

20MCA283

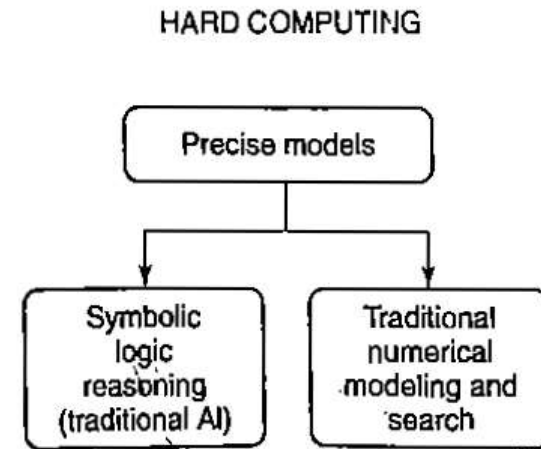
DEEP LEARNING

Introduction

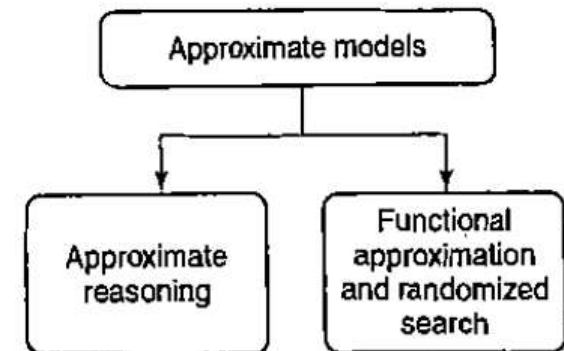
- Two major Problem Solving Techniques are
 - Hard computing
 - Soft Computing

Introduction

- Hard computing
- Hard computing deals with precise models where accurate solutions are achieved quickly.
- Hard Computing technique require exact input data
- It is strictly sequential and provides precise answers to complex problems.



- Soft computing
- Soft Computing deals with approximate models and gives solutions to complex problems and deals with imprecise, uncertain and partial truth data.
- Soft computing is a combination of Neural Network, Fuzzy Logic and Genetic Algorithm.

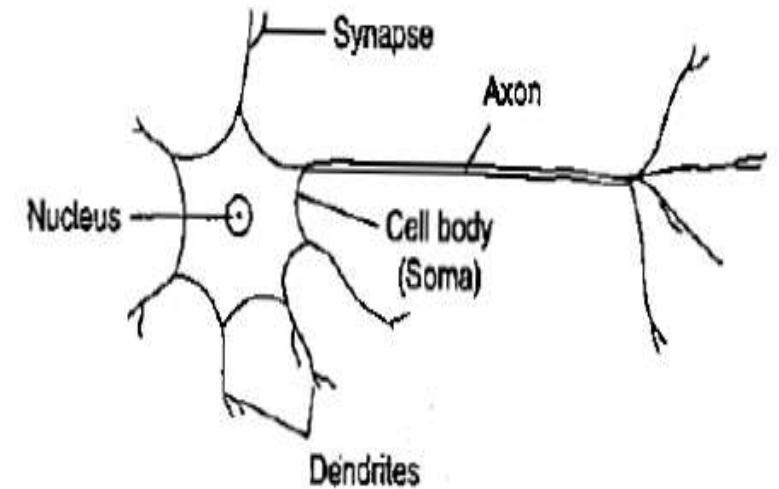


Biological Neuron

- Human Brain consist of billions of neural cells called neurons (approximately 10^{11})
- Neuron process information.
- Each cell works like single processor.
- Interaction between cells and their parallel processing helps brain to learn, re-organize itself from experience and adapt to the environment.
- Information transported between neurons in the form of electric signals.

