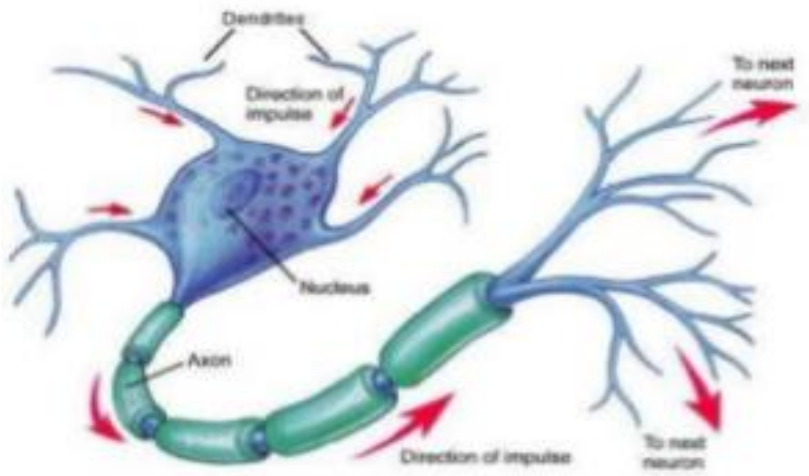
What kind of signals? Electrical Impulse Signals

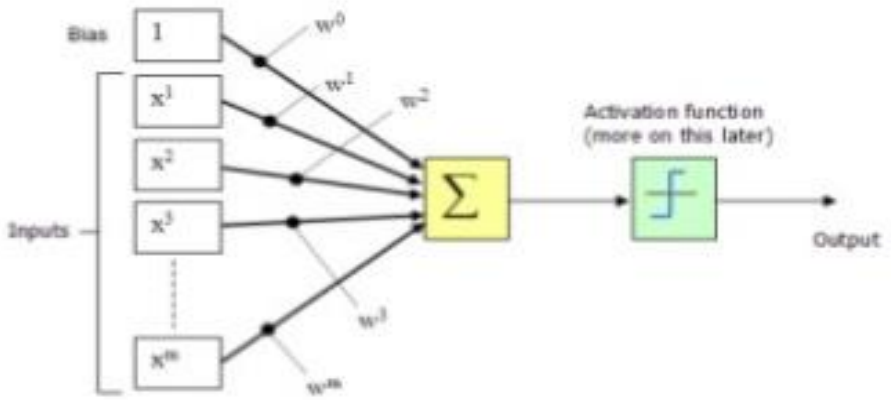
Introduction- Human neurons to Artificial Neurons

- The dendrites in biological neural network is analogous to the weighted inputs based on their synaptic interconnection in artificial neural network.
- Cell body is analogous to the artificial neuron unit in artificial neural network which also comprises of summation and threshold unit.
- Axon carry output that is analogous to the output unit in case of artificial neural network. So, ANN are modelled using the working of basic biological neurons.



Introduction- Biological Vs Artificial Neuron





Human neurons	Artificial neurons
Neurons	Neurons
Axon, Synapse	Wkj (weight)
Synaptic terminals to next neuron	output terminals
Synaptic terminals taking input	input terminals (Xj)
human response time=1 ms	silicon chip response time=1ns

Introduction- Biological Vs Artificial Neuron

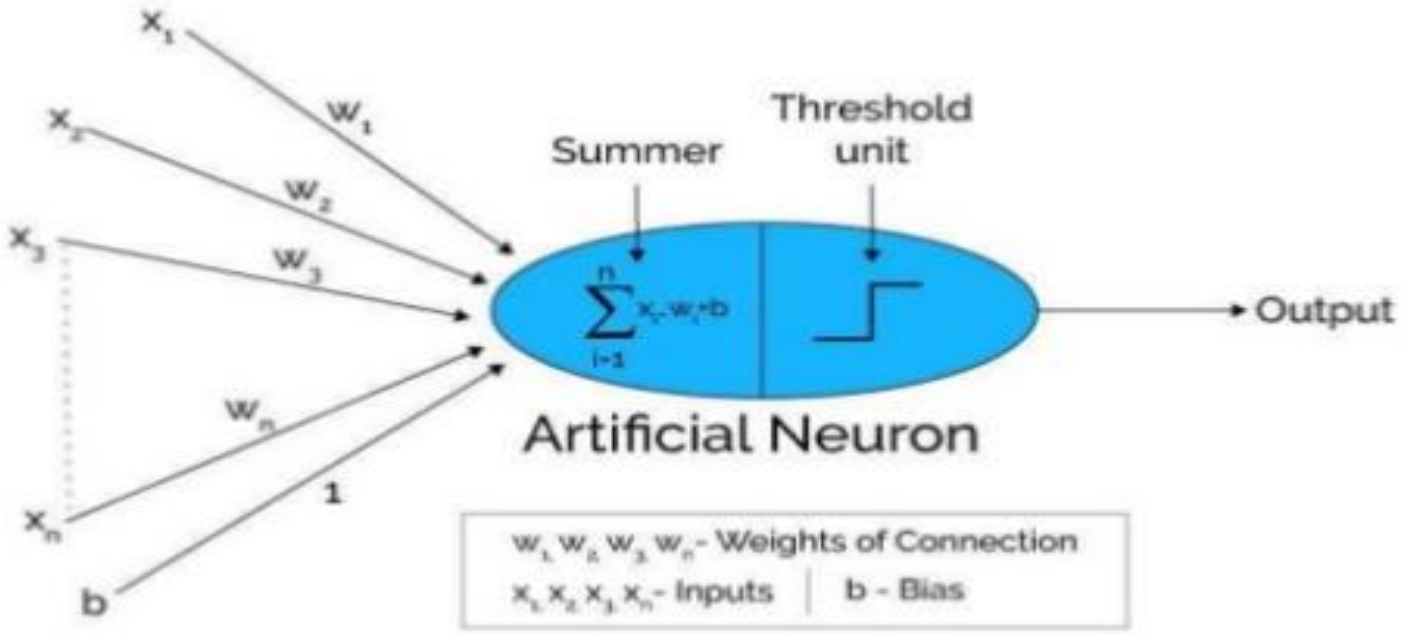
Biological Neuron	Artificial Neuron
1. Cycle time is in milliseconds	1. Cycle time is in nanoseconds
2. Massively parallel	2. Several parallel
3. 1011 neurons and 1015 interconnections	3. No. of neurons is application specific
4. New information can be added without destroying old	4. New information overload memory locations
5. Fault tolerant	5. Not fault tolerant
6. Control mechanism is complicated	6. Control mechanism is simple

Introduction- Why Neural Network?

1. It can be used to extract patterns and detect trends
2. **Adaptive learning:** learn based on training or initial experience.
3. **Self-Organization:** organize and represent information on its own.
4. **Real Time Operation:** applies parallel computations
5. **Fault Tolerance**: Partial destruction leads to graceful degradation

Introduction- Model of ANN

Model of Artificial Neural Network



Introduction- Model of ANN

- Artificial neural networks can be viewed as weighted directed graphs in which artificial neurons are nodes and directed edges with weights are connections between neuron outputs and neuron inputs.
- The Artificial Neural Network receives input from the external world in the form of pattern and image in vector form. These inputs are mathematically designated by the notation x(n) for n number of inputs.
- Each input is multiplied by its corresponding weights. Weights are the information used by the neural network to solve a problem. Typically weight represents the strength of the interconnection between neurons inside the neural network.
- The weighted inputs are all summed up inside computing unit (artificial neuron). In case the weighted sum is zero, bias is added to make the output not- zero or to scale up the system response. Bias has the weight and input always equal to '1'.

Introduction- Model of ANN

- The sum corresponds to any numerical value ranging from 0 to infinity.
- In order to limit the response to arrive at desired value, the threshold value is set up. For this, the sum is passed through activation function.
- The activation function is set of the transfer function used to get desired output. There are linear as well as the non-linear activation function.

Introduction- Modelling Neurons

Net input signal received through synaptic junctions is

$$net = b + \sum w_i x_i = b + W^T X$$

Weight vector: $W = [w_1\ w_2 \ldots w_m]^T$

Input vector: $X = [x_1\ x_2 \ldots x_m]^T$

Each output is a function of the net stimulus signal (f is called the activation function)

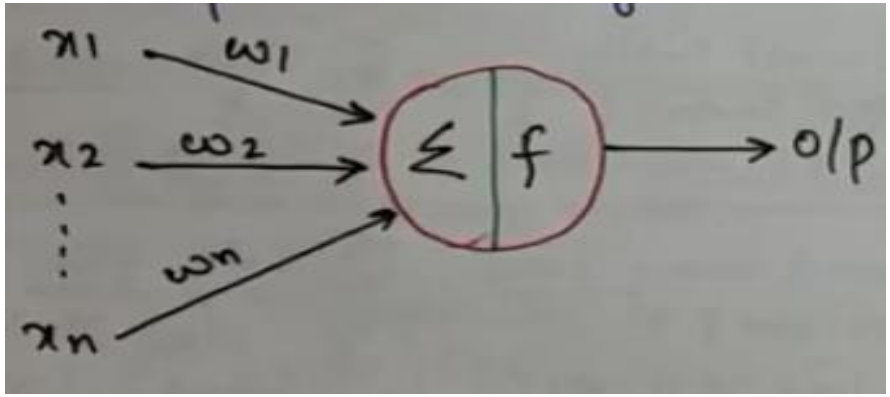
$$y = f(net) = f(b + W^T X)$$

Introduction- Important Terminologies of ANN

- **Artificial Neurons-** Nodes in Network
- **Directed Edges with weights-** Connection between inputs and outputs
- **Weights-** information used by neural network to solve a problem. Typically it represents strength of interconnection between neurons inside neural network.
- **Bias-** added to make the output non zero or to scale up the system. It act as a constant.
- **Sum-** Corresponds to any numerical value from 0 to infinity.
- **Threshold-** In order to limit the response to arrive at desired value threshold is set up. So sum is passed through activation function
- **Activation Function-** It is set of transfer function used to get desired output. There are linear and non linear activation functions.

Introduction- McCulloch-Pitts Neuron Model

- The first mathematical model of biological neuron was given by Warren McCulloch and Walter Pitts (1943).
- Also known as Linear Threshold gates model. *(The linear threshold gate simply classifies the set of inputs into two different classes.)*
- Basic building block of neural Network.
- It is a directed weighted graph used for connecting neurons.
- **Two possible states of neuron are there**
  - Active state(when output is 1)
  - Silent state (when output is 0)



Introduction- McCulloch-Pitts Neuron Model

Properties of Mc Culloch Pitts Neuron Model:

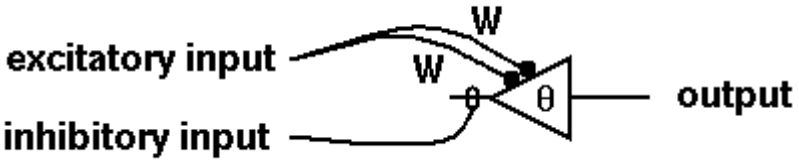
1. They are binary devices (Vi = [0,1])
2. Each neuron has a fixed threshold, theta
3. The neuron receives inputs from **excitatory synapses**, all having identical weights. (However it my receive multiple inputs from the same source, so the excitatory weights are effectively positive integers.)
4. **Inhibitory inputs** have an absolute veto power over any excitatory inputs.
5. At each time step the neurons are simultaneously (synchronously) updated by summing the weighted excitatory inputs and setting the output (Vi) to 1 iff the sum is greater than or equal to the threshold AND if the neuron receives no inhibitory input.

Introduction- McCulloch-Pitts Neuron Model

- We can summarize these rules with the McCullough-Pitts output rule

$$V_i = \begin{cases} 1 & : \sum_j W V_j \geq \theta \text{ AND no inhibition} \\ 0 & : \text{otherwise} \end{cases}$$

- The diagram:

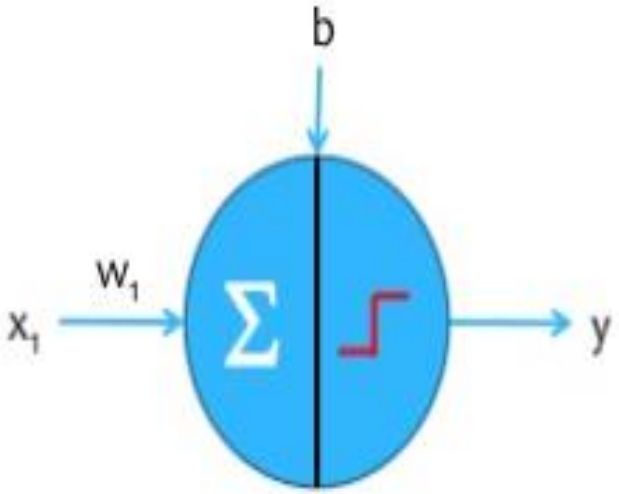


- Although this neuron model is simple it has substantial computing potential.
- Every basic Boolean function can be implemented using combination of McCulloch-Pitts Neuron
- It can perform basic logic operations NOT, OR, and AND provided its weights and thresholds are appropriately selected.

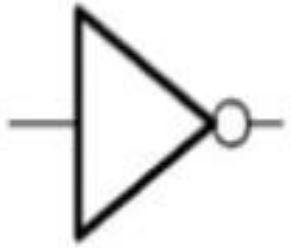
Introduction- McCulloch-Pitts Neuron Model- Not gate

Single-input McCulloch-Pitts neurode with b=0, w_1=-1 for binary inputs:

x_1	net	y
0	0	1
1	-1	0



Conclusion?

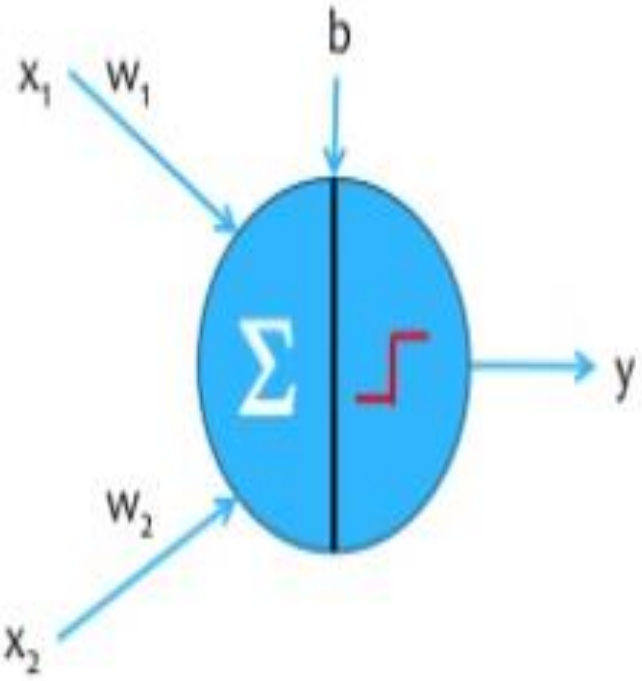


Introduction- McCulloch-Pitts Neuron Model- OR gate

Two-input McCulloch-Pitts neurode with b=-1,

$w_1=w_2=1$ for binary inputs:

x_1	x_2	net	y
0	0	?	?
0	1	?	?
1	0	?	?
1	1	?	?

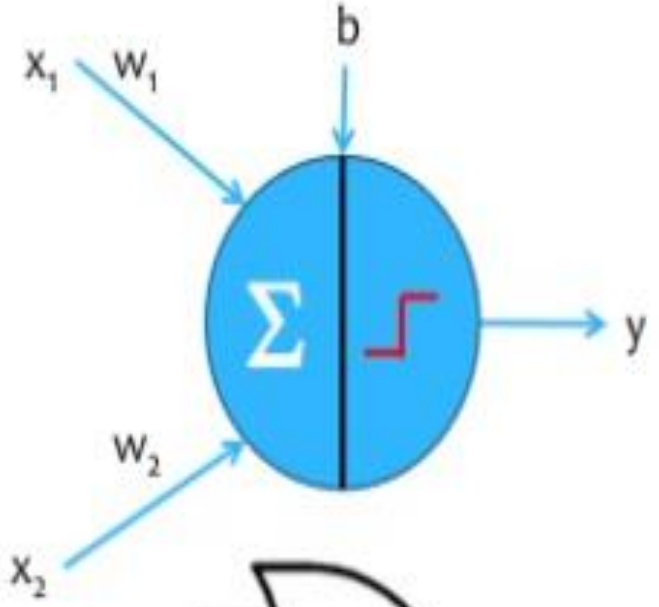


Introduction- McCulloch-Pitts Neuron Model- OR gate

Two-input McCulloch-Pitts neurode with b=-1,

$w_1=w_2=1$ for binary inputs:

x_1	x_2	net	y
0	0	-1	0
0	1	0	1
1	0	0	1
1	1	1	1

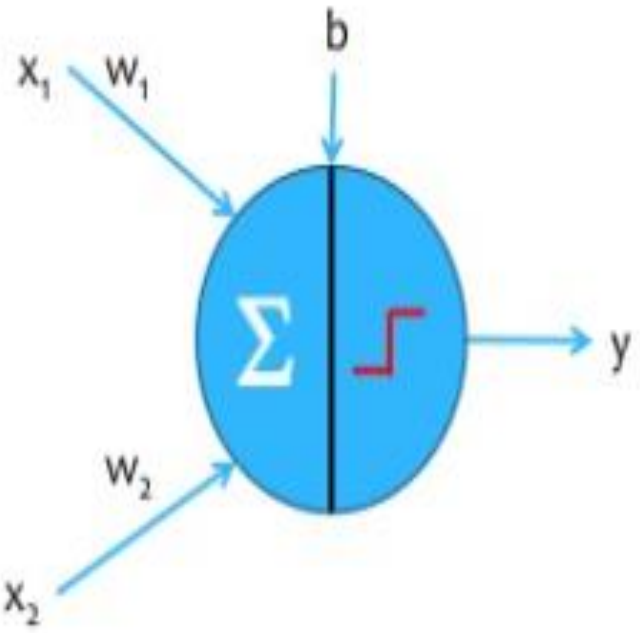


Introduction- McCulloch-Pitts Neuron Model- AND gate

Two-input McCulloch-Pitts neurode with b=-2,

$w_1=w_2=1$ for binary inputs :

x_1	x_2	net	y
0	0	?	?
0	1	?	?
1	0	?	?
1	1	?	?

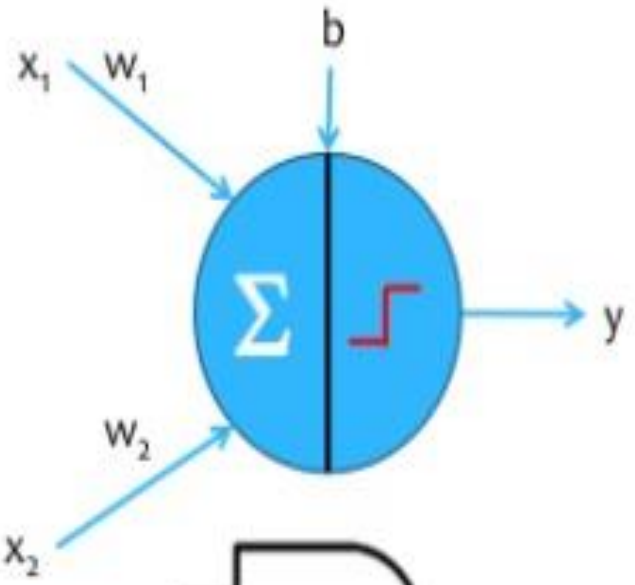


Introduction- McCulloch-Pitts Neuron Model- AND gate

Two-input McCulloch-Pitts neurode with b=-2,

$w_1=w_2=1$ for binary inputs :

x_1	x_2	net	y
0	0	-2	0
0	1	-1	0
1	0	-1	0
1	1	0	1



Introduction- McCulloch-Pitts Neuron Model- Limitation

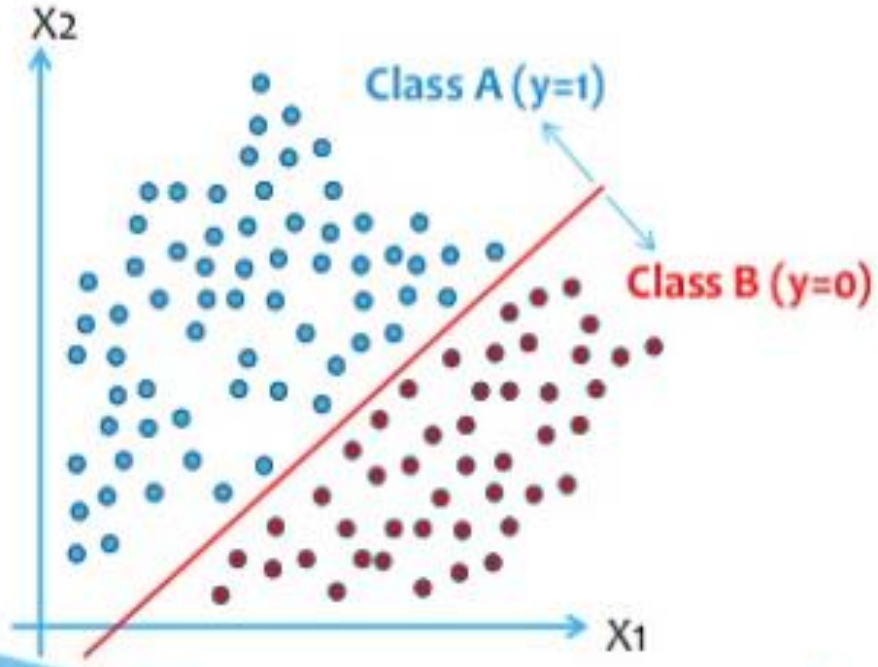
- It allows binary 0, 1 states only,
- It operates under a discrete-time assumption
- Weights and thresholds are fixed in the model
- ANN employ a variety of neuron models that have more diversified features than this model

Introduction- McCulloch-Pitts Neuron Model

the McCulloch-Pitts neuron can be used as a classifier that separate the input signals into two classes (perceptron):

Class A ⇔ y=?
y = 1 ⇔ net = ?
net ≥ 0 ⇔ ?
$b+w_1x_1+w_2x_2 \geq 0$

Class B ⇔ y=?
y = 0 ⇔ net = ?
net < 0 ⇔ ?
$b+w_1x_1+w_2x_2 < 0$



Introduction- McCulloch-Pitts Neuron Model

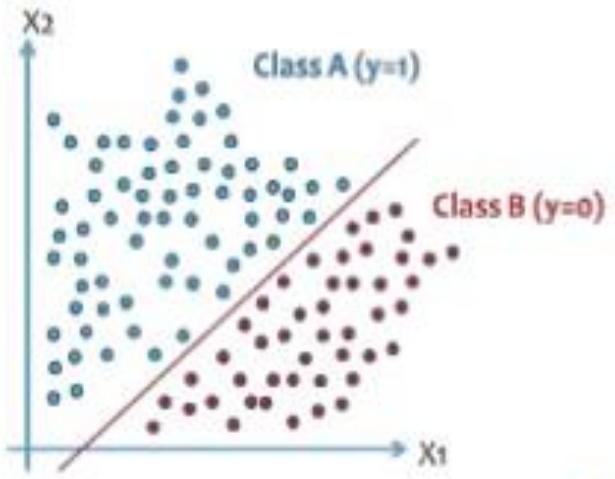
Class A $\Leftrightarrow$ $b+w_1x_1+w_2x_2 \geq 0$

Class B $\Leftrightarrow$ $b+w_1x_1+w_2x_2 < 0$

Therefore, the decision boundary is a hyperline given by

$$b+w_1x_1+w_2x_2 = 0$$

Where w_1 and w_2 come from?