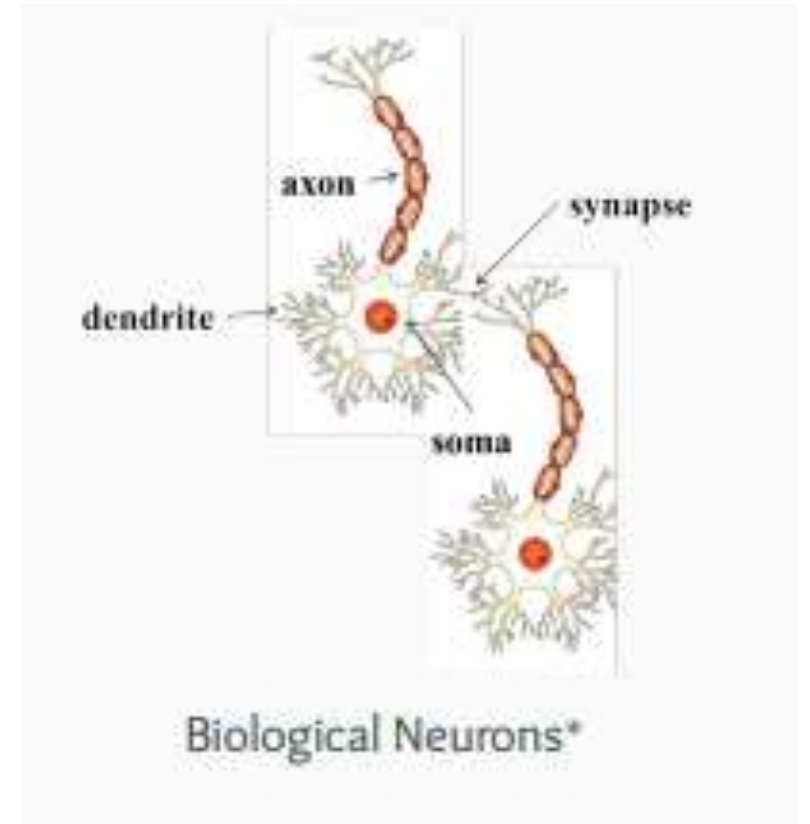
Neuron consists of the following four parts

- **Dendrites**—responsible for receiving the information from other neurons it is connected to.
- **Soma (Cell body)** — It is responsible for processing of information they have received from dendrites. Nucleus is located here.
- **Axon** — It is just like a cable through which neurons send the information.
- **Synapses** — It is the connection between the axon and other neuron dendrites.



Schematic diagram of a biological neuron.

- The incoming information from dendrites are added up at nucleus and then delivered to synapse via axon.
- If the incoming stimulation has exceeded a certain threshold, the neuron is activated.
- If the stimulation is too low the neuron is inhibited, and the information will not be transported to any further.

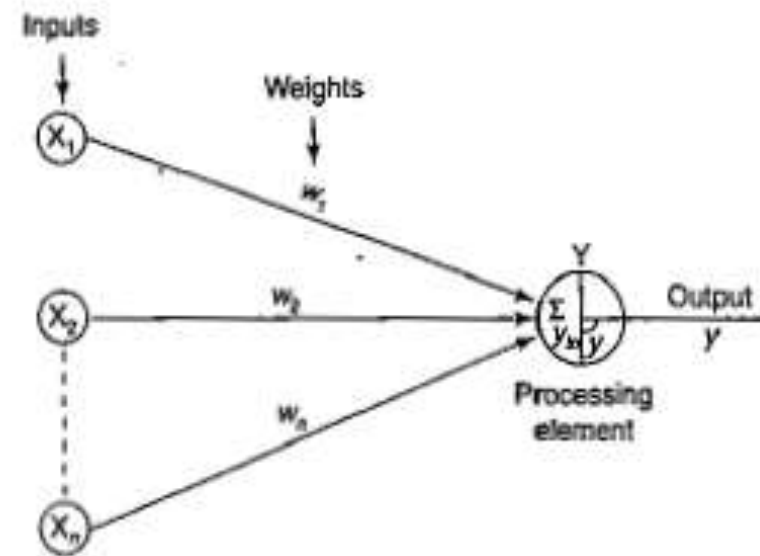


Artificial Neuron

- Artificial Neuron is a mathematical model of the biological neuron.
- Artificial Neuron is the basic unit of Artificial Neural Network(ANN).
- A neuron can accept any number of inputs and send a single output signal.