Introduction- McCulloch-Pitts Neuron Model

- Design a Mc-Culloh Pitts model for XOR Gate? **Dec 2019 10 marks**

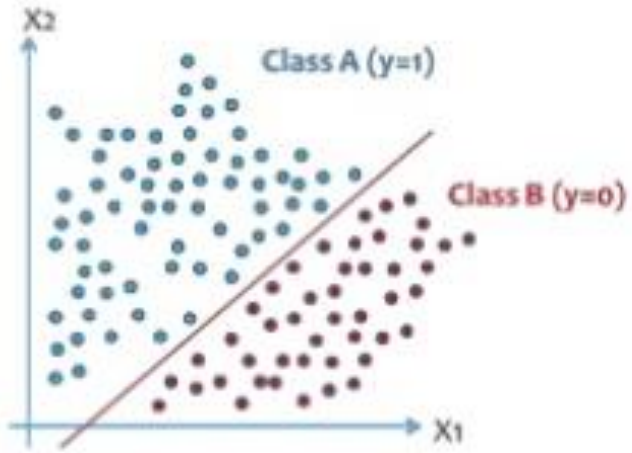
Neural Network Architecture

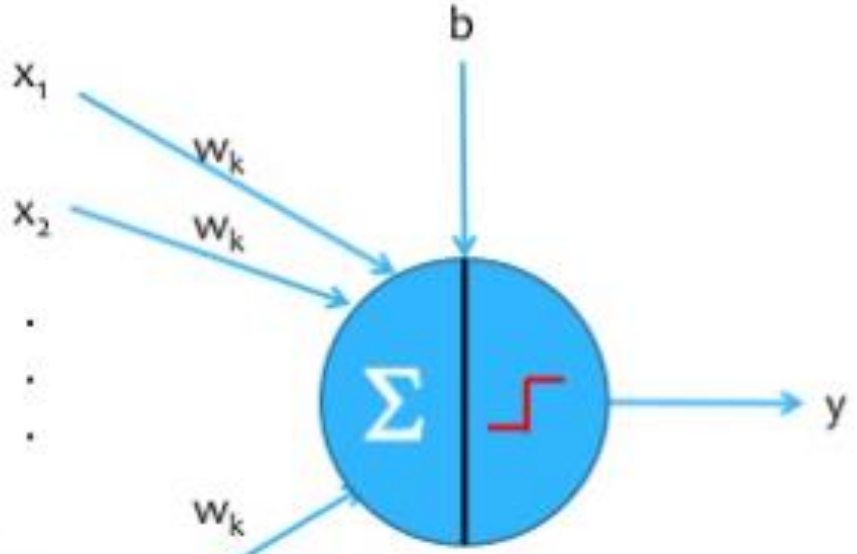
Neural Network Architecture- Perceptron

- The next major advance was the **perceptron**, introduced by Frank Rosenblatt in his 1958 paper.
- **The perceptron had the following differences from the McCullough-Pitts neuron:**
  - The weights and thresholds were not all identical.
  - Weights can be positive or negative.
  - There is no absolute inhibitory synapse.
  - Although the neurons were still two-state, the output function f(u) goes from [-1,1], not [0,1]. (This is no big deal, as a suitable change in the threshold lets you transform from one convention to the other.)
  - Most importantly, there was a learning rule.

Neural Network Architecture- Perceptron

- The goal of the perceptron to classify input data into two classes A and B
- Only when the two classes can be separated by a linear boundary
- The perceptron is built around the McCulloch-Pitts Neuron model
- A linear combiner followed by a hard limiter
- Accordingly the neuron can produce +1 and 0





Neural Network Architecture- Perceptron

There exist a weight vector **w** such that we may state

$\mathbf{W^T x} > 0$ for every input vector **x** belonging to **A**

$\mathbf{W^T x} \leq 0$ for every input vector **x** belonging to **B**

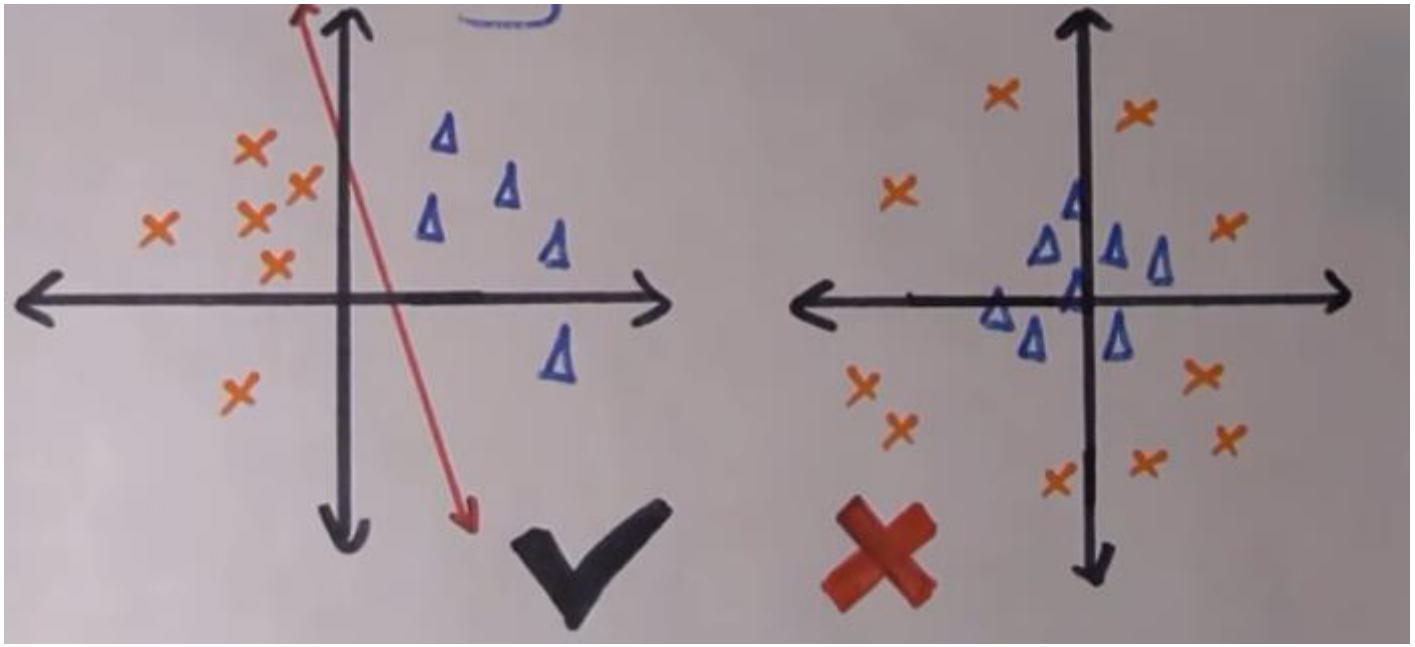
Neural Network Architecture- Perceptron

Elementary perceptron Learning Algorithm

1) Initiation: $w(0)$ = a random weight vector
2) At time index n, form the input vector $\mathbf{x}(n)$
3) IF ($\mathbf{w}^T\mathbf{x} > 0$ and x belongs to A) or ($\mathbf{w}^T\mathbf{x} \leq 0$ and $\mathbf{x}$ belongs to B) THEN $\mathbf{w}(n)=\mathbf{w}(n-1)$ Otherwise $\mathbf{w}(n)=\mathbf{w}(n-1)-\eta\mathbf{x}(n)$
4) Repeat 2 until $\mathbf{w}(n)$ converges

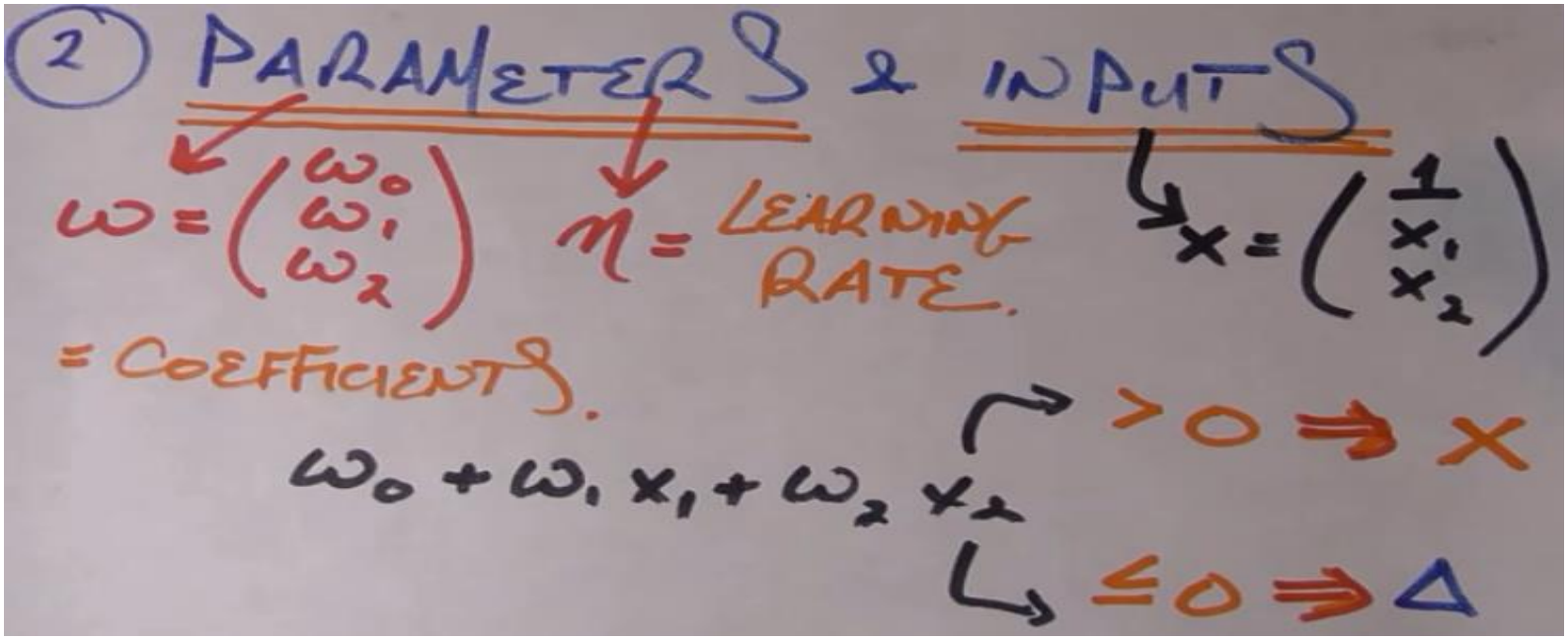
Neural Network Architecture- Perceptron

- Perceptron can be used when the dataset is linearly separable.
- When we can draw a line that can separate two classes.

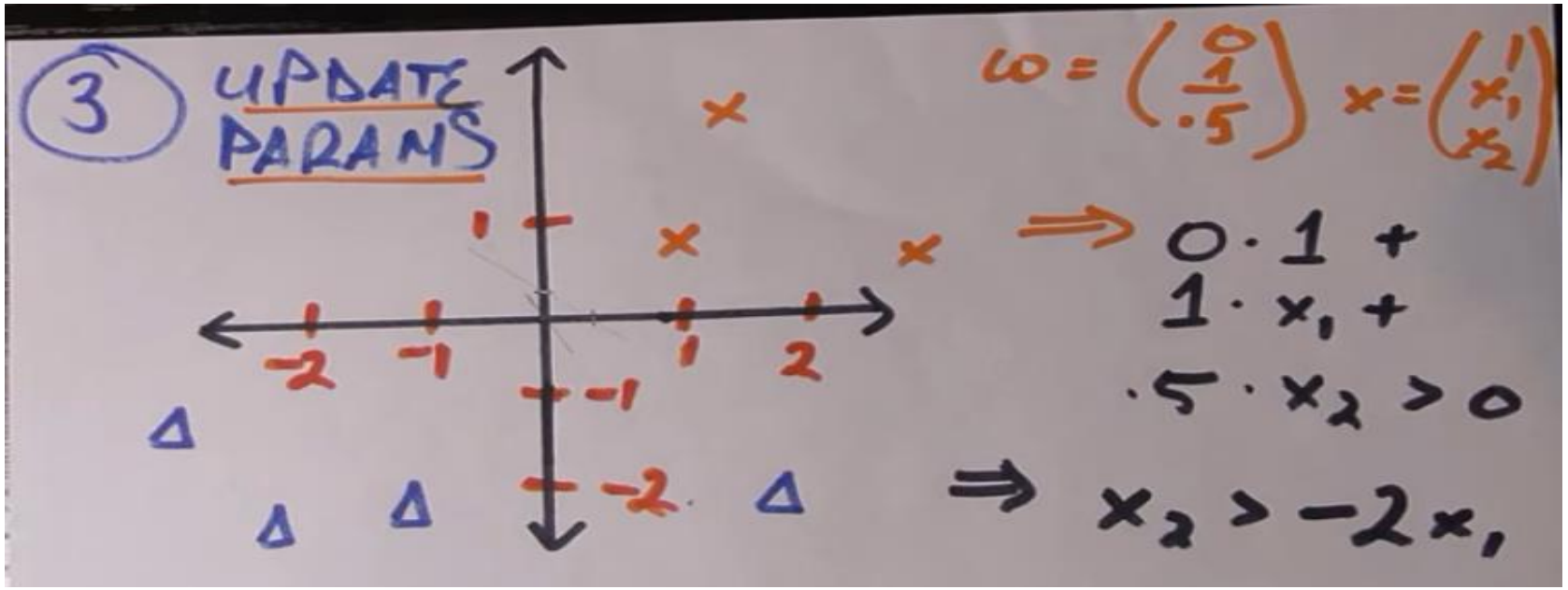


Neural Network Architecture- Perceptron

- **Parameters and inputs of perceptron:**
  - W it's a vector taken for no. of dimensions + 1 extra dimension for constant (these are the system Coefficients)
  - N- neu is the learning rate (how fast it will take to separate the class)
  - Input- value of x1, x2 in two dimension.



Neural Network Architecture- Perceptron



Woxo+w1x1+w2x2>0

0*1 + 1*x1+0.5*x2>0

X1+0.5X2>0

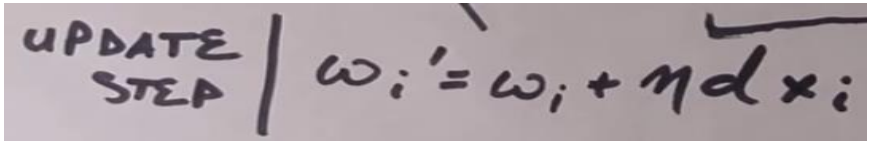
X2>-2X1

So this line of equation divides the plane in two classes (anything above that line is classified as x and anything below that line is classified as Triangle).

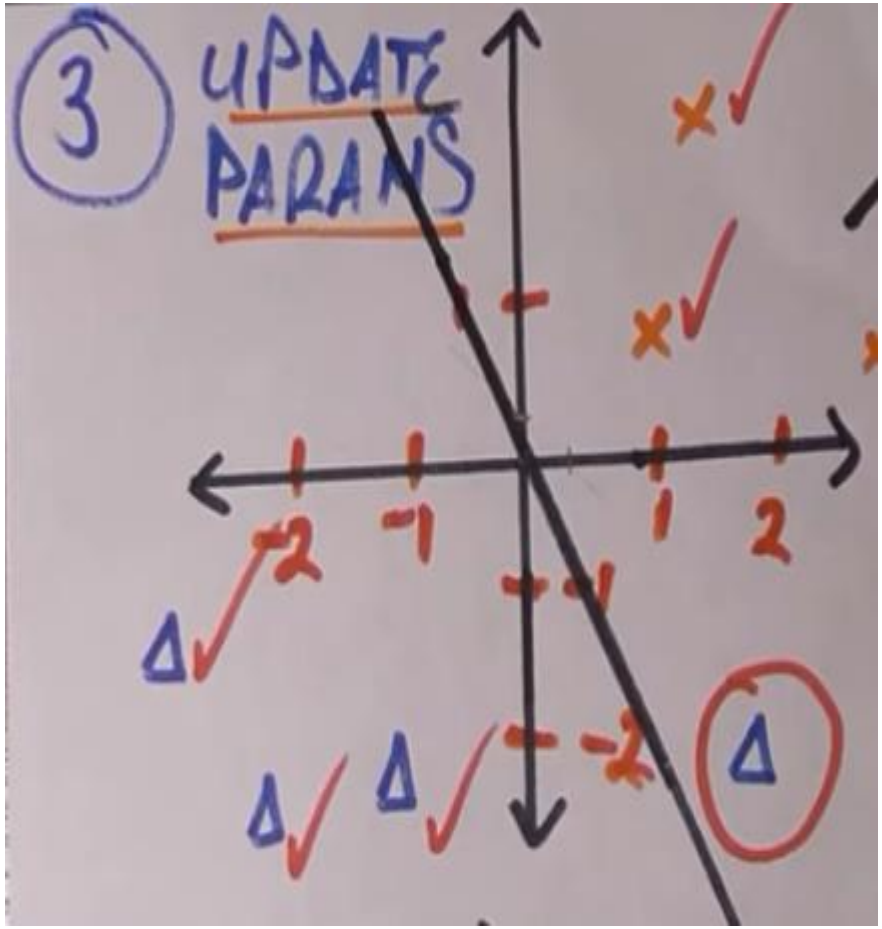
That's how we take omega that perceptron takes and draws a line to define two class.

Neural Network Architecture- Perceptron

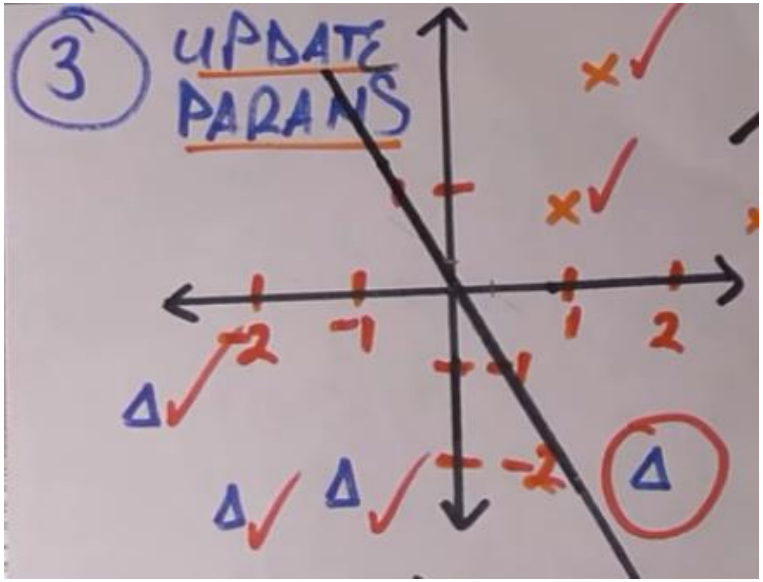
- As all data points are not classified correctly we need to be fixing this line to capture the division correctly.
- So we update our line using the learning rate as follows:



- Wi`- new omega
- Wi- old omega
- d- desired output –actual output{ 1 if misclassified point should be in upper , -1 if misclassified point should be lower}. Here d=-1
- Xi- mis-classified point (2,-2)



Neural Network Architecture- Perceptron



UPDATE STEP | $w_i' = w_i + \eta d x_i$

FOR EACH MISCLASSIFICATION, UPDATE

$w_0' = w_0 + \eta d x_0 = 0 + (.2)(-1)(1) = -0.2$

$w_1' = w_1 + \eta d x_1 = 1 + (.2)(-1)(2) = 0.6$

$w_2' = w_2 + \eta d x_2 = 0.5 + (.2)(-1)(-2) = 0.9$

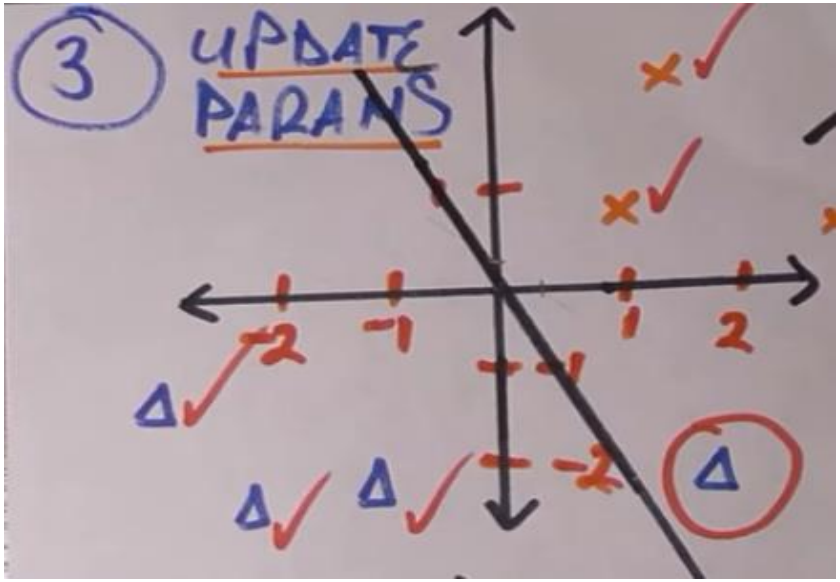
So we now get the update line equation as: -0.2x0+0.6x1+0.9x2

$\Rightarrow (-0.2)(1) + (0.6)(x_1) + (0.9)(x_2)$

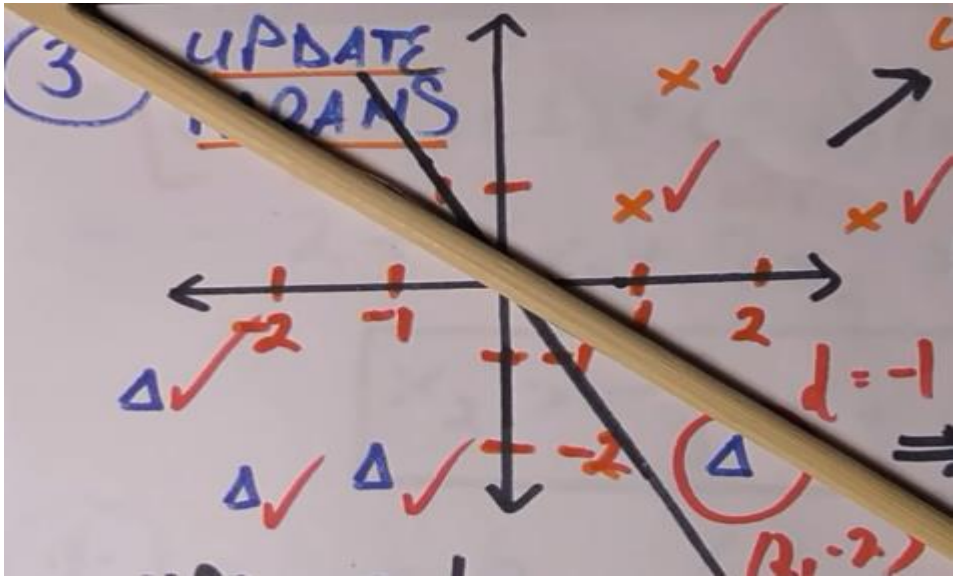
$= -.2 + .6x_1 + .9x_2 > 0 \Rightarrow$

$$x_2 > -\frac{2}{3}x_1 + \frac{2}{9}$$

Neural Network Architecture- Perceptron



For equation: X2>=-2X1



For equation: -0.2+0.6x1+0.9x2

Here we got now all points correctly classified but If were not then we will again take the misclassified point and update till we get all point correctly classified.