Artificial Neuron – Mathematical Model

- Let the inputs be x_1, x_2, \dots, x_n
- Inputs are connected to the cell body through links having weights w_1, w_2, \dots, w_n respectively
- Weights represents the connection strength similar to synaptic strength in biological neuron.



Artificial Neuron – Net Input Calculation

- Net input

$$Y_{in} = x_1w_1 + x_2w_2 + \dots + x_nw_n$$
$$= \sum_{i=1}^n x_iw_i$$

- Output Calculation

To calculate output, an activation function is applied over the net input Y_{in}

Eg: for Activation functions

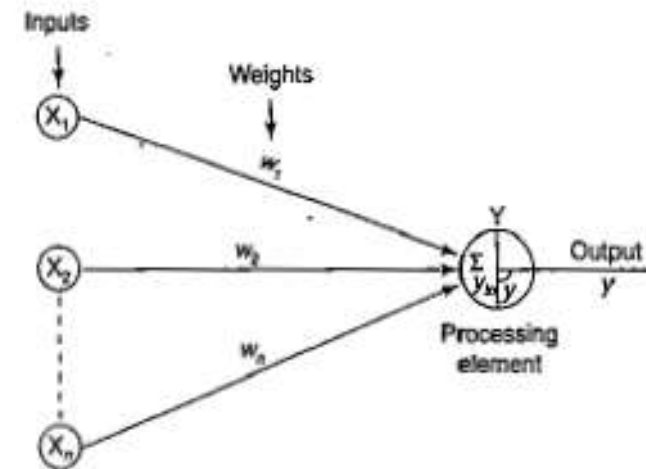
Hard Limiter

Sign

Sigmoid

Ramp Function

Hyperbolic Tangent



Biological Neuron Vs. Artificial Neuron

Terminology relationship between biological and artificial neuron

Biological Neuron	Artificial Neuron
Cell	Neuron
Dendrites	Weight or Interconnection
Soma	Net Input
Axon	Output

Biological Neuron Vs. Artificial Neuron

	Biological Neuron	Artificial Neuron
1. Speed of execution	Milliseconds	Nano seconds
2. Processing Data	Can perform massive parallel operations	Can perform massive parallel operations
3. Size and Complexity	Total number of neurons in brain is about 10^{11} and total number of interconnections is about 10^{15} Complexity is High	Size and complexity is based on application and designer.
4. Tolerance	Fault tolerant Can store and retrieve data even if the interconnections got disconnected	Information got corrupted if interconnections got disconnected.

Biological Neuron Vs. Artificial Neuron

	Biological Neuron	Artificial Neuron
5. Storage or memory	Store data on synapse and new data can be added by adjusting synaptic strength without destroying old data.	Store data at weight matrix. Adding new data may destroy old data.
6. Control Mechanism	No control unit.	A control unit present in CPU, which transfer values from one unit to another

Eg: ANN Models

- McCulloch Pits Model (1943)
- Hebb Network (1949)
- Perceptron (1958)
- Adaline (1960)
- Back Propagation Network(1986)

Characteristics of ANN

- ✓Neurally implemented mathematical model.
- ✓Contains interconnected processing elements(neurons).
- ✓WEIGHTED LINKS (interconnections) hold information.
- ✓Neurons can learn, recall, generalize data by adjusting weights.
- ✓No single neuron carries specific information; only a collection of neurons hold data.

ANN

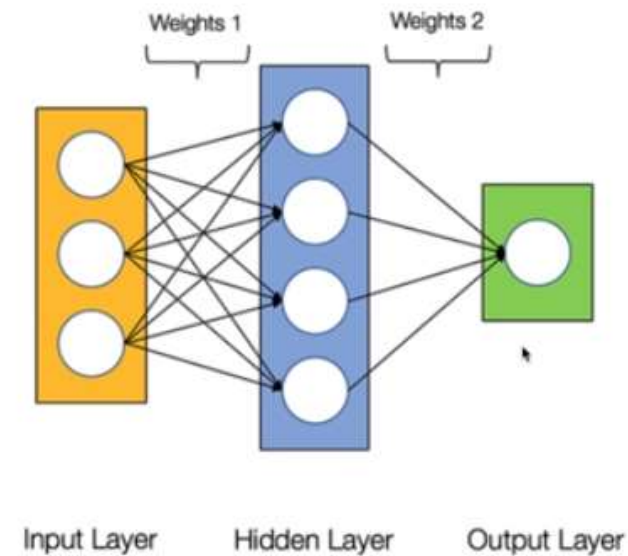
- In Neural networks neurons are organized in layers.

Input layer

Hidden Layer

Output layer

- When some data is fed to the ANN, it is processed via layers of neurons to produce desired output.
- Data is presented to the network via the input layer.
- Input layer communicates to hidden layers
- The hidden layers then link to an output layer where the answer is output



Basic Models of Artificial Neural Networks (ANN)

- Ann Models are specified by three entities
 1. Inter Connections
 2. Learning
 3. Activation Function

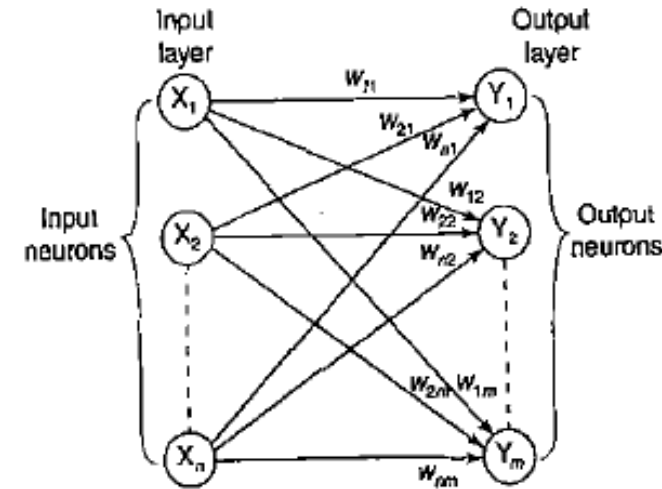
Basic Models of ANN

1. Inter Connections

- Connection pattern formed within and between layers is called the network architecture.
- There exist five basic types of neuron connection architectures.
 1. single-layer feed-forward network;
 2. multilayer feed-forward network;
 3. single node with its own feedback;
 4. single-layer recurrent network;
 5. multilayer recurrent network.

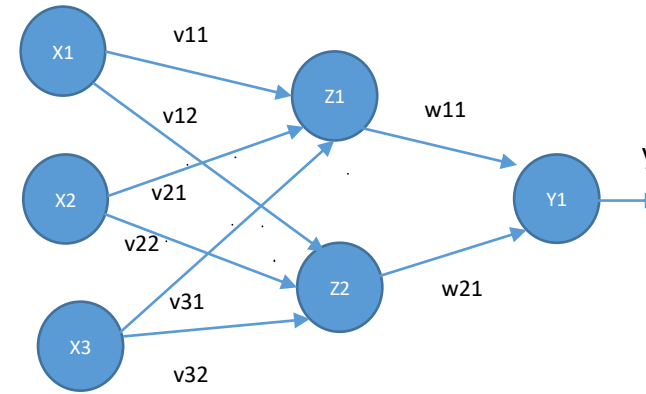
1.1 Single-layer feed-forward network

- In feed forward networks the data flow in a single direction, from the input layer data to the output layer.
- Single layer feed forward network contain input layer and output layer
- Here inputs are given directly to the neurons of output layer.
- Each neurons of output layer will calculate $\text{NET INPUT}(y_{in})$ and ACTIVATION FUNCTION is applied over it to produce the output(y)