Introduction- Perceptron Network Model

- Implement AND function using perceptron networks for bipolar inputs and Targets? **Dec 2019 10 marks**

Introduction- Perceptron Network Model

1. Implement AND function using perceptron networks for bipolar inputs and targets.

Solution: Table 1 shows the truth table for AND function with bipolar inputs and targets:

Table 1

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

The perceptron network, which uses perceptron learning rule, is used to train the AND function. The network architecture is as shown in Figure 1. The input patterns are presented to the network one by one. When all the four input patterns are presented, then one epoch is said to be completed. The initial weights and threshold are set to zero, i.e., $w_1 = w_2 = b = 0$ and $\theta = 0$. The learning rate α is set equal to 1.

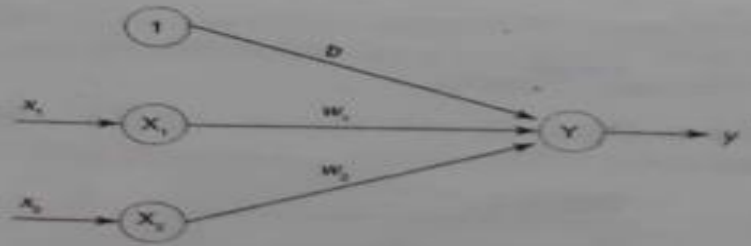


Figure 1 Perceptron network for AND function.

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

- Calculate the net input

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

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

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

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

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

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

Here, the change in weights are

$$\Delta w_1 = \alpha t x_1;$$
$$\Delta w_2 = \alpha t x_2;$$
$$\Delta b = \alpha t$$

The weights $w_1 = 1$, $w_2 = 1$, $b = 1$ are the final weights after first input pattern is presented. The same process is repeated for all the input patterns. The process can be stopped when all the targets become equal to the calculated output or when a separating line is obtained using the final weights for separating the positive responses from negative responses. Table 2 shows the training of perceptron network until its

Neural Network Architecture

Different types of Neural Networks? Dec 2019 5 marks

- Architecture deals with how neurons form layers and how they are interconnected.
- **There are Two types of Neural Network:**
  1. Feed forward or
  2. Feedback
- A layer is formed by combining processing elements
- A layer is a stage that links input stage and output stage.

Neural Network Architecture

1. **Feed Forward**:

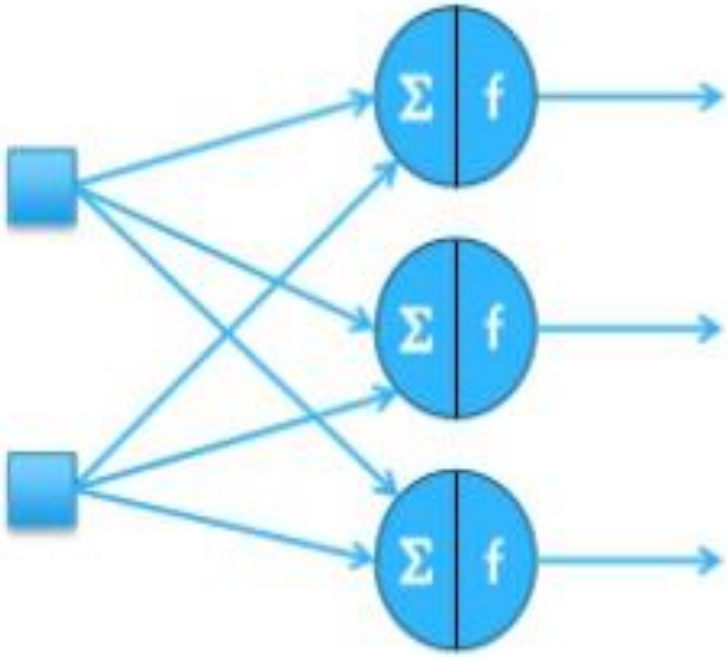
- A network is said to be a Feed-Forward Network if output layer is not connected to the same layer or preceding layers.
- It allow signals to travel one way only: from input to output.
- There are no feedback (loops); i.e., the output of any layer does not affect that same layer.
- Feed-forward ANNs tend to be straightforward networks that associate inputs with outputs.
- They are extensively used in pattern recognition.
- This type of organization is also referred to as bottom-up or top-down.

Neural Network Architecture- 1.1 Single Layer

Single Layer Feed-forward Network

Single Layer:
There is only one computational layer.

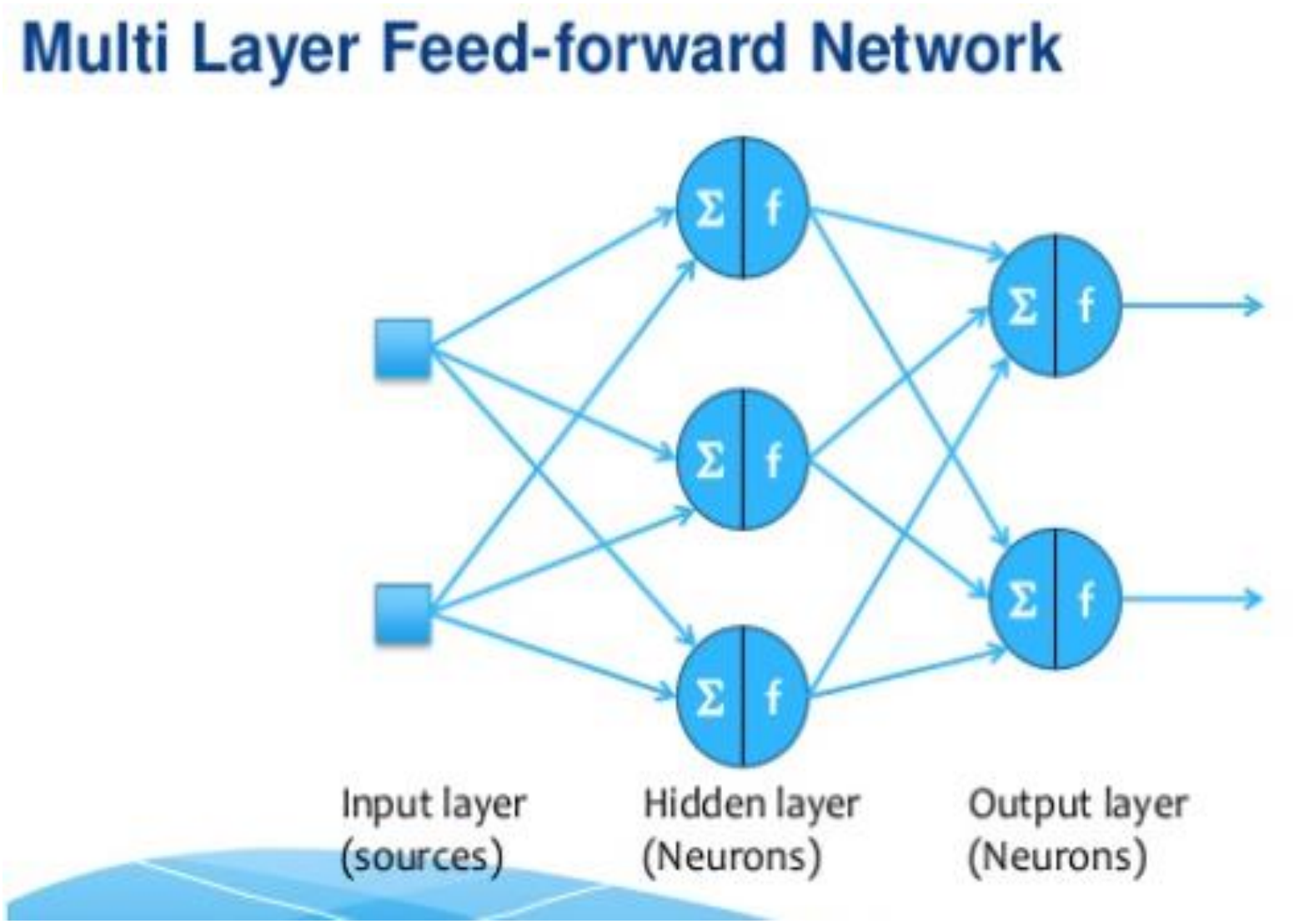
Feed-forward:
Input layer projects to the output layer not vice versa.



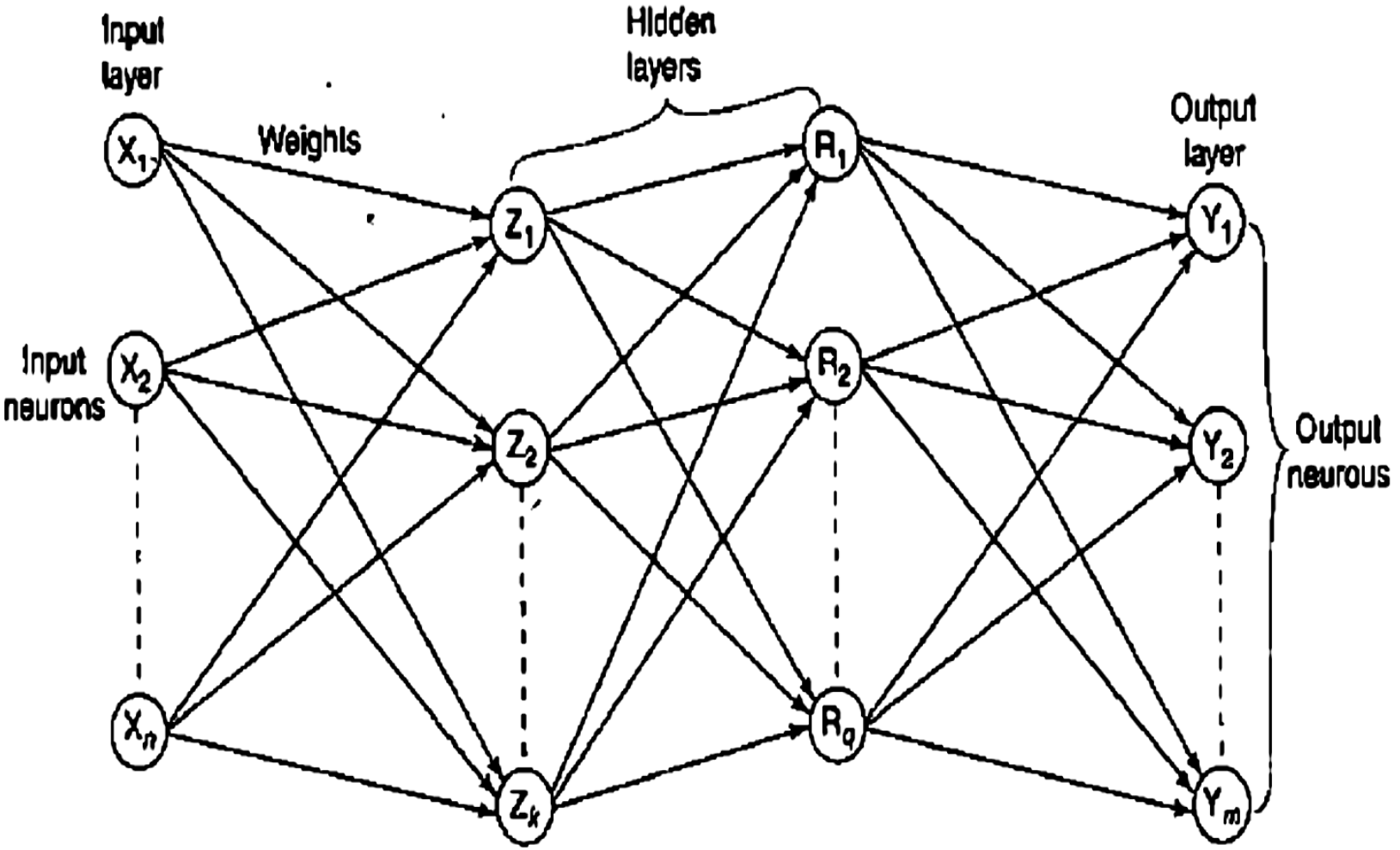
Neural Network Architecture- 1.1. Single Layer

- It is formed by the interconnection of input to output layer.
- Input layer receives input and it only buffers input signal
- Output layer generates output

Neural Network Architecture- 1.2. Multi Layer



Neural Network Architecture- 1.2. Multi Layer



Neural Network Architecture- 1.2. Multi Layer

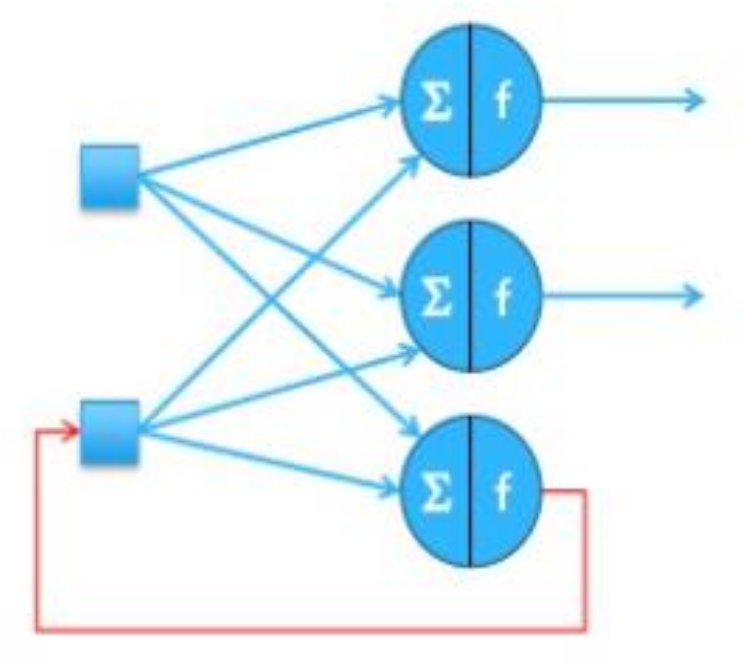
- It is formed by the interconnection of several layers.
- Input layer receives input and it only buffers input signal
- Output layer generates output
- Any layer between input and output is a hidden layer
- Hidden layer is internal to network and has no direct contact with the external environment.
  - No. of hidden layers : zero or more
  - More hidden layer ➔ more complexity but may provide an efficient output response

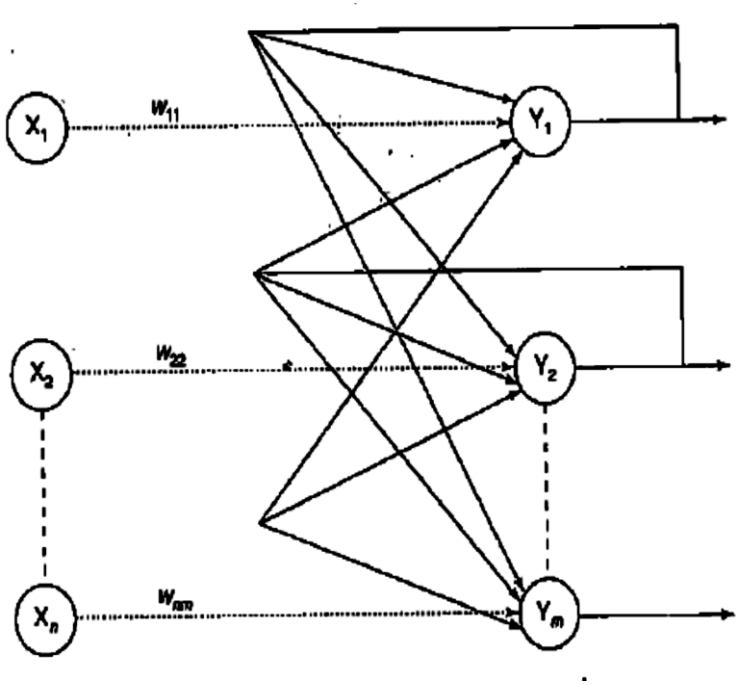
Neural Network Architecture- 2. Feedback/ Recurrent Network

- When outputs are connected back as inputs to same or preceding layer then it is a **feedback network.**
- If the feedback of the output is directed back to the same layer then it is called **lateral feedback.**
- **Recurrent networks** are feedback networks with closed loop.
- It can be single layer or multilayer.
  1. Single layer Recurrent network
  2. Multi layer Recurrent network

Neural Network Architecture- 2.1 Single Layer Recurrent Networks

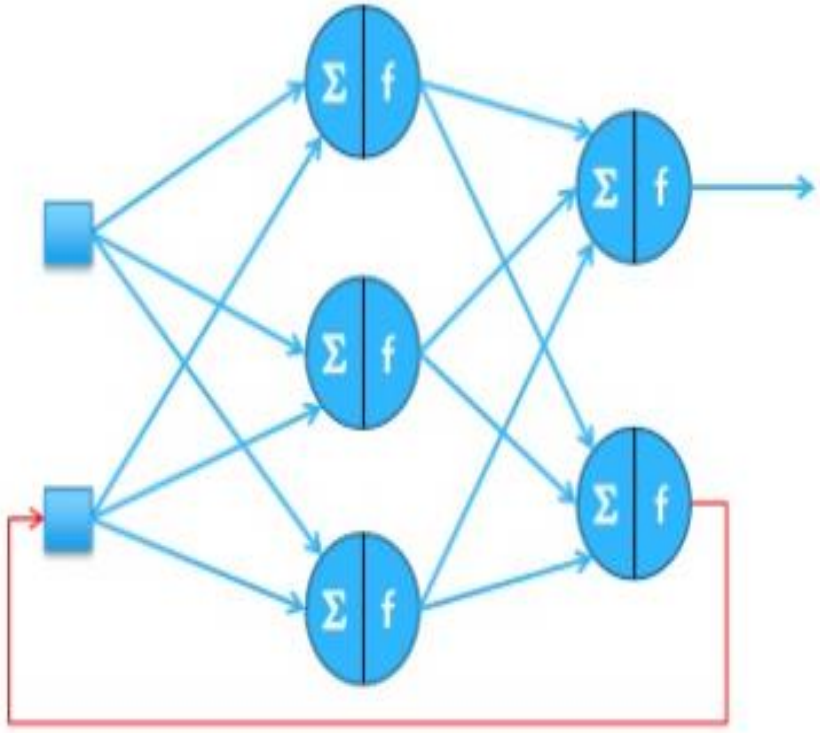
1. Single layer Recurrent network: Output processing element can be directed back to itself or other processing elements or both.

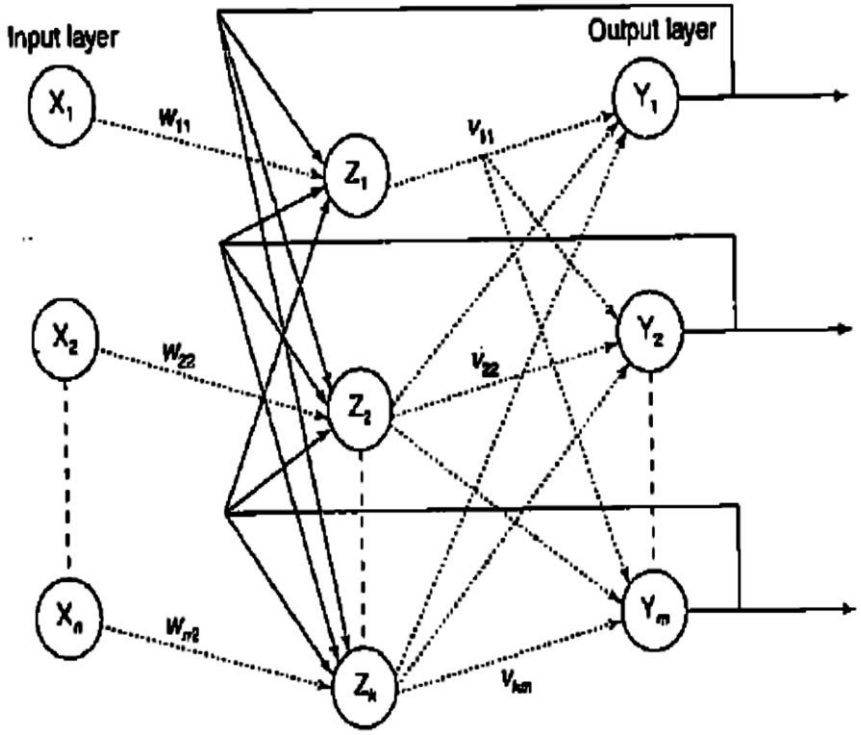




Neural Network Architecture- 2.2 Multi Layer Recurrent Networks

2. Multi layer Recurrent network: Output processing element is directed back to the nodes in a preceding layer or same layer.





Activation Function

Activation Function

- It is also known as **Transfer function**. It can also be attached between 2 neural networks.
- **Activation functions** are important for a Artificial Neural network to learn and understand complex patterns.
- The main function of it is to introduce non linear properties into the network.
- It calculates the weighted sum and adds direction and decides whether to fire a particular neuron or not.
- It convert an input signal of a node in a ANN to an output signal. This output signal now is used as a input in the next layer in the stack.