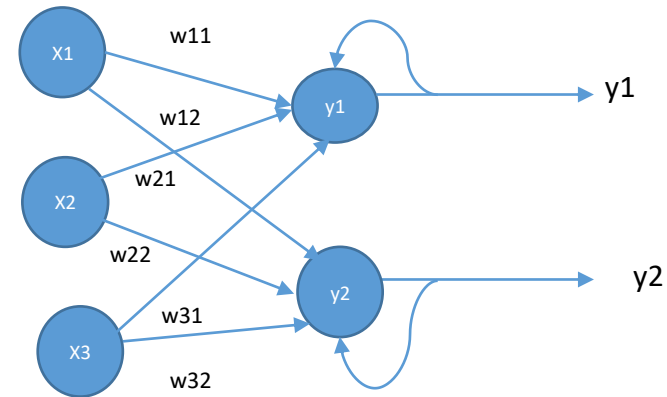
1.2 Multi layer Feed Forward network

- This type network contain one or more layers (hidden layers) between input and output layer.
- More the number of the hidden layers, more is the complexity of the network.



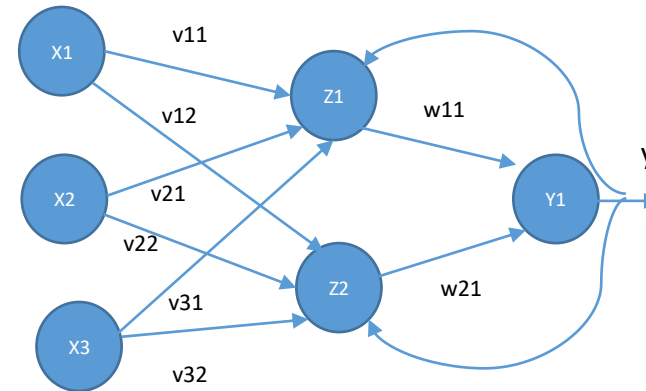
1.3 Single layer Recurrent Network (Feed back Network)

- Networks contain output links directed back as inputs to the same or preceding layer nodes is called feedback networks.
- Recurrent networks are feedback networks with closed loop.



1.4 Multi layer Recurrent Network (Feed back Network)

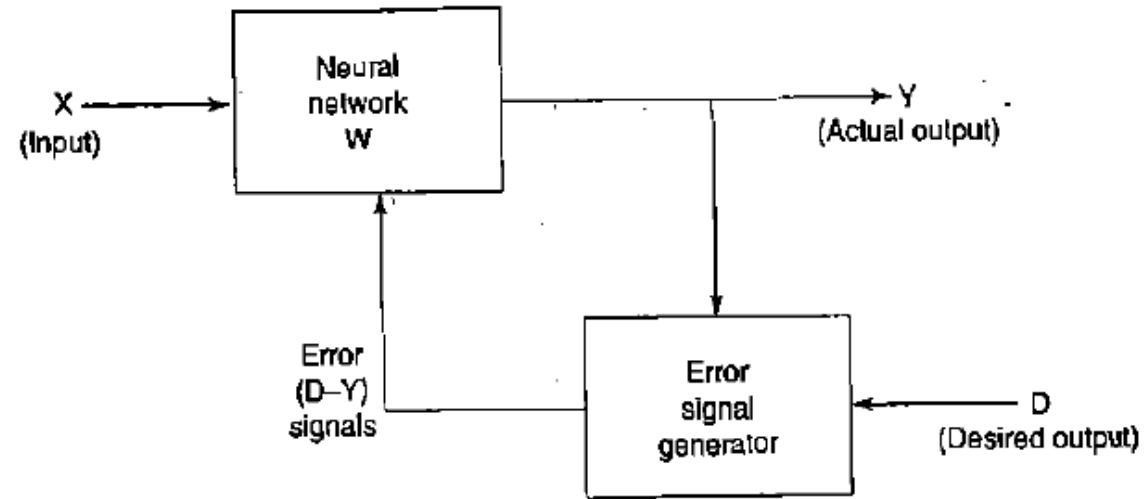
- Networks contain output links directed back as inputs to the same or preceding layer nodes is called feedback networks.
- Recurrent networks are feedback networks with closed loop.