Need of Activation Function

- If we do not apply a Activation function then the output signal would simply be a simple ***linear function***.
- A *linear function* is just a polynomial of **one degree.** Now, a linear equation is easy to solve but they are limited in their complexity and have less power to learn complex functional mappings from data.
- A Neural Network without Activation function would simply be a **Linear regression Model,** which has limited power and does not performs good most of the times.
- *Also without activation function our Neural network would not be able to learn and model other complicated kinds of data such as images, videos , audio , speech etc.*

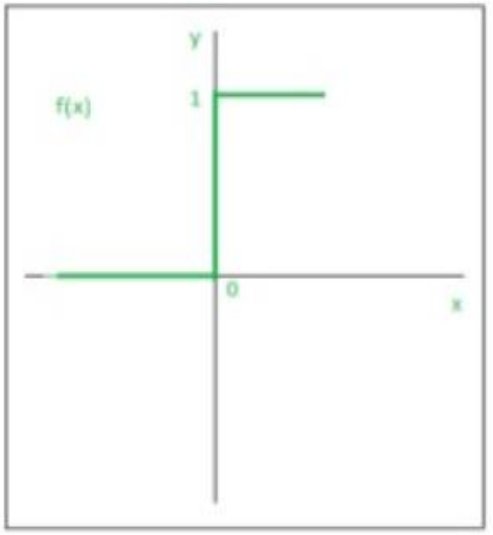
Types of Activation Function

- **There are 3 types of activation function:**
  1. Binary Step Function
  2. Linear or identity activation function
  3. Non Linear activation function

1. Binary Step function/ threshold based

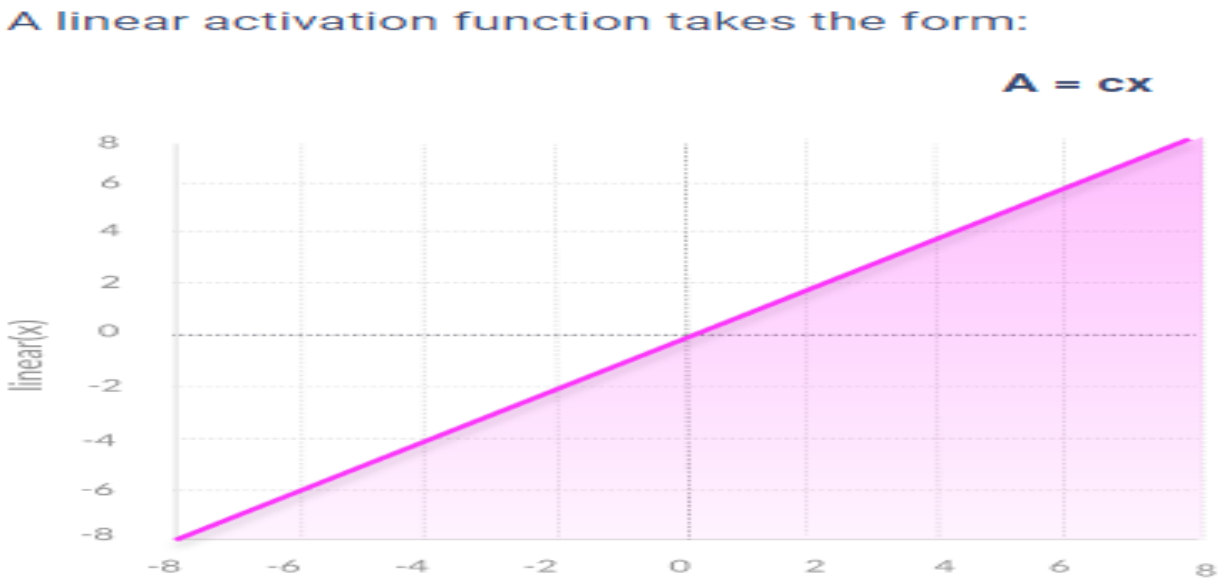
- **Step Function:**
  Step Function is one of the simplest kind of activation functions. In this, we consider a threshold value and if the value of net input say **y** is greater than the threshold then the neuron is activated.
- F(x)=1,if x>=0

The problem with a **step function** is that it does not allow multi-value outputs—for example, it cannot support classifying the inputs into one of several categories.



2. Linear or identity activation function

- **Equation :** Linear function has the equation similar to as of a straight line i.e. **y = ax**
- No matter how many layers we have, if all are linear in nature, the final activation function of last layer is nothing but just a linear function of the input of first layer.
- **Range :** -inf to +inf
- **Uses : Linear activation function** is used at just one place i.e. output layer.



Need of Non Linear activation function

A linear activation function has two major problems:

1. **Not possible to use** backpropagation (gradient descent) to train the model—the derivative of the function is a constant, and has no relation to the input, X. So it's not possible to go back and understand which weights in the input neurons can provide a better prediction.
2. **All layers of the neural network collapse into one**—with linear activation functions, no matter how many layers in the neural network, the last layer will be a linear function of the first layer (because a linear combination of linear functions is still a linear function). So a linear activation function turns the neural network into just one layer.

A neural network with a linear activation function is simply a linear regression model. It has limited power and ability to handle complexity varying parameters of input data.

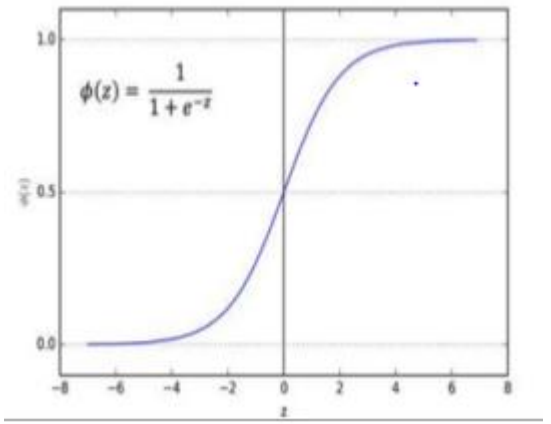
Non Linear activation function

- Modern neural network models use non-linear activation functions. They allow the model to create complex mappings between the network's inputs and outputs, which are essential for learning and modeling complex data, such as images, video, audio, and data sets which are non-linear or have high dimensionality.
- Almost any process imaginable can be represented as a functional computation in a neural network, provided that the activation function is non-linear.
- **Non-linear functions address the problems of a linear activation function:**

1. They **allow backpropagation** because they have a derivative function which is related to the inputs.
2. They **allow "stacking" of multiple layers of neurons** to create a deep neural network. Multiple hidden layers of neurons are needed to learn complex data sets with high levels of accuracy.

1. Sigmoid or Logistic activation function

- It is a function which is plotted as ‘S’ shaped graph.
- **Equation:** $A = 1/(1 + e^{-x})$
- **Nature :** Non-linear. Notice that X values lies between -8 to 8, Y values are very steep. This means, small changes in x would also bring about large changes in the value of Y.
- **Value Range :** 0 to 1
- **Uses :** Usually used in output layer of a binary classification, where result is either 0 or 1, as value for sigmoid function lies between 0 and 1 only so, result can be predicted easily to be 1 if value is greater than 0.5 and 0 otherwise.



1.Sigmoid or Logistic activation function

Disadvantages

1. **Vanishing gradient**—for very high or very low values of X, there is almost no change to the prediction, causing a vanishing gradient problem. This can result in the network refusing to learn further, or being too slow to reach an accurate prediction.
2. **Outputs not zero centered**.
3. **Computationally expensive**

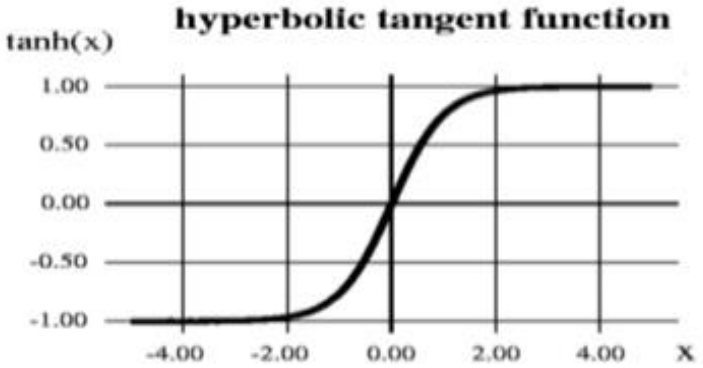
2. Hyperbolic tangent function

- The activation that works almost always better than sigmoid function is Tanh function also knows as **Tangent Hyperbolic function**. It's actually mathematically shifted version of the sigmoid function. Both are similar and can be derived from each other.
- **Equation:**

$$f(x) = \tanh(x) = 2/(1 + e^{-2x}) - 1$$

OR

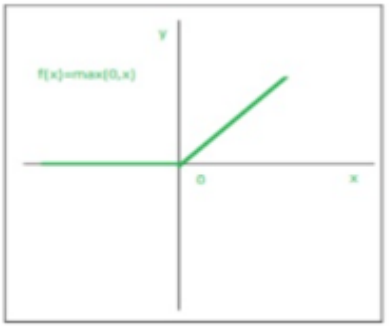
$$\tanh(x) = 2 * \text{sigmoid}(2x) - 1$$



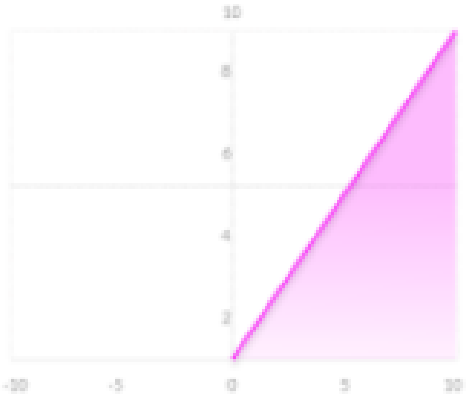
- **Value Range :** -1 to +1
- **Nature :** non-linear
- **Uses :-**Usually used in hidden layers of a neural network as it's values lies between **-1 to 1** hence the mean for the hidden layer comes out be 0 or very close to it, hence helps in centering the data by bringing mean close to 0. This makes learning for the next layer much easier.
- **Issue:** Suffers vanishing gradient problem

3.Relu (Rectified Linear Units)function

- **RELU :-** Stands for *Rectified linear unit*. It is the most widely used activation function. Chiefly implemented in *hidden layers* of Neural network.
- **Equation :-** ***A(x) = max(0,x)***. It gives an output x if x is positive and 0 otherwise. Hence it avoids and rectifies vanishing gradient problem
- **Value Range :-** [0, inf)
- **Nature :-** non-linear, which means we can easily back propagate the errors and have multiple layers of neurons being activated by the ReLu function.
- **Uses :-** ReLu is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations. At a time only a few neurons are activated making the network sparse making it efficient and easy for computation.
- In simple words, RELU learns much faster than sigmoid and Tanh function.



3.Relu (Rectified Linear Units)function



Advantages

- **Computationally efficient**—allows the network to converge very quickly
- **Non-linear**—although it looks like a linear function, ReLU has a derivative function and allows for backpropagation

Disadvantages

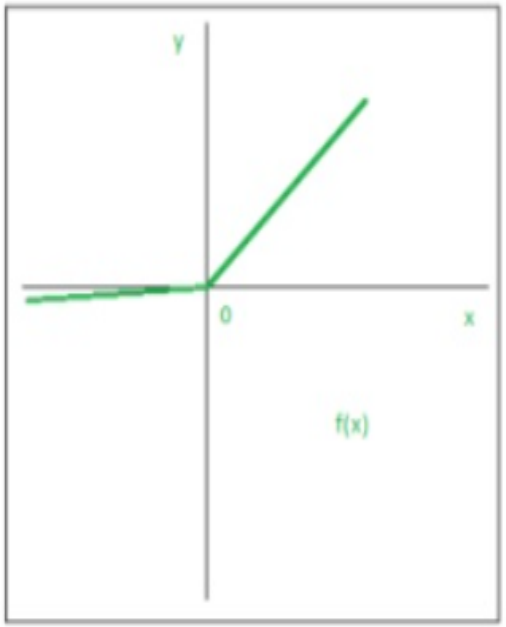
- **The Dying ReLU problem**—when inputs approach zero, or are negative, the gradient of the function becomes zero, the network cannot perform backpropagation and cannot learn.

4. Leaky Relu

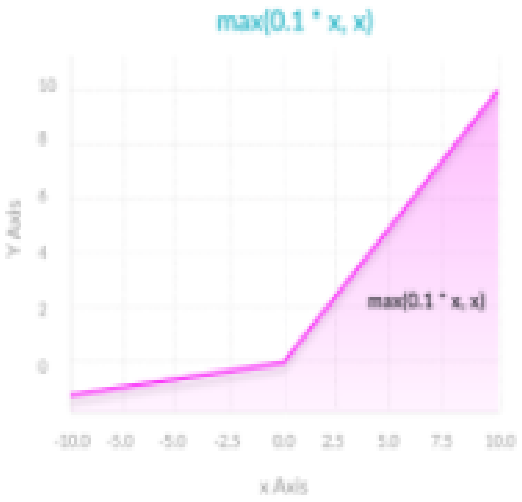
Leaky ReLU function is nothing but an improved version of the ReLU function.Instead of defining the Relu function as 0 for x less than 0, we define it as a small linear component of x. It can be defined as:

- F(x)=ax ,x<o
- Else x
- It introduces a small slope to keep the updates

alive.



4.Leaky Relu



Advantages

- **Prevents dying ReLU problem**—this variation of ReLU has a small positive slope in the negative area, so it does enable backpropagation, even for negative input values
- Otherwise like ReLU

Disadvantages

- **Results not consistent**—leaky ReLU does not provide consistent predictions for negative input values.

5. Soft Max

- **Soft max Function :-** The soft max function is also a type of sigmoid function but is handy when we are trying to handle classification problems.
- **Nature :-** non-linear
- **Uses :-** Usually used when trying to handle multiple classes. The soft max function would squeeze the outputs for each class between 0 and 1 and would also divide by the sum of the outputs.
- **Output:-** The soft max function is ideally used in the output layer of the classifier where we are actually trying to attain the probabilities to define the class of each input.

5. Soft Max

$$\sigma(z)j = \frac{ezj}{\sum_{k=1}^{k} ezk} \quad for\ j = 1....k.$$

Advantages

- **Able to handle multiple classes** only one class in other activation functions—normalizes the outputs for each class between 0 and 1, and divides by their sum, giving the probability of the input value being in a specific class.
- **Useful for output neurons**—typically Softmax is used only for the output layer, for neural networks that need to classify inputs into multiple categories.

Activation Function

- **CHOOSING THE RIGHT ACTIVATION FUNCTION**
  - The basic rule of thumb is if you really don't know what activation function to use, then simply use *RELU* as it is a general activation function and is used in most cases these days.
  - If your output is for binary classification then, *sigmoid function* is very natural choice for output layer.
- **Foot Note:**
  - The activation function does the non-linear transformation to the input making it capable to learn and perform more complex tasks.

Learning

Learning

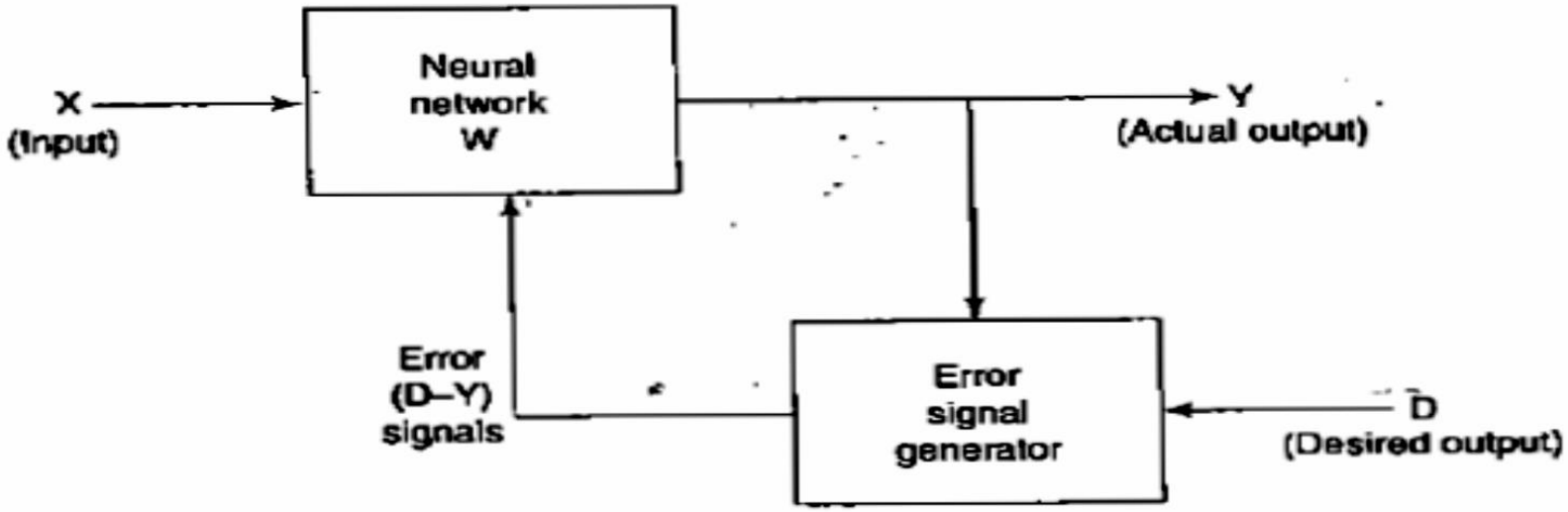
- Learning or training is a process by which a neural network adapts itself to a stimulus by making proper parameter adjustments resulting in the desired response.
- The mechanism based on which a neural network can adjust its weight.
- Learning can be classified into three categories as:
  1. **Supervised Learning**
  2. **Unsupervised Learning**
  3. **Reinforcement Learning.**

1. Supervised Learning

- Learning with the help of a teacher.
  - Like learning process of a small child : learns based on output he/she has to produce.
- Each input vector requires a corresponding target vector which is the desired output.
- Input vector with target vector is **training pair**.
- The network knows about output for each input.

1. Supervised Learning

- During training, the training pair is presented.
- **Actual output vector** is compared with **Desired output** to generate an **error signal**
- This error signal is used for adjustment of weights until the actual output matches the desired (target) output.

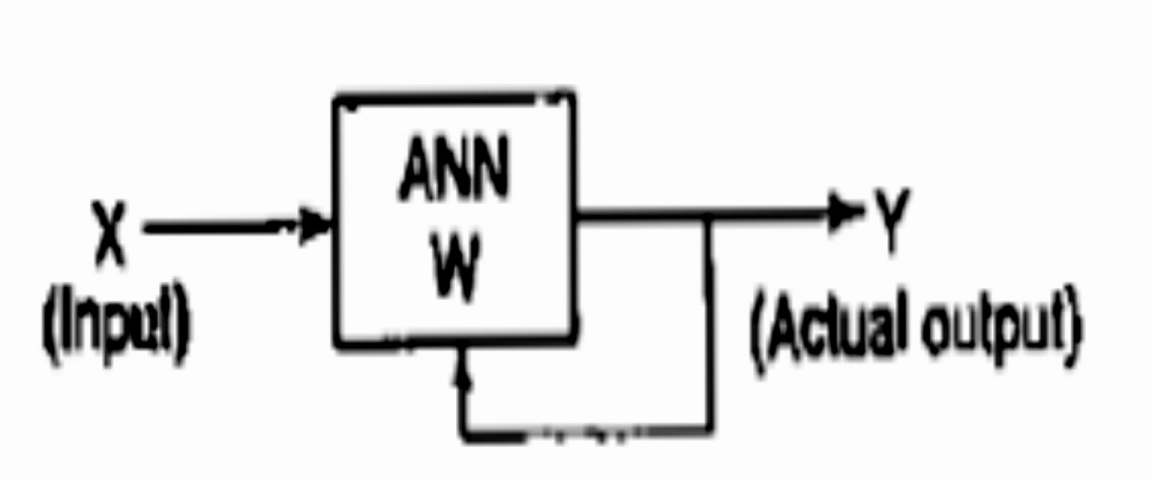


2. Un Supervised Learning

- Without the help of a teacher.
  - E.g. grouping of shopping items.
- Input vectors of similar types are grouped together
- In training process, the network receives input and organizes them to form clusters.
- When a new input pattern is applied, the neural network gives an output response indicating which class it belongs.
  - If a class cannot be found then a new class is generated.

2. Un Supervised Learning

- The network itself discover patterns, regularities, features or categories.
- While discovering these features, network undergoes change in its parameters. This process is called self-organizing.



3. Reinforcement Learning

- Similar to supervised learning
- In some cases, less information might be available.
  - For e.g. , the network might be told that its actual output is only “50%correct”.
- Here only critic information is available and not the exact information
- The feedback sent is the reinforcement signal.

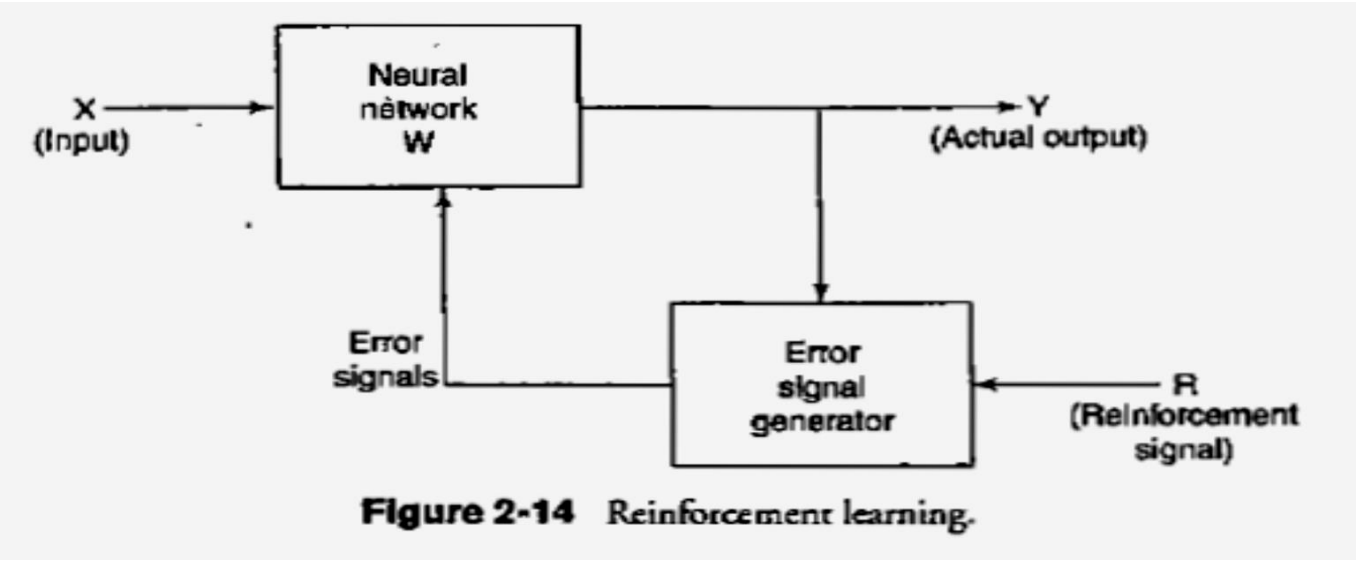
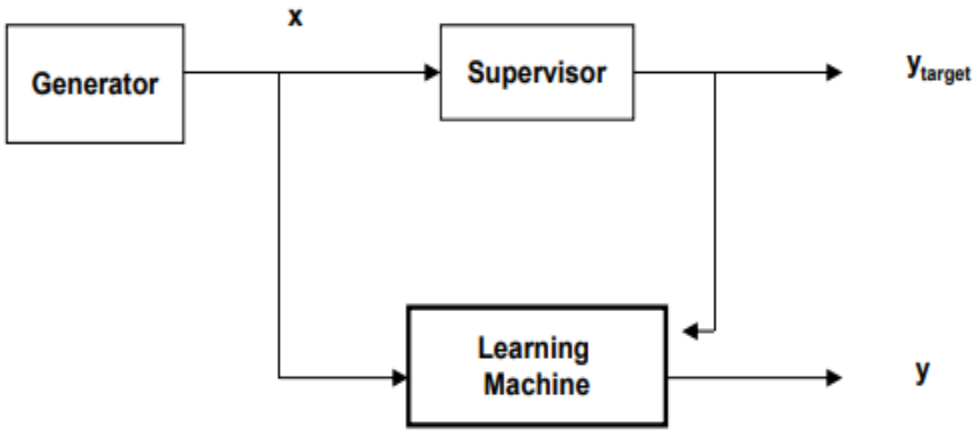


Figure 2-14 Reinforcement learning.

Supervised Learning

Supervised Learning

Review of Supervised Learning



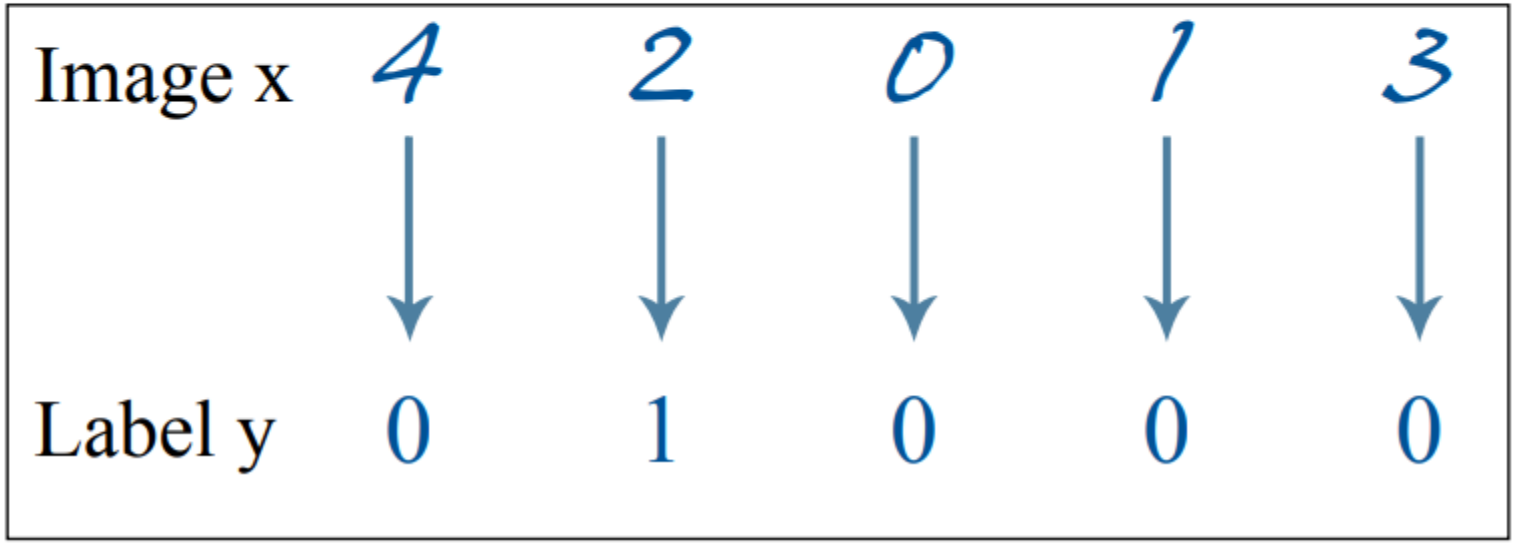
Training: Learn from training pairs ($\mathbf{x}$, $\mathbf{y_{target}}$)

Testing: Given $\mathbf{x}$, output a value y close to the supervisor's output y_{target}

Supervised Learning

Example: handwritten digits

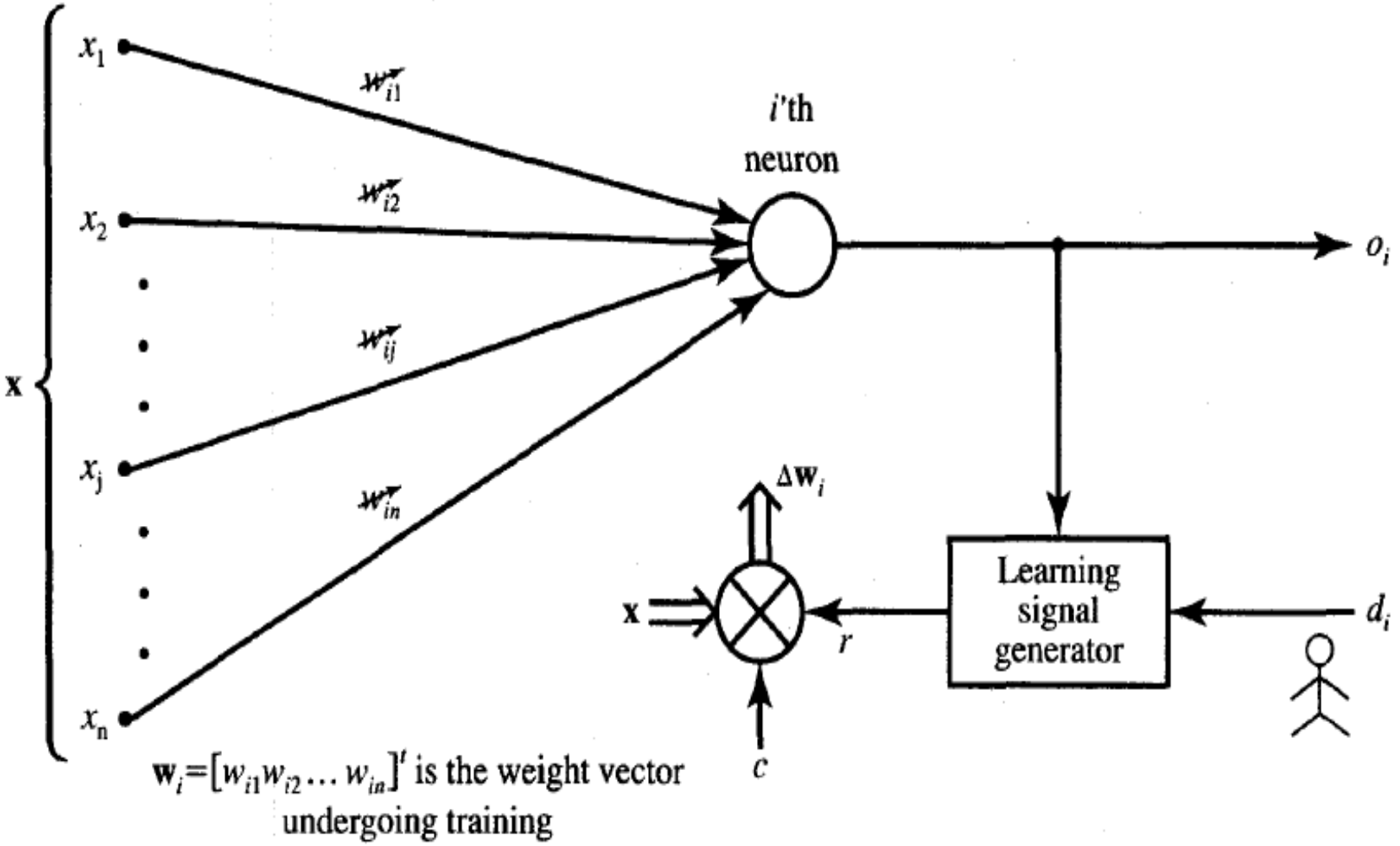
- Find a perceptron that detects “two”s.



Supervised Learning- NN Learning Rule

- A neuron is an adaptive element.
- Its weights are modifiable depending on
  - the input signal it receives,
  - its output value, and
  - the associated teacher response (not available in unsupervised learning).

Supervised Learning- NN Learning Rule



Supervised Learning- NN Learning Rule

- The weight vector $w_i=[w_{i1}\ w_{i2}\ w_{i3}\ \ldots\ w_{in}]^t$ increases in proportion to the product of input (x) and learning signal (r).
- The learning signal r is in general a function of wi,x, and sometimes of the teacher's signal d,

$$r = r(\mathbf{w}_i, \mathbf{x}, d_i)$$

- The increment of the weight vector (w_i) produced by the learning step at time (t) is given by

$$\Delta \mathbf{w}_i(t) = cr\left[\mathbf{w}_i(t), \mathbf{x}(t), d_i(t)\right]\mathbf{x}(t)$$

Here **c** is learning constant that determines the rate of learning

Supervised Learning- NN Learning Rule

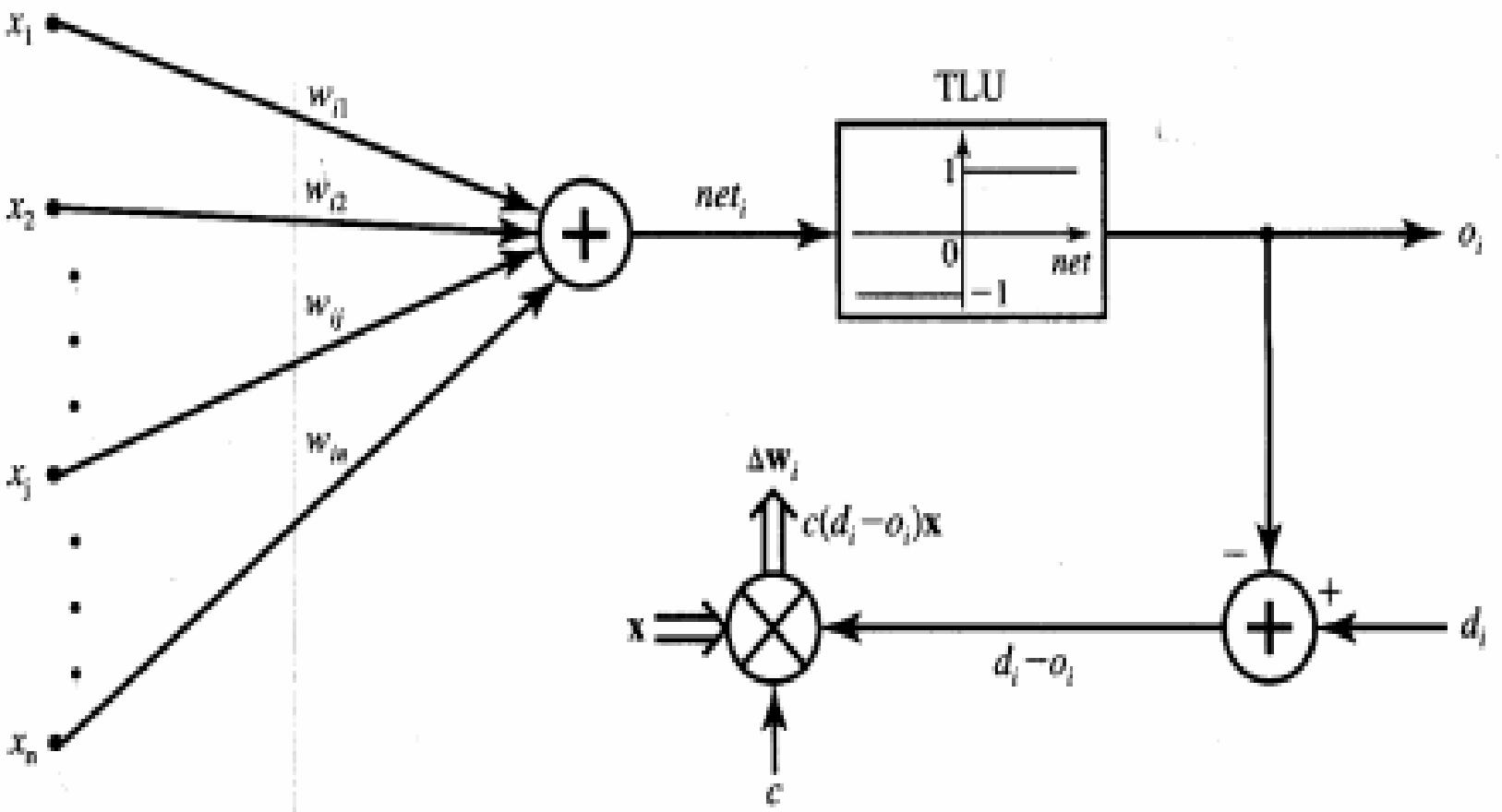
- At the next instant learning step, the weight vector adapted at time (t) becomes:

$$\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + cr\left[\mathbf{w}_i(t), \mathbf{x}(t), d_i(t)\right]\mathbf{x}(t)$$

- The superscript convention will be used in this context to index the discrete – time training steps. For the k^{th} step, we have:

$$\mathbf{w}_i^{k+1} = \mathbf{w}_i^k + cr(\mathbf{w}_i^k, \mathbf{x}^k, d_i^k)\mathbf{x}^k$$

Supervised Learning- Perceptron learning Rule



Supervised Learning- Perceptron learning Rule

- the learning signal is the difference between the desired and actual neuron's response.

$$r \triangleq d_i - o_i$$

- learning is supervised
- applicable only for binary neuron response.
- Weight adjustment : $\Delta \mathbf{w}_i = c\,[d_i - \mathrm{sgn}\,(\mathbf{w}_i^t \mathbf{x})]\,\mathbf{x}$
- The weights are initialized at any values in this method.

Supervised Learning- Perceptron learning Rule

- Implement the Perceptron rule training using c=0.1, and using bipolar binary activation function, with the given following data:

$$\mathbf{x}_1 = \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix}, \quad \mathbf{x}_2 = \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix}, \quad \mathbf{x}_3 = \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix} \quad \mathbf{w}^1 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix}$$

$$d_1 = -1, d_2 = -1, \text{and } d_3 = 1$$

Supervised Learning- Perceptron learning Rule

Step 1 Input is $\mathbf{x}_1$, desired output is d_1:

$$net^1 = \mathbf{w}^{1t}\mathbf{x}_1 = \begin{bmatrix} 1 & -1 & 0 & 0.5 \end{bmatrix} \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix} = 2.5$$

Correction in this step is necessary since $d_1 \neq \text{sgn}(2.5)$. We thus obtain updated weight vector

$$\mathbf{w}^2 = \mathbf{w}^1 + 0.1(-1 - 1)\mathbf{x}_1$$

Plugging in numerical values we obtain

$$\mathbf{w}^2 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix} - 0.2 \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix} = \begin{bmatrix} 0.8 \\ -0.6 \\ 0 \\ 0.7 \end{bmatrix}$$

$$f(net) \stackrel{\Delta}{=} \text{sgn}(net) = \begin{cases} +1, & net > 0 \\ -1, & net < 0 \end{cases}$$

Supervised Learning- Perceptron learning Rule

Step 2 Input is $\mathbf{x}_2$, desired output is d_2. For the present weight vector $\mathbf{w}^2$ we compute the activation value net^2 as follows:

$$net^2 = \mathbf{w}^{2t}\mathbf{x}_2 = \begin{bmatrix} 0 & 1.5 & -0.5 & -1 \end{bmatrix} \begin{bmatrix} 0.8 \\ -0.6 \\ 0 \\ 0.7 \end{bmatrix} = -1.6$$

Correction is not performed in this step since $d_2 = \text{sgn}(-1.6)$

Supervised Learning- Perceptron learning Rule

Step 3 Input is $\mathbf{x}_3$, desired output is d_3, present weight vector is $\mathbf{w}^3$. Computing net^3 we obtain:

$$net^3 = \mathbf{w}^{3t}\mathbf{x}_3 = \begin{bmatrix} -1 & 1 & 0.5 & -1 \end{bmatrix} \begin{bmatrix} 0.8 \\ -0.6 \\ 0 \\ 0.7 \end{bmatrix} = -2.1$$

Correction is necessary in this step since $d_3 \neq \text{sgn}(-2.1)$. The updated weight values are

$$\mathbf{w}^4 = \mathbf{w}^3 + 0.1(1 + 1)\mathbf{x}_3$$

or

$$\mathbf{w}^4 = \begin{bmatrix} 0.8 \\ -0.6 \\ 0 \\ 0.7 \end{bmatrix} + 0.2 \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix} = \begin{bmatrix} 0.6 \\ -0.4 \\ 0.1 \\ 0.5 \end{bmatrix}$$

Supervised Learning- Perceptron learning Rule

This terminates the sequence of learning steps unless the training set is recycled. It is not a coincidence that the fourth component of $\mathbf{x}_1$, $\mathbf{x}_2$, and $\mathbf{x}_3$ in this example is invariable and equal to -1. Perceptron learning requires fixing of one component of the input vector, although not necessarily at the

Supervised Learning- Perceptron learning Rule

- Implement the Perceptron rule training using c = 1, and using bipolar binary activation function, with the given following data. Show that the network is learning

$$w^1 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \quad x_1 = \begin{bmatrix} 2 \\ 1 \\ -1 \end{bmatrix} \quad x_2 = \begin{bmatrix} 0 \\ -1 \\ -1 \end{bmatrix}$$

$$d_1 = -1 \qquad d_2 = 1$$

Supervised Learning- Delta learning Rule

- Only valid for **continuous** activation function
- Used in **supervised** training mode
- Learning signal is called delta
- The aim of the delta rule is to minimize the error over all training patterns

$$r \triangleq [d_i - f(\mathbf{w}_i^t \mathbf{x})] f'(\mathbf{w}_i^t \mathbf{x})$$

Supervised Learning- Delta learning Rule

- Learning rule is derived from least squared error:

$$E \triangleq \frac{1}{2}(d_i - o_i)^2 = \frac{1}{2}\left[d_i - f(\mathbf{w}_i^t\mathbf{x})\right]^2$$

- Calculating the gradient vector with respect to w_i

$$\nabla E = -(d_i - o_i) f'(\mathbf{w}_i^t\mathbf{x})\mathbf{x}$$

$$\Delta \mathbf{w}_i = -\eta \nabla E$$

$$\Delta \mathbf{w}_i = \eta(d_i - o_i) f'(net_i)\mathbf{x}$$

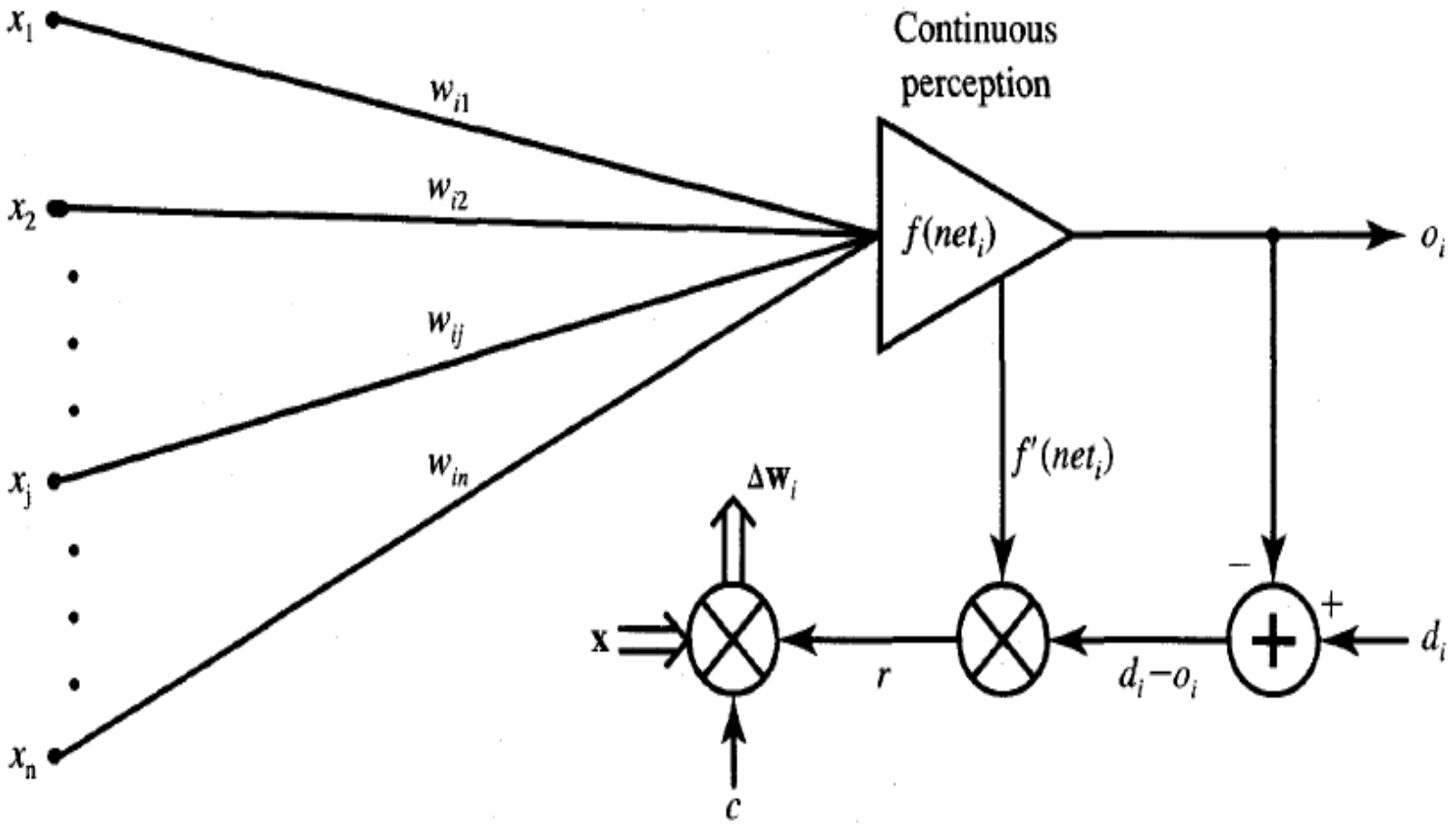
Supervised Learning- Delta learning Rule