2. Learning or Training

- Learning is the process which improves ANN's performance and is applied repeatedly over the network.
- Learning is done by updating weights till the desired output is obtained.
- For learning there is a learning algorithm.
- Data called as training data set is fed to the learning algorithm and the algorithm draws inferences from the training data set.
- Types of learning are:
 - Supervised learning
 - Unsupervised learning
 - Reinforcement learning

2. Learning or Training – Supervised learning

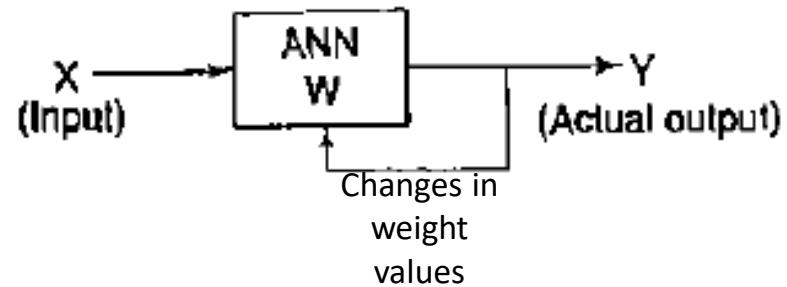


- Learn with the help of a Teacher/ Supervisor.
- In this method both input and output patterns (training pair) are provided.
- During training, input is given to ANN, which gives an output(actual output).
- This actual output is compared with the target output(output given in the training pair).
- If there exists a difference an error signal is generated.
- This error signal is used for adjusting weights.
- This process repeats until actual output matches the target output or we attain some accuracy.
- Supervisor helps to reduce error, so it is called supervised learning.

2. Learning or Training – Unsupervised learning

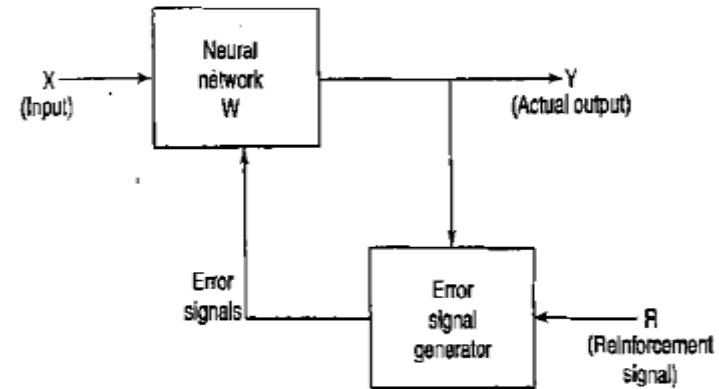
- Learning process is independent
- Inputs are grouped as clusters without the use of training data, it finds the hidden structure in the input.
- In the training process, the network receives the input patterns and organize these patterns to form clusters.
- When an input is applied to ANN, it gives a response indicating the cluster to which that input belongs.
- If an input didn't belongs to a cluster, a new cluster is formed.
- There is no mechanism to check whether the outputs are correct or not.

- ANN must itself discover pattern regularities and features from input and the relations for the input data over the output.
- While discovering these features network undergo changes in weights. This process is called self –organizing.