- The weight adjustment becomes

$$\Delta \mathbf{w}_i = c(d_i - o_i) f'(\text{net}_i)\mathbf{x}$$

- The weights are initialized at any values for this method of training.
- It is also called continuous *perceptron training rule.*

Supervised Learning- Delta learning Rule



Supervised Learning- Delta learning Rule

- Implement the Delta rule training using c=0.1, and using bipolar binary activation function, with the given following data:

$$\mathbf{x}_1 = \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix}, \quad \mathbf{x}_2 = \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix}, \quad \mathbf{x}_3 = \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix} \quad \mathbf{w}^1 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix}$$

$$d_1 = -1, d_2 = -1, \text{and } d_3 = 1$$

Supervised Learning- Delta learning Rule

O= $$f(net) \triangleq \frac{2}{1+\exp(-\lambda net)} - 1$$

$$f'(\text{net}) = \frac{1}{2}(1 - o^2)$$

$$\Delta \mathbf{w}_i = c(d_i - o_i) f'(\text{net}_i)\mathbf{x}$$

C=learning rate given as 0.1

$$\mathbf{w}^2 = c(d_1 - o^1) f'(net^1)\mathbf{x}_1 + \mathbf{w}^1$$

Supervised Learning- Delta learning Rule

Step 1 Input is vector $\mathbf{x}_1$, initial weight vector is $\mathbf{w}^1$:

$$net^1 = \mathbf{w}^{1t}\mathbf{x}_1 = 2.5$$

$$o^1 = f(net^1) = 0.848$$

$$f'(net^1) = \frac{1}{2}[1 - (o^1)^2] = 0.140$$

$$\mathbf{w}^2 = c(d_1 - o^1)f'(net^1)\mathbf{x}_1 + \mathbf{w}^1$$

$$= [0.974 \quad -0.948 \quad 0 \quad 0.526]^t$$

Supervised Learning- Delta learning Rule

Step 2 Input is vector $\mathbf{x}_2$, weight vector is $\mathbf{w}^2$:

$$net^2 = \mathbf{w}^{2t}\mathbf{x}_2 = -1.948$$

$$o^2 = f(net^2) = -0.75$$

$$f'(net^2) = \frac{1}{2}[1 - (o^2)^2] = 0.218$$

$$\mathbf{w}^3 = c(d_2 - o^2)f'(net^2)\mathbf{x}_2 + \mathbf{w}^2$$

$$= [0.974 \quad -0.956 \quad 0.002 \quad 0.531]^t$$

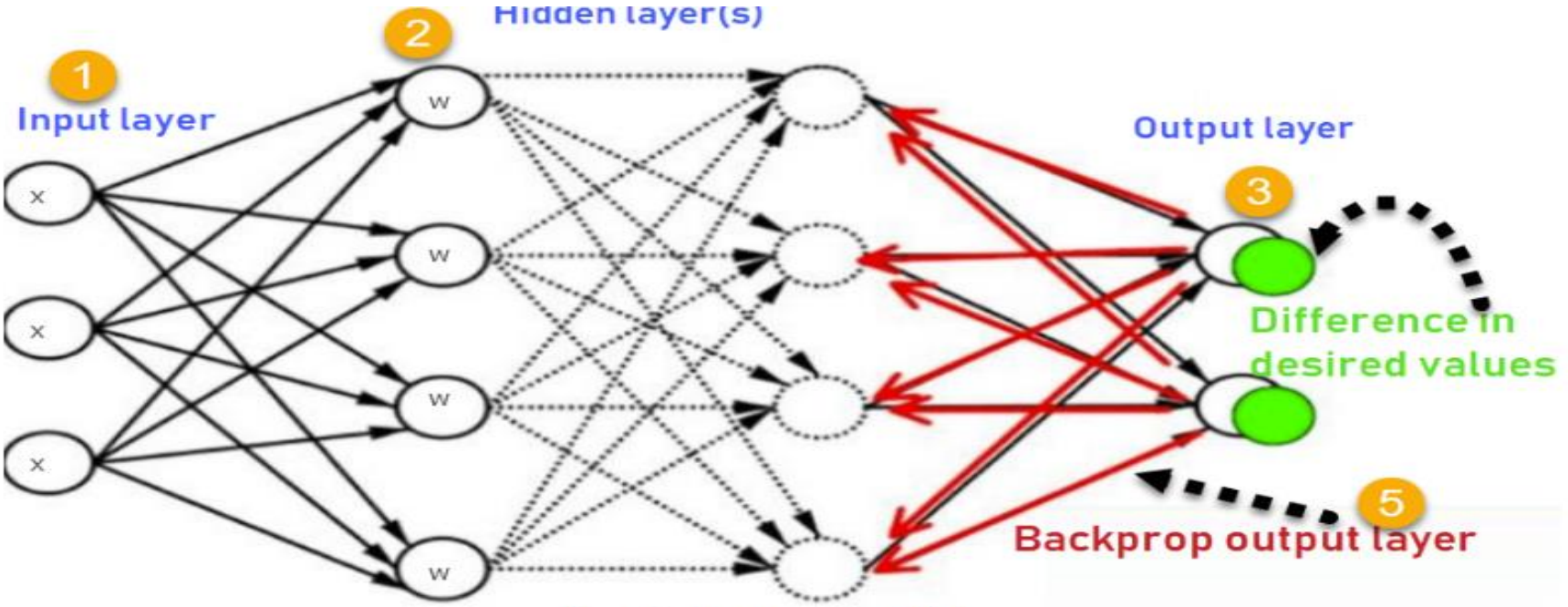
Supervised Learning- Delta learning Rule

Step 3 Input is x_3, weight vector is $\mathbf{w}^3$:

$$net^3 = \mathbf{w}^{3t}\mathbf{x}_3 = -2.46$$
$$o^3 = f(net^3) = -0.842$$
$$f'(net^3) = \frac{1}{2}[1 - (o^3)^2] = 0.145$$
$$\mathbf{w}^4 = c(d_3 - o^3)f'(net^3)\mathbf{x}_3 + \mathbf{w}^3$$
$$= [0.947 \quad -0.929 \quad 0.016 \quad 0.505]^t$$

Obviously, since the desired values are ± 1 in this example, the corrections will be performed in each step, since $d_i - f(net_i) \neq 0$ throughout the entire training. This method usually requires small c values, since it is based on moving the weight vector in the weight space in the negative error gradient direction. ■

Supervised Learning- Back propagation Algo



How Backpropagation Works

1. Inputs X, arrive through the preconnected path
2. Input is modeled using real weights W. The weights are usually randomly selected.
3. Calculate the output for every neuron from the input layer, to the hidden layers, to the output layer.
4. Calculate the error in the outputs $Error_B = Actual\ Output - Desired\ Output$
5. Travel back from the output layer to the hidden layer to adjust the weights such that the error is decreased.

Keep repeating the process until the desired output is achieved

Supervised Learning- Back propagation Algo

The Back propagation algorithm looks for the minimum value of the error function in weight space using a technique called the **delta rule or gradient descent**. The weights that minimize the error function is then considered to be a solution to the learning problem.

Let's understand how it works with an example: You have a dataset, which has labels. Consider the below table:

Input	Desired Output
0	0
1	2
2	4

Now the output of your model when 'W" value is 3:

Input	Desired Output	Model output (W=3)
0	0	0
1	2	3
2	4	6

Notice the difference between the actual output and the desired output:

Input	Desired Output	Model output (W=3)	Absolute Error	Square Error
0	0	0	0	0
1	2	3	1	1
2	4	6	2	4

Supervised Learning- Back propagation Algo

Let's change the value of 'W'. Notice the error when 'W' = '4'

Input	Desired Output	Model output (W=3)	Absolute Error	Square Error	Model output (W=4)	Square Error
0	0	0	0	0	0	0
1	2	3	1	1	4	4
2	4	6	2	4	8	16

Now if you notice, when we increase the value of 'W' the error has increased. So, obviously there is no point in increasing the value of 'W' further. But, what happens if I decrease the value of 'W'? Consider the table below:

Input	Desired Output	Model output (W=3)	Absolute Error	Square Error	Model output (W=2)	Square Error
0	0	0	0	0	0	0
1	2	3	2	4	3	0
2	4	6	2	4	4	0

Supervised Learning- Back propagation Algo

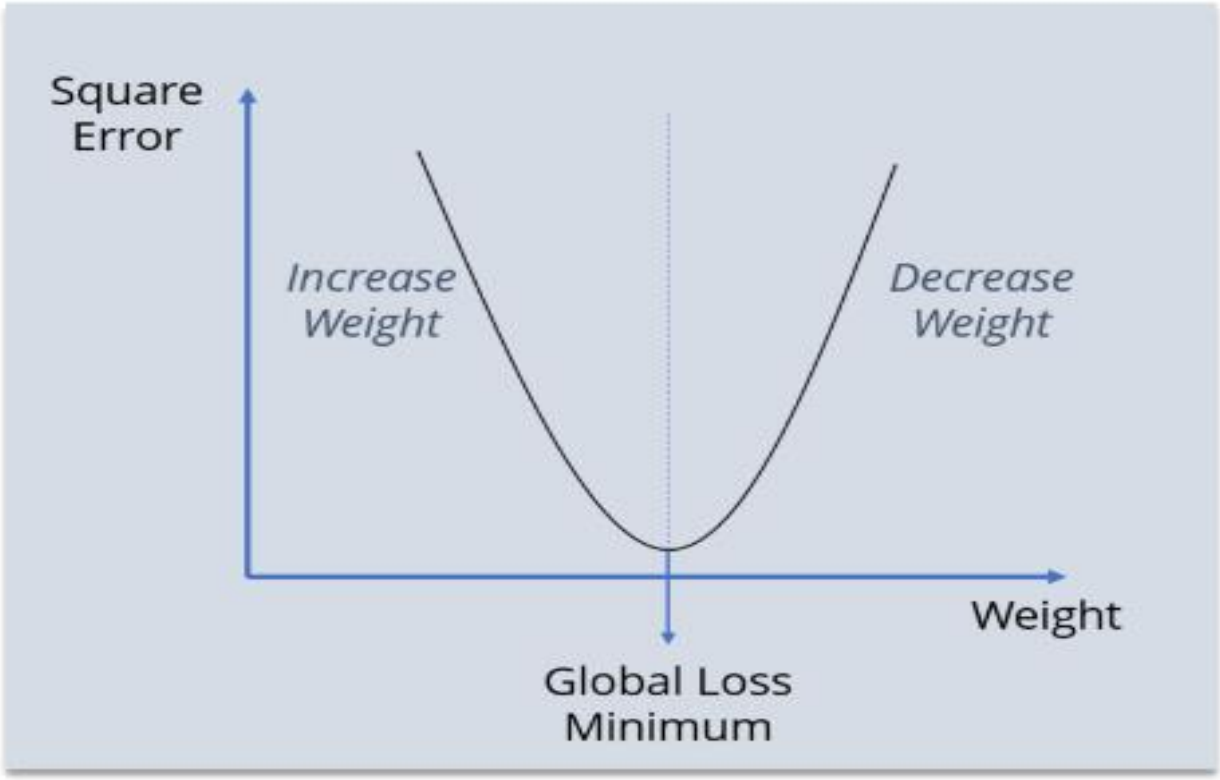
Now, what we did here:

- We first initialized some random value to 'W' and propagated forward.
- Then, we noticed that there is some error. To reduce that error, we propagated backwards and increased the value of 'W'.
- After that, also we noticed that the error has increased. We came to know that, we can't increase the 'W' value.
- So, we again propagated backwards and we decreased 'W' value.
- Now, we noticed that the error has reduced.

So, we are trying to get the value of weight such that the error becomes minimum. Basically, we need to figure out whether we need to increase or decrease the weight value. Once we know that, we keep on updating the weight value in that direction until error becomes minimum. You might reach a point, where if you further update the weight, the error will increase. At that time you need to stop, and that is your final weight value.

Supervised Learning- Back propagation Algo

Consider the graph below:



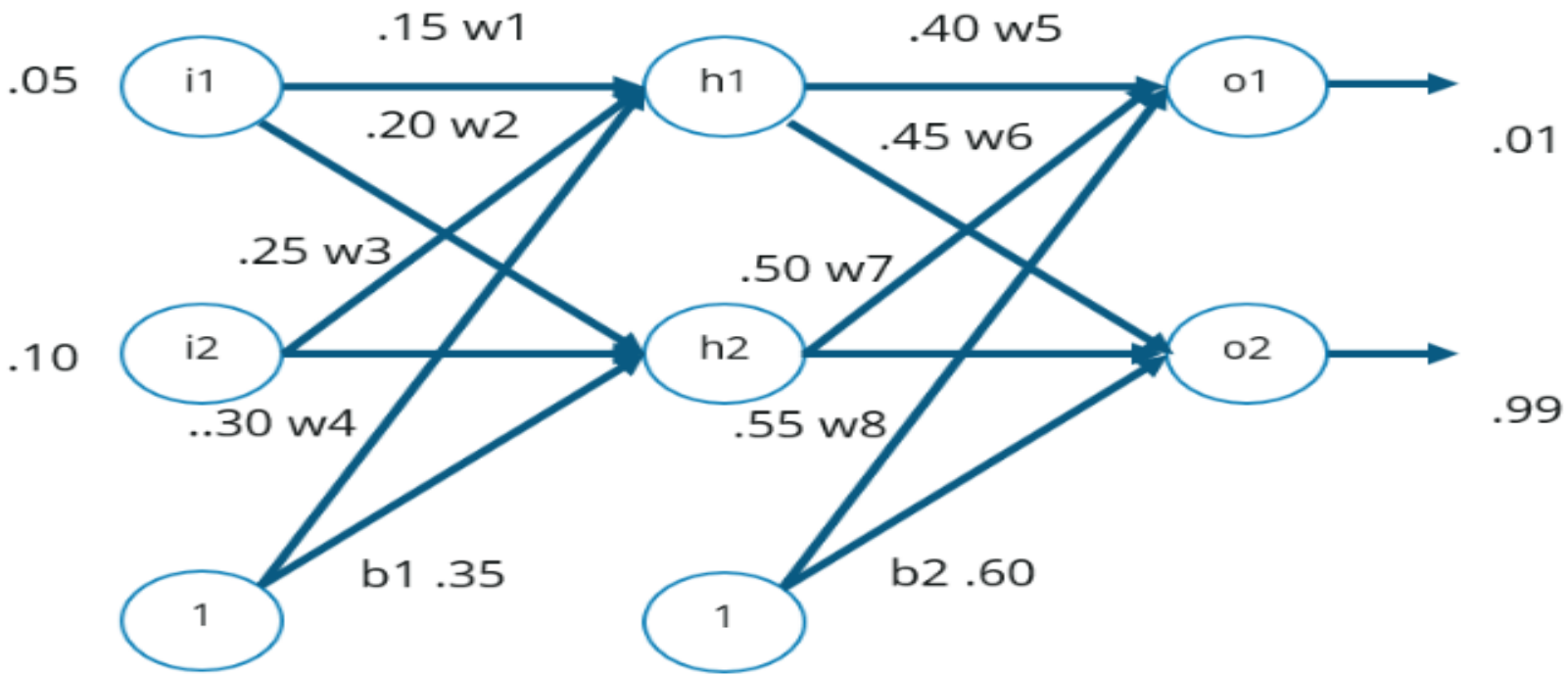
We need to reach the 'Global Loss Minimum'.

This is nothing but Backpropagation.

Supervised Learning- Back propagation Algo

How Backpropagation Works?

Consider the below Neural Network:



The above network contains the following:

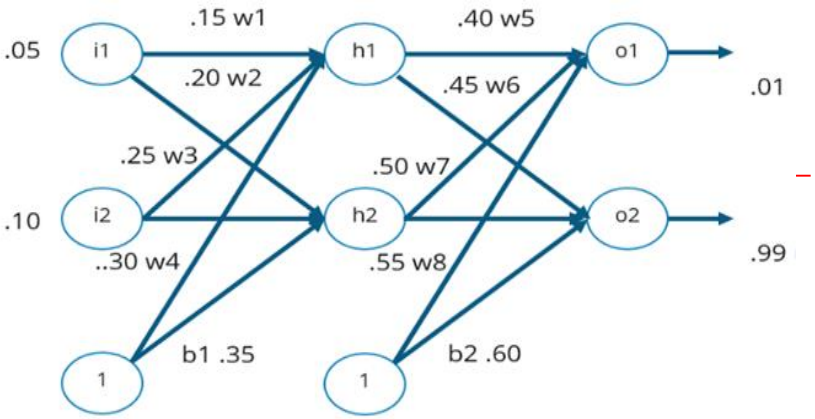
- two inputs
- two hidden neurons
- two output neurons
- two biases

Supervised Learning- Back propagation Algo

Below are the steps involved in Backpropagation:

- Step – 1: Forward Propagation
- Step – 2: Backward Propagation
- Step – 3: Putting all the values together and calculating the updated weight value

Supervised Learning- Back



Step – 1: Forward Propagation

We will start by propagating forward.

Net Input For h1:

net h1 = w1*i1 + w2*i2 + b1*1 → net h1 = 0.15*0.05 + 0.2*0.1 + 0.35*1 = 0.3775

Output Of h1:

out h1 = $1/1+e^{-\text{net h1}}$ → $1/1+e^{.3775}$ = 0.593269992

Output Of h2:

out h2 = 0.596884378

We will repeat this process for the output layer neurons, using the output from the hidden layer neurons as inputs.

Output For o1:

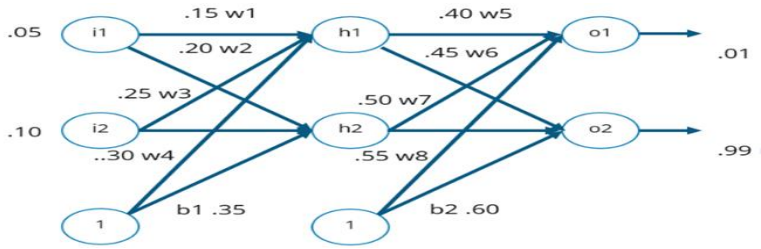
net o1 = w5*out h1 + w6*out h2 + b2*1 → 0.4*0.593269992 + 0.45*0.596884378 + 0.6*1 = 1.105905967

Out o1 = $1/1+e^{-\text{net o1}}$ → $1/1+e^{-1.105905967}$ = 0.75136507

Output For o2:

Out o2 = 0.772928465

Supervised Learning- Back propagation Algo



Now, let's see what is the value of the error:

Error For o1:

$E_{o1} = \Sigma 1/2(target - output)^2$ → $\frac{1}{2}(0.01 - 0.75136507)^2 = 0.274811083$

Error For o2:

$E_{o2} = 0.023560026$

Total Error:

$E_{total} = E_{o1} + E_{o2}$ → $0.274811083 + 0.023560026 = 0.298371109$

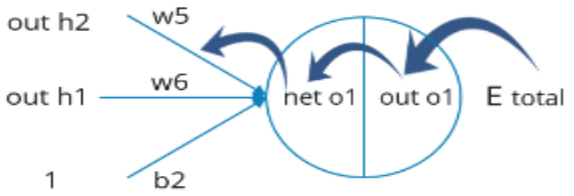
Supervised Learning- Back propagation Algo

Step – 2: Backward Propagation

Now, we will propagate backwards. This way we will try to reduce the error by changing the values of weights and biases.

Consider W5, we will calculate the rate of change of error w.r.t change in weight W5.

$$\frac{\delta Etotal}{\delta w5} = \frac{\delta Etotal}{\delta out\ o1} * \frac{\delta out\ o1}{\delta net\ o1} * \frac{\delta net\ o1}{\delta w5}$$



Since we are propagating backwards, first thing we need to do is, calculate the change in total errors w.r.t the output O1 and O2.

$$E_{total} = 1/2(target\ o1 - out\ o1)^2 + 1/2(target\ o2 - out\ o2)^2$$

$$\frac{\delta Etotal}{\delta out\ o1} = -(target\ o1 - out\ o1) = -(0.01 - 0.75136507) = 0.74136507$$

Now, we will propagate further backwards and calculate the change in output O1 w.r.t to its total net input.

$$out\ o1 = 1/1+e^{-neto1}$$

$$\frac{\delta out\ o1}{\delta net\ o1} = out\ o1\ (1 - out\ o1) = 0.75136507\ (1 - 0.75136507) = 0.186815602$$

Supervised Learning- Back propagation Algo

Let's see now how much does the total net input of O1 changes w.r.t W5?

$$\text{net o1} = w5 * \text{out h1} + w6 * \text{out h2} + b2 * 1$$

$$\frac{\delta net\ o1}{\delta w5} = 1 * \text{out h1}\ w5^{(1-1)} + 0 + 0 = 0.593269992$$

Step – 3: Putting all the values together and calculating the updated weight value

Now, let's put all the values together:

$$\frac{\delta Etotal}{\delta w5} = \frac{\delta Etotal}{\delta out\ o1} * \frac{\delta out\ o1}{\delta net\ o1} * \frac{\delta net\ o1}{\delta w5} \rightarrow 0.082167041$$

Let's calculate the updated value of W5:

$$w5^{+} = w5 - n\frac{\delta Etotal}{\delta w5} \rightarrow w5^{+} = 0.4 - 0.5 * 0.082167041$$

Updated w5 → 0.35891648

Supervised Learning- Back propagation Algo

- Similarly, we can calculate the other weight values as well.
- After that we will again propagate forward and calculate the output. Again, we will calculate the error.
- If the error is minimum we will stop right there, else we will again propagate backwards and update the weight values.
- This process will keep on repeating until error becomes minimum.

Conclusion:

Well, if I have to conclude Backpropagation, the best option is to write pseudo code for the same.