2. Learning or Training – Reinforcement Learning

- This is a form of supervised learning.
- The exact information if output is not available but a critic information is available.
- Feedback is sent as reinforcement signal.
- This reinforcement signal is processed in an error signal generator.
- ANN adjusts weights according to the error signal and the training process repeats.



What is an Activation Function?

Activation functions are an extremely important feature of the artificial neural networks. They basically decide whether a neuron should be activated or not. Whether the information that the neuron is receiving is relevant for the given information or should it be ignored.

$$Y = \text{Activation}(\Sigma(\text{weight} * \text{input}) + \text{bias})$$

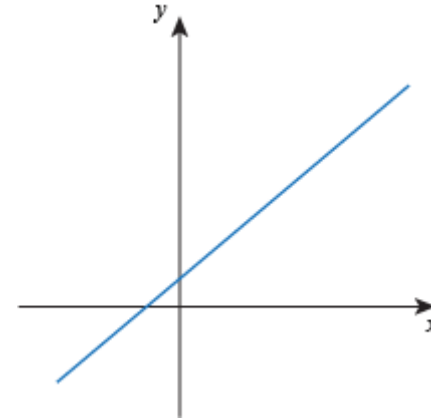
The activation function is the non linear transformation that we do over the input signal. This transformed output is then seen to the next layer of neurons as input.

- Linear **Activation Function**
- Non Linear **Activation Function**

What is an Activation Function?

Linear Function

The function is a line or linear. Therefore, the output of the functions will not be confined between any range



Non Linear Function

They make it easy for the model to generalize or adapt with variety of data and to differentiate between the output

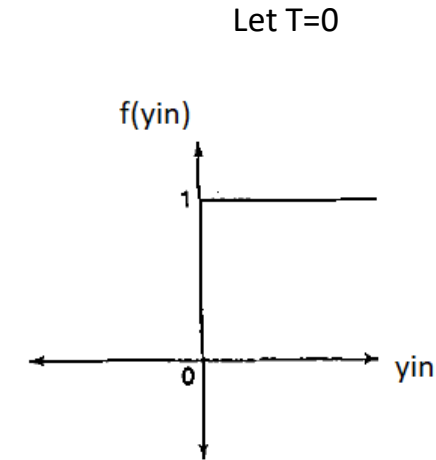
The **Nonlinear Activation** Functions are mainly divided on the basis of their **range or curves**

1. Threshold
2. Sigmoid
3. Tanh
4. ReLU
5. Leaky ReLU
6. Softmax

Various Activation Functions

1. Hard Limiter or STEP function or Binary Step function

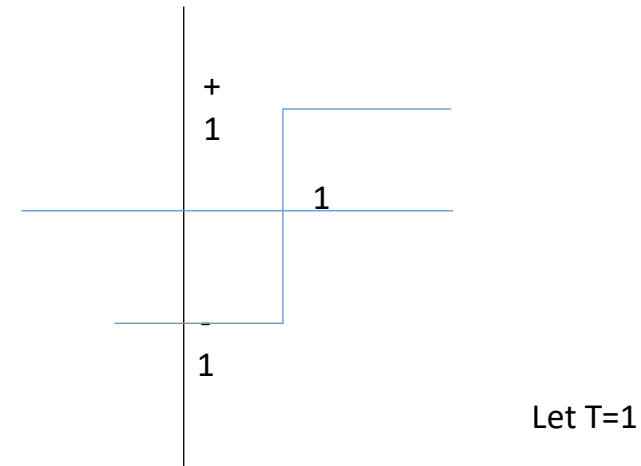
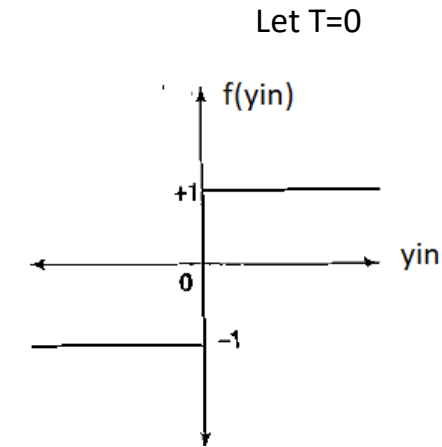
$$\begin{aligned} f(y_{in}) &= 0 \text{ if } y_{in} < \text{Threshold}(T) \\ &= 1 \text{ if } y_{in} \geq \text{Threshold}(T) \end{aligned}$$