Supervised Learning- Back propagation Algo

Backpropagation Algorithm:

```
initialize network weights (often small random values)
  do
     forEach training example named ex
        prediction = neural-net-output(network, ex)  // forward pass
        actual = teacher-output(ex)
        compute error (prediction - actual) at the output units
        compute {displaystyle Delta w_{h}} for all weights from hidden layer to outpu
        compute {displaystyle Delta w_{i}} for all weights from input layer to hidden
        update network weights // input layer not modified by error estimate
  until all examples classified correctly or another stopping criterion satisfied
  return the network
```

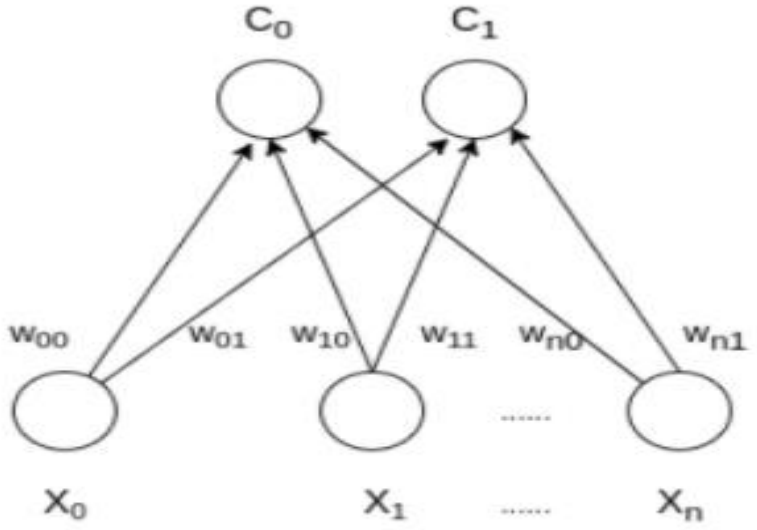
Un Supervised Learning

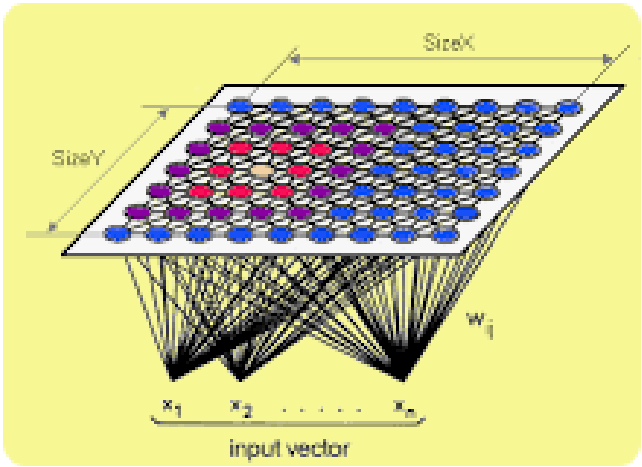
Un Supervised Learning- *Self Organizing Maps*

- **Self Organizing Map (or Kohonen Map or SOM)** is a type of Artificial Neural Network which is also inspired by biological models of neural systems form the 1970's.
- It follows an unsupervised learning approach and trained its network through a competitive learning algorithm.
- SOM is used for clustering and mapping (or dimensionality reduction) techniques to map multidimensional data onto lower-dimensional which allows people to reduce complex problems for easy interpretation.

Un Supervised Learning- *Self Organizing Maps*

- SOM has two layers, one is the Input layer and the other one is the Output layer.
- The architecture of the Self Organizing Map with two clusters and n input features of any sample is given below:





Un Supervised Learning- *SOM Working*

Input

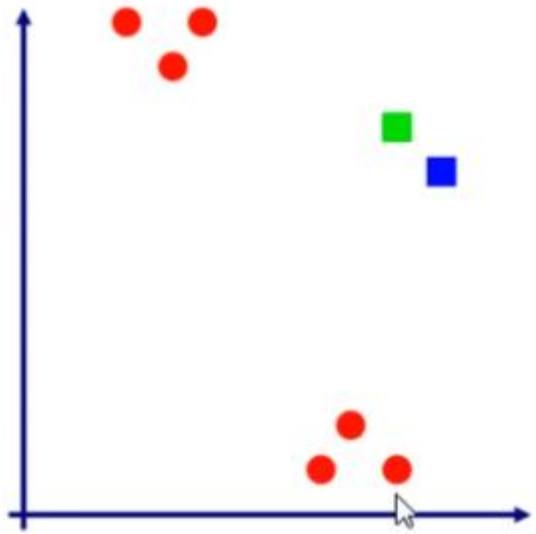
ID	Atributo 1	Atributo 2	Atributo 3	Atributo 4	Atributo 5	Atributo 6
1	148	22	49	56	27	172
2	94	67	45	74	70	31
3	146	66	110	58	101	58
4	91	54	73	49	59	90
5	183	4	48	178	59	183
6	61	133	10	116	134	48
7	118	135	57	153	99	102
8	62	86	121	163	63	175
9	181	71	182	156	158	58
10	123	174	179	58	169	170
11	77	178	64	90	126	6
12	149	5	59	73	22	199
13	68	41	112	156	1	86
14	171	29	52	20	193	189
15	91	81	140	93	78	132
16	199	77	75	159	112	136
17	69	3	47	86	2	93
18	186	46	100	168	126	152
19	197	103	171	192	70	140
20	22	143	20	36	128	140

Number of neurons
Maximum of epochs
Learning Rate

Output

ID	Atributo 1	Atributo 2	Atributo 3	Atributo 4	Atributo 5	Atributo 6	Classe
1	148	22	49	56	27	172	5
2	94	67	45	74	70	31	1
3	146	66	110	58	101	58	3
4	91	54	73	49	59	90	5
5	183	4	48	178	59	183	3
6	61	133	10	116	134	48	2
7	118	135	57	153	99	102	1
8	62	86	121	163	63	175	2
9	181	71	182	156	158	58	5
10	123	174	179	58	169	170	4
11	77	178	64	90	126	6	2
12	149	5	59	73	22	199	3
13	68	41	112	156	1	86	4
14	171	29	52	20	193	189	1
15	91	81	140	93	78	132	1
16	199	77	75	159	112	136	5
17	69	3	47	86	2	93	2
18	186	46	100	168	126	152	1
19	197	103	171	192	70	140	3
20	22	143	20	36	128	140	2

Un Supervised Learning- *SOM Working*



Operations

Select random input

Compute winner neuron

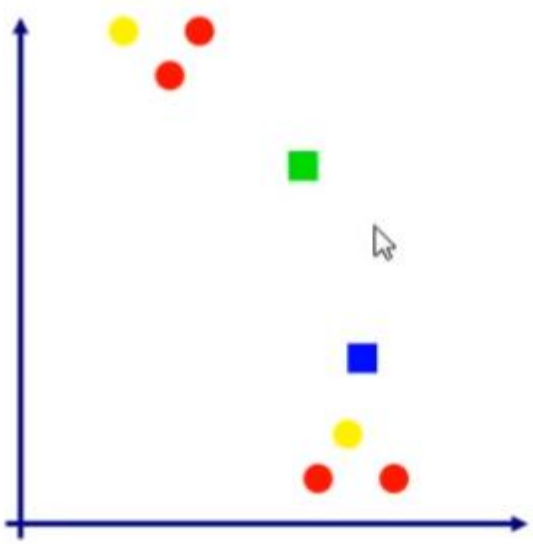
Update neurons

Repeat for all input data

Classify input data

□ 2 Neurons
○ 6 Inputs

Un Supervised Learning- *SOM Working*



Operations

Select random input

Compute winner neuron

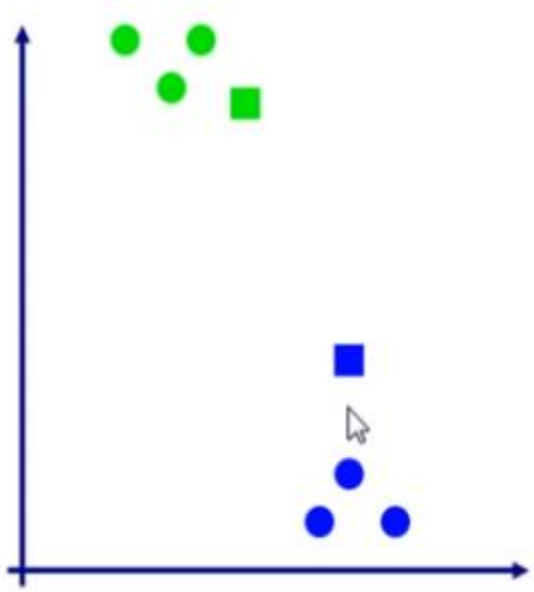
Update neurons

Repeat for all input data

Classify input data

□ 2 Neurons
○ 6 Inputs

Un Supervised Learning- *SOM Working*



Operations

Select random input

Compute winner neuron

Update neurons

Repeat for all input data

Classify input data

□ 2 Neurons
○ 6 Inputs

Un Supervised Learning- *Self Organizing Maps*

- Competitive learning network.
- Used for clustering data without knowing the class memberships of the input data.
- It can be used to detect features inherent to problem and hence called *Self Organizing Feature Map(SOFM).*
- Neurons usually form a two-dimensional lattice
- The property of topology means that the mapping preserves the relative distance between the points.
- It has generalization capability which implies network can recognize inputs it has never encountered before

Un Supervised Learning- *Self Organizing Maps*

- The weight vectors define each cluster.
- Input patterns are compared to each cluster, and associated with the cluster it best matches.
- The comparison is usually based on minimum Euclidean distance.
- When a best match is found, the associated cluster gets its weights and its neighboring units updated.
- Weight vectors are arranged into lines or various grid structures.

Un Supervised Learning- *Self Organizing Maps algorithm*

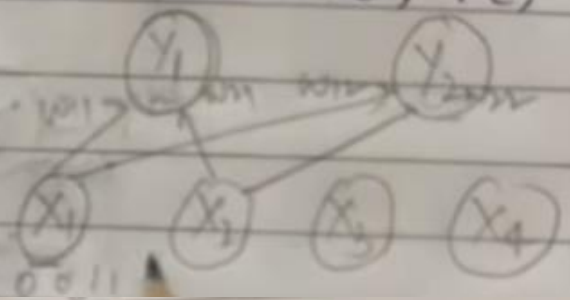
Step 0	Initialize weights. Set max value for R. Set learning rate
Step 1	While stopping condition false do steps 2 to 8
Step 2	For each input vector x do steps 3 to 5
Step 3	For each j neuron, compute the Euclidean distance $D_j = \sum_i (w_{ij} - x_i)^2$
Step 4	Find the index J such that D(J) is a minimum
Step 5	For all neurons j within a specified neighbourhood of J and for all i .$w_{ij}(new) = w_{ij}(old) + \alpha\left(x_i - w_{ij}(old)\right)$
Step 6	Update the learning rate α. It is a decreasing function of number of epochs
Step 7	Reduce radius of the topological neighbourhood at specified times
Step 8	Test the stopping condition. Typically this is a small value of learning rate with which the weight updates are insignificant.

Un Supervised Learning- *Self Organizing Maps*

Step 0: Initialize the weights w_{ij}. Random values may be assumed. Initialize the learning rate α.

Step 1: Calculate square of the Euclidean distance, i.e., for each $j = 1$ to m,

$$D(j) = \sum_{i} \sum_{j=1}^{n} (x_i - w_{ij})^2$$

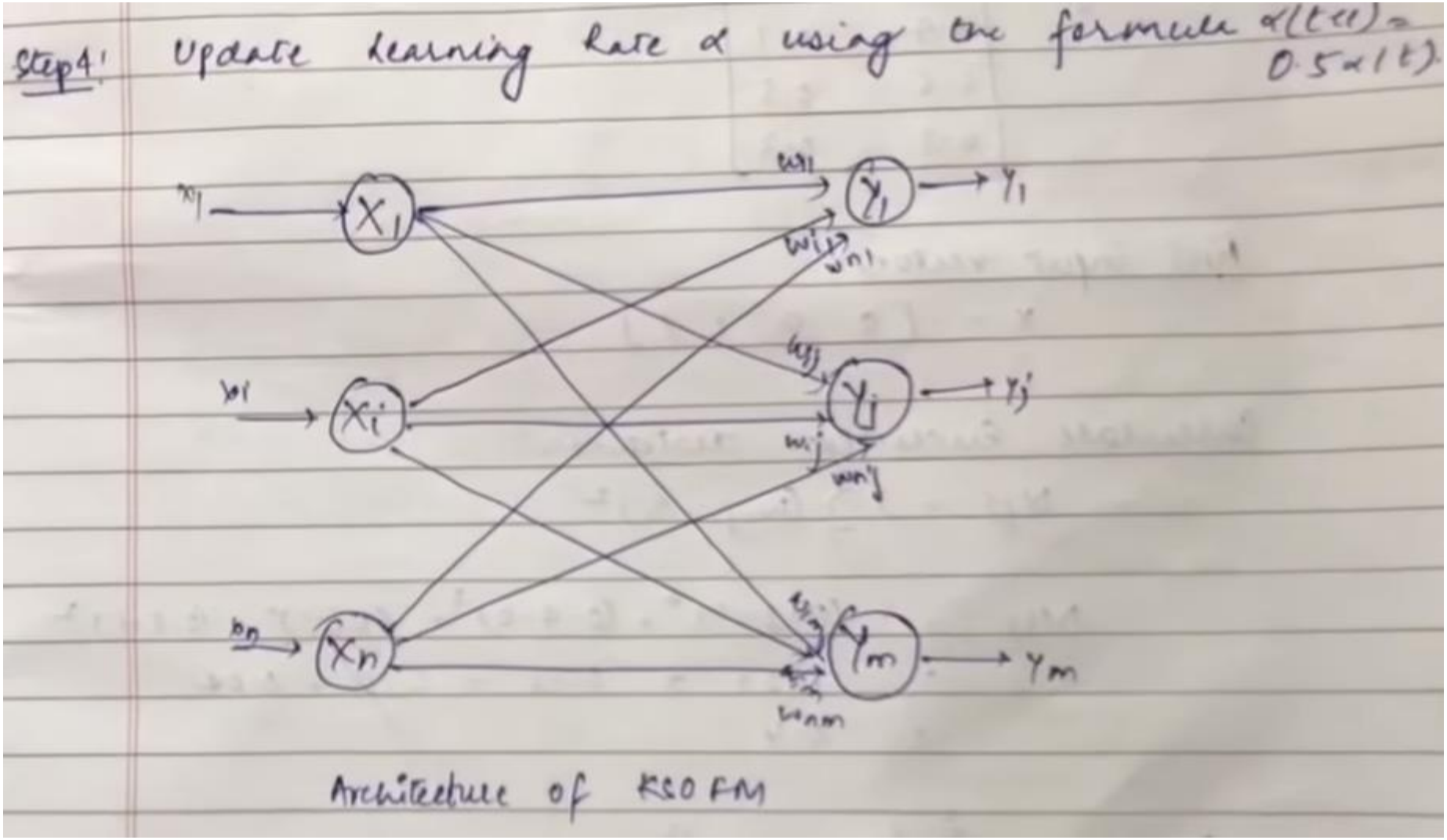


Step 2: Find winning unit index J, so that $D(j)$ is minimum.

Step 3: For all units j within a specific neighbourhood of J and for all i, calculate new weights:

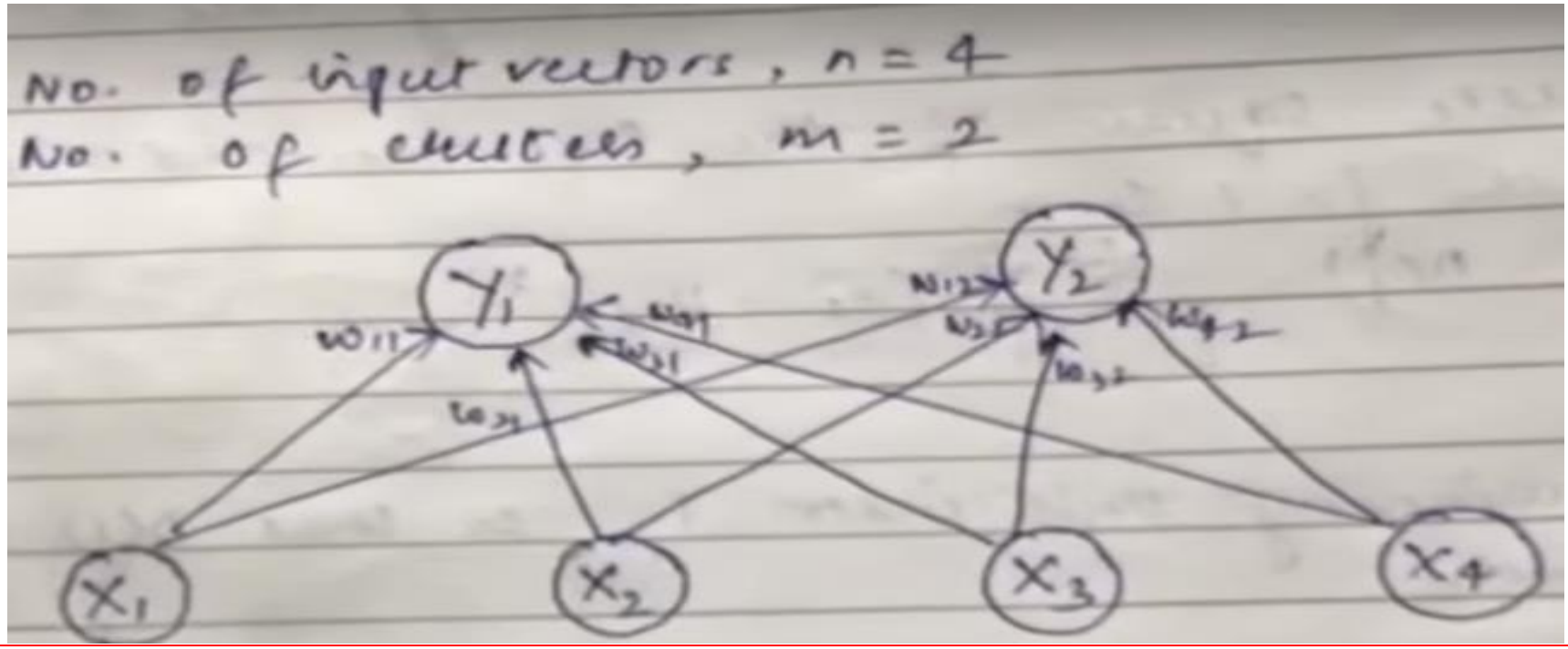
$$w_{ij}(new) = w_{ij}(old) + \alpha[x_i - w_{ij}(old)]$$

Un Supervised Learning- *Self Organizing Maps*



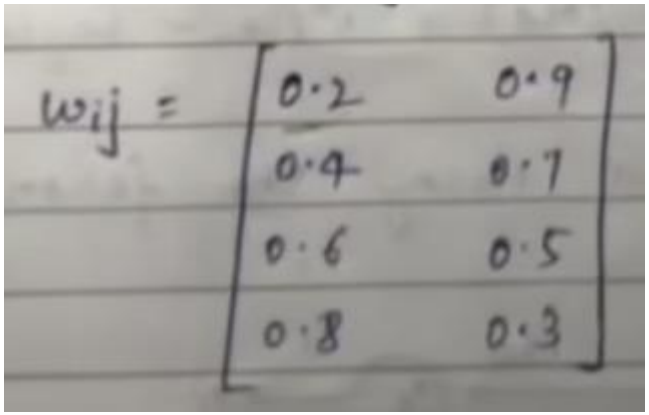
Un Supervised Learning- *Self Organizing Maps*

Construct KSOFM to cluster four given vectors [0 0 1 1], [1 0 0 0], [0 1 1 0] and [0 0 0 1]. No. of clusters to be formed is 2. Assume an initial learning rate of 0.5.



Un Supervised Learning- *Self Organizing Maps*

Step 0: Initialize weights randomly between 0 & 1



Un Supervised Learning- *Self Organizing Maps*

Step 1: Calculate Euclidean distance for input vector 1

First input vector,

$x = [0 \quad 0 \quad 1 \quad 1]$

Calculate Euclidean distance.

$$D(j) = \sum_{i} (w_{ij} - x_i)^2$$

$$D(1) = (0.2-0)^2 + (0.4-0)^2 + (0.6-1)^2 + (0.8-1)^2$$
$$= 0.04 + 0.16 + 0.16 + 0.04$$
$$= 0.4$$

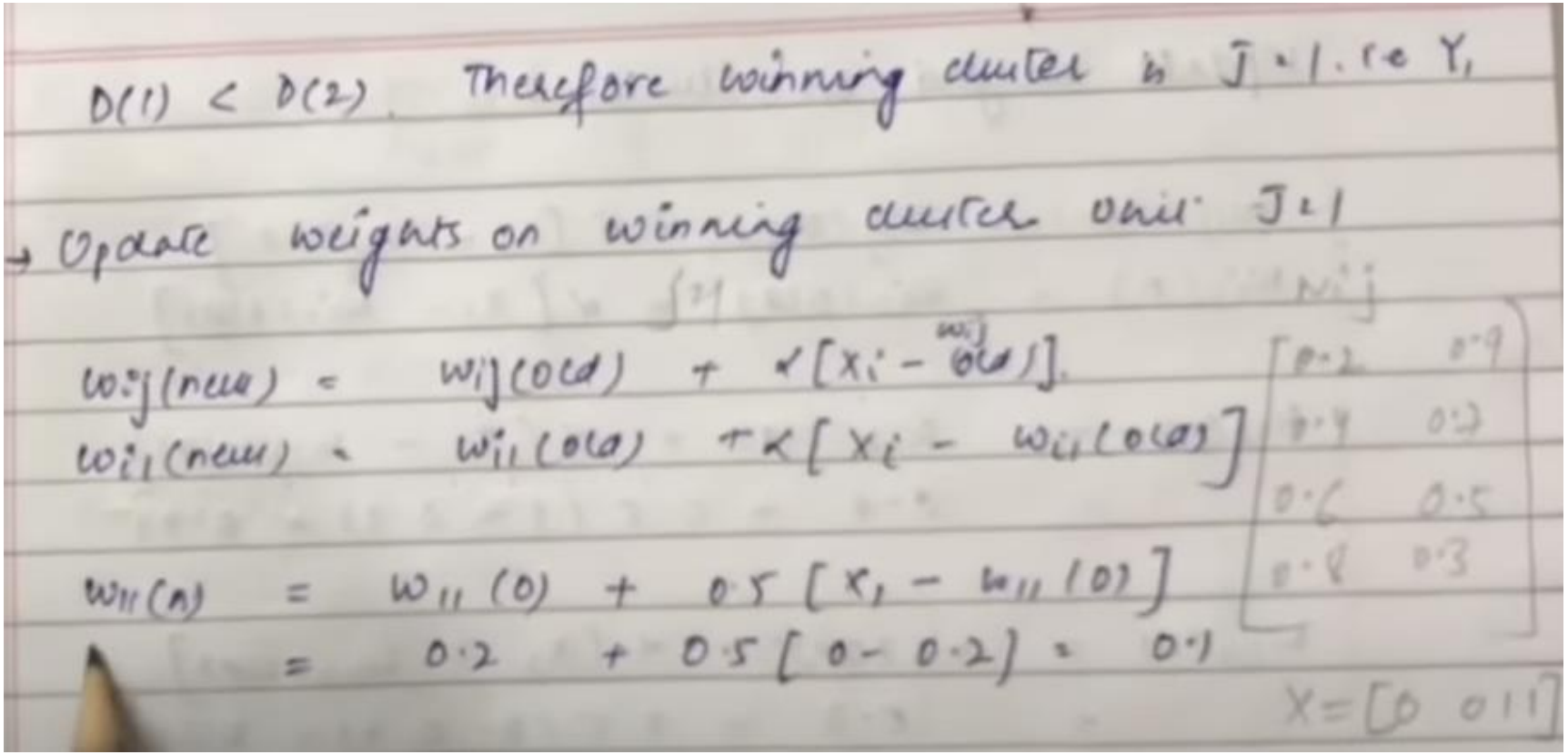
$$D(2) = \sum_{i=1}^{4} (w_{i2} - x_i)^2$$

$$= (0.9-0)^2 + (0.7-0)^2 + (0.5-1)^2 + (0.3-1)^2$$
$$= 0.81 + 0.49 + 0.25 + 0.49 = 2.04$$

Un Supervised Learning- *Self Organizing Maps*

Step 2: Find winning unit index J

Step 3: Calculate the new weights



Un Supervised Learning- *Self Organizing Maps*

Step 3: Calculate the new weights

$w_{21}(n) = w_{21}(old) + 0.5\,[x_2 - w_{21}(old)]$

$= 0.4 + 0.5\,(0 - 0.4) = 0.2$

$w_{31}(n) = 0.6 + 0.5\,(1 - 0.6) = 0.8$

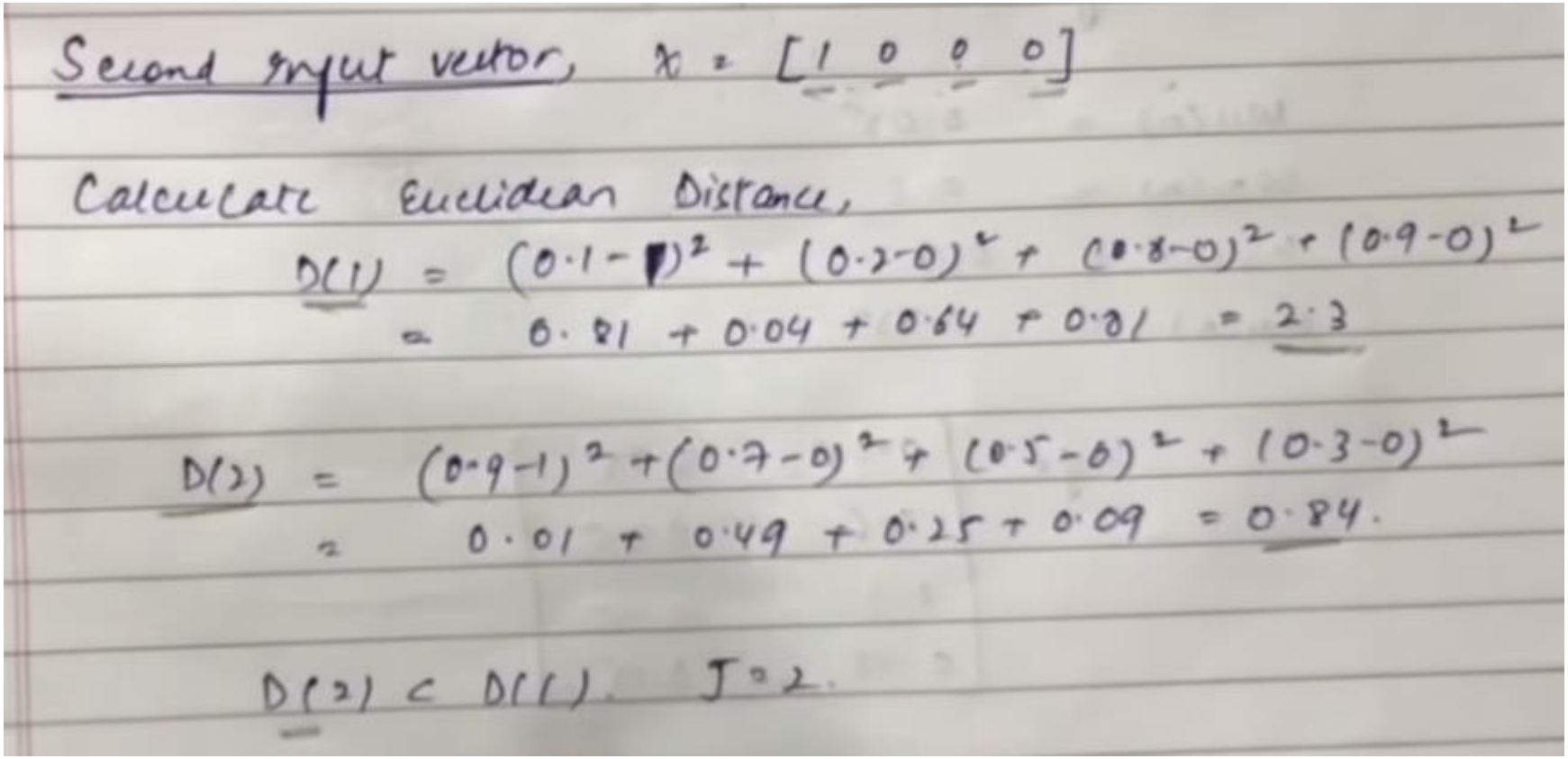
$w_{41}(n) = 0.8 + 0.5\,(1 - 0.8) = 0.9$

Updated weight matrix,

$$w_{ij} = \begin{bmatrix} 0.1 & 0.9 \\ 0.2 & 0.7 \\ 0.8 & 0.5 \\ 0.9 & 0.3 \end{bmatrix}$$

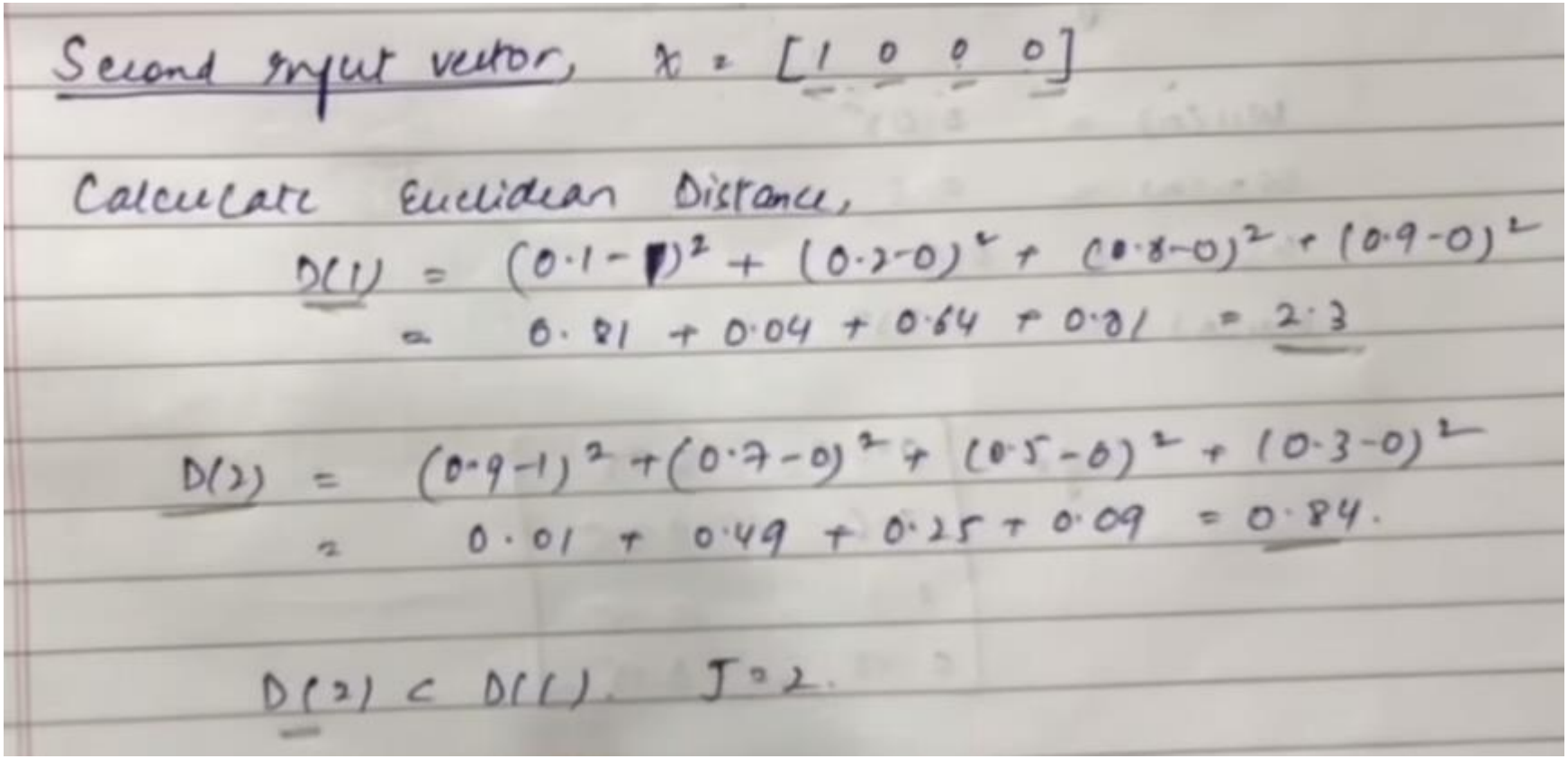
Un Supervised Learning- *Self Organizing Maps*

Step 1 for 2nd input vector: Calculate Euclidean distance for input vector 2



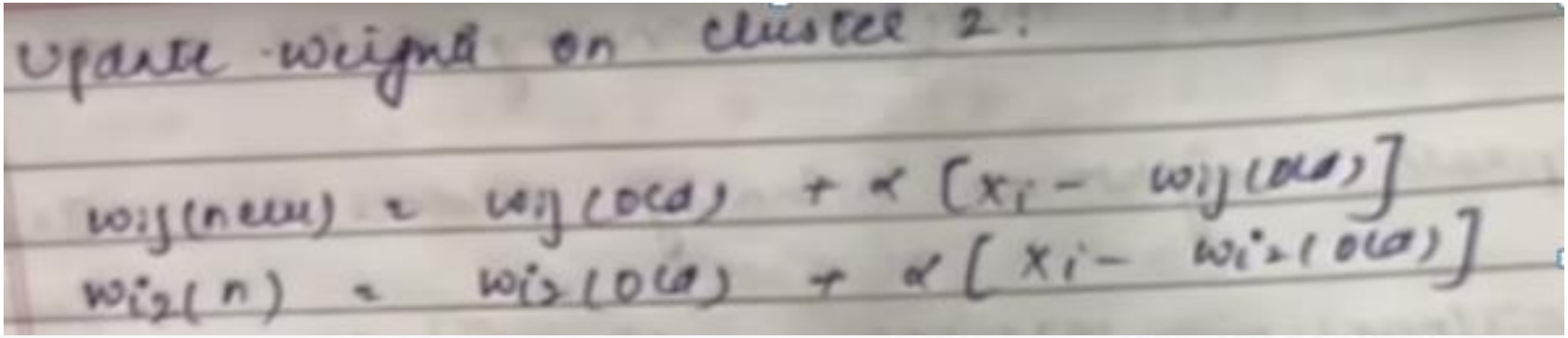
Un Supervised Learning- *Self Organizing Maps*

Step 2 for 2nd input vector: Calculate Euclidean distance for input vector 2



Un Supervised Learning- *Self Organizing Maps*

Step 3 for 2nd input vector: Calculate new weights



$w_{12}(n) = w_{12}(old) + \alpha[x_1 - w_{12}(old)]$
$= 0.9 + 0.5(1 - 0.9) = 0.95$

$w_{22}(n) = w_{22}(old) + \alpha[x_2 - w_{22}(old)]$
$= 0.7 + 0.5(0 - 0.7) = 0.35$

$w_{32}(n) = 0.5 + 0.5(0 - 0.5) = 0.25$

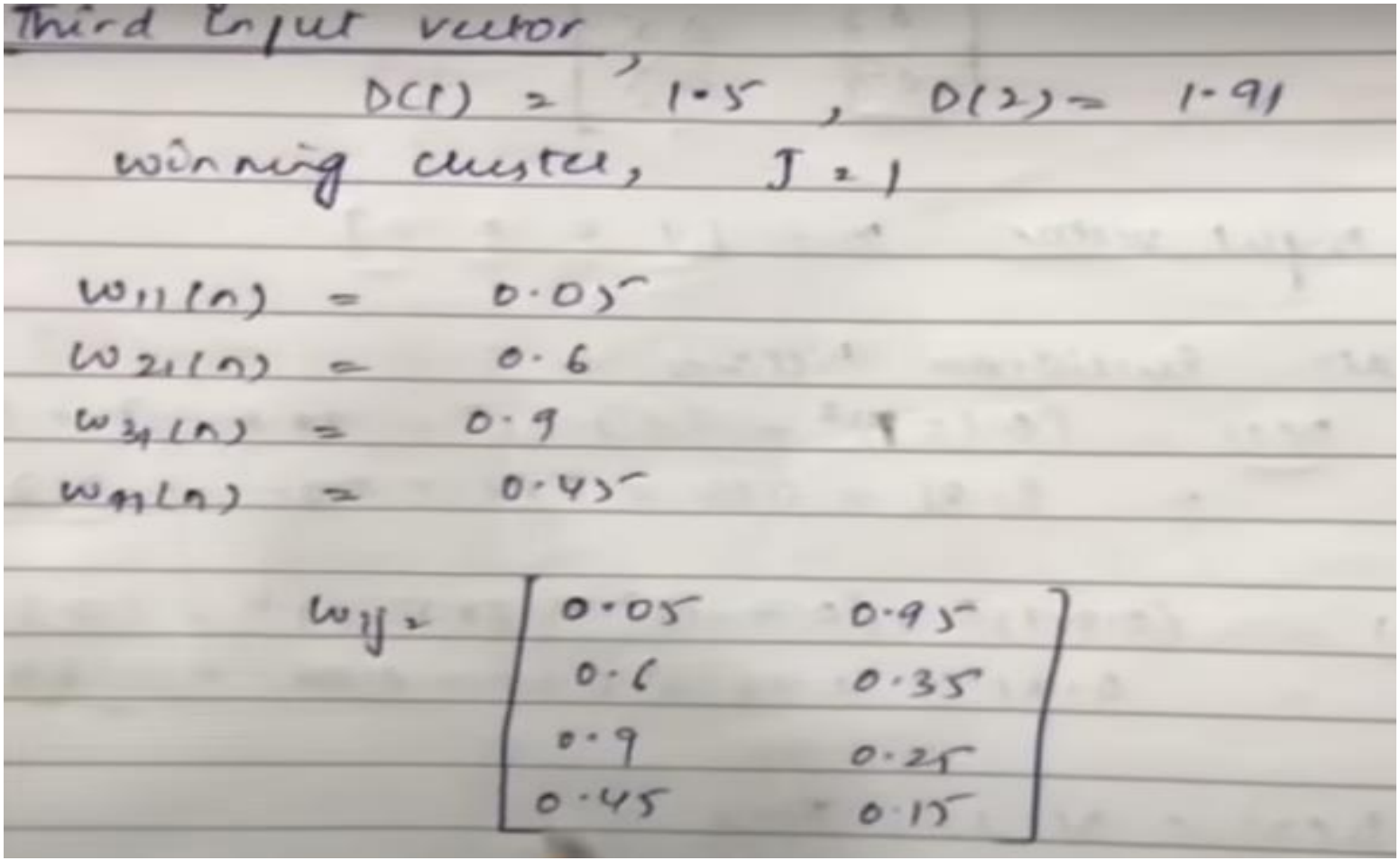
$w_{42}(n) = 0.3 + 0.5(0 - 0.3) = 0.15$

$$w_{ij} = \begin{bmatrix} 0.1 & 0.95 \\ 0.2 & 0.35 \\ 0.8 & 0.25 \\ 0.9 & 0.15 \end{bmatrix}$$

Un Supervised Learning- *Self Organizing Maps*

Step 2 for 3rd input vector: Calculate Euclidean distance for input vector 3

Step 3: Calculate the new weights



Un Supervised Learning- *Self Organizing Maps*

Step 2 for 4th input vector: Calculate Euclidean distance for input vector 4

Step 3: Calculate the new weights

Fourth input vector, [0 0 0 1]

D(1) = 1.475 D(2) = 1.81

Winning cluster, J = 1.

$w_{11}(n) = 0.025$

$w_{21}(n) = 0.3$

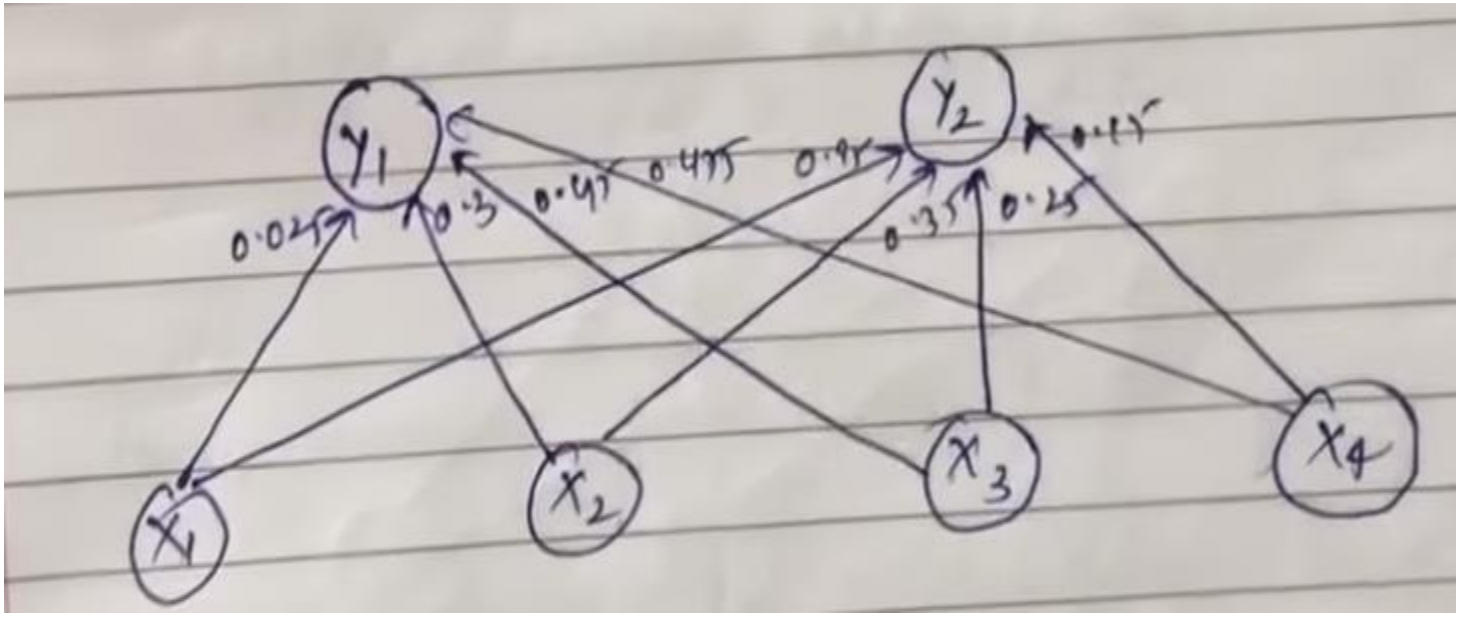
$w_{31}(n) = 0.45$

$w_{41}(n) = 0.475$

$$w_{ij} = \begin{bmatrix} 0.025 & 0.95 \\ 0.3 & 0.35 \\ 0.45 & 0.25 \\ 0.475 & 0.15 \end{bmatrix}$$

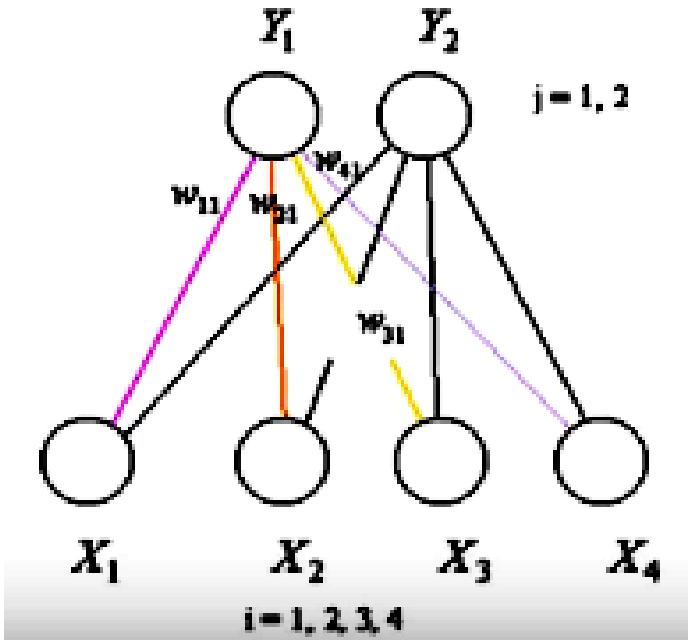
Un Supervised Learning- *Self Organizing Maps*

Step 4:



Un Supervised Learning- *Self Organizing Maps Numerical*

Consider there are two output neurons and the connections are as shown. Consider the simple case where there are 4 input training patterns. Assume that learning rate is given as $\alpha(t+1) = \frac{1}{2}\alpha(t)$ and $\alpha(0) = 0.6$ and R=0.Show how the weight updating takes place in this SOM Network



Let the initial weight matrix be

$$\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \\ w_{41} & w_{42} \end{bmatrix} = \begin{bmatrix} 0.2 & 0.8 \\ 0.6 & 0.4 \\ 0.5 & 0.7 \\ 0.9 & 0.3 \end{bmatrix}$$

x_1	x_2	x_3	x_4
1	1	0	0
0	0	0	1
1	0	0	0
0	0	1	1

Introduction- *Self Organizing Maps Numerical*

- Construct kohonen Self-organizing map to cluster the four given vectors, [0 0 11], [1 0 0 0], [0 11 0] and [0 0 0 1]. The number of cluster formed is two. Assume an initial learning rate of 0.5. **Dec 2019 10 marks**

Thank You