2. Sign function or Bipolar Step

$$\begin{aligned} f(y_{in}) &= -1 \text{ if } y_{in} < \text{Threshold}(T) \\ &= 1 \text{ if } y_{in} \geq \text{Threshold}(T) \end{aligned}$$

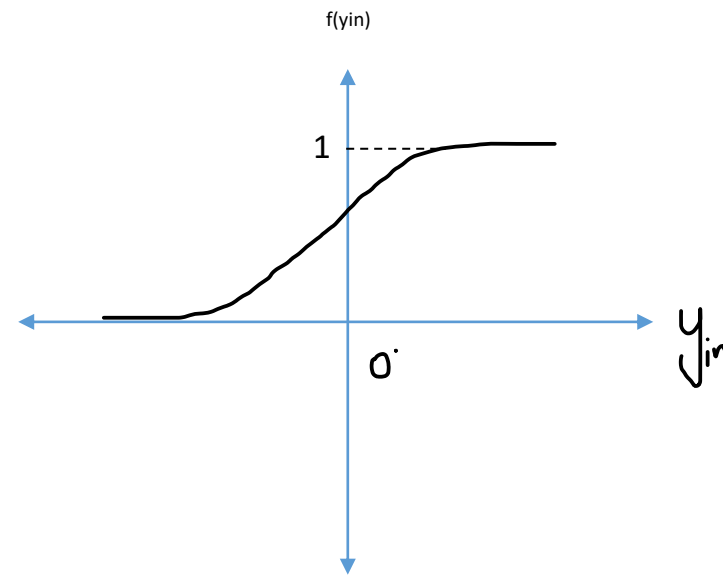
Here y is -1 or +1



3. Binary Sigmoid

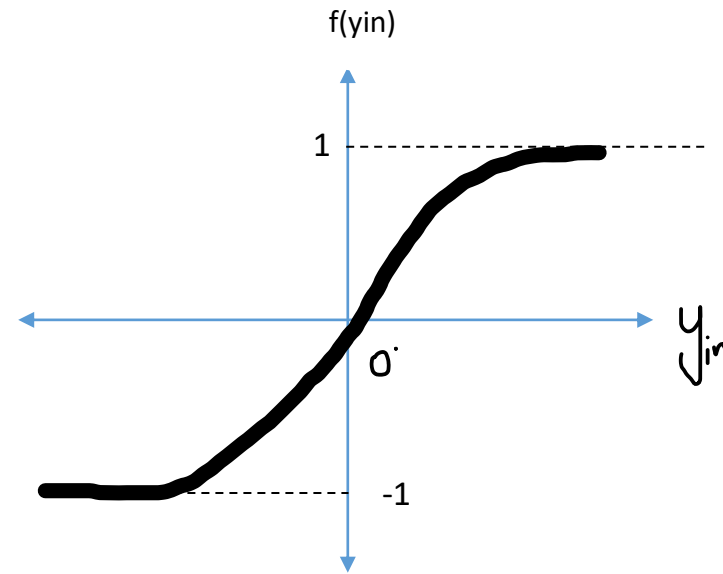
$$f(y_{in}) = \frac{1}{1 + e^{-y_{in}}}$$

Here y lies between 0 and 1



4. Bipolar Sigmoid

$$f(y_{in}) = \frac{1 - e^{-y_{in}}}{1 + e^{-y_{in}}}$$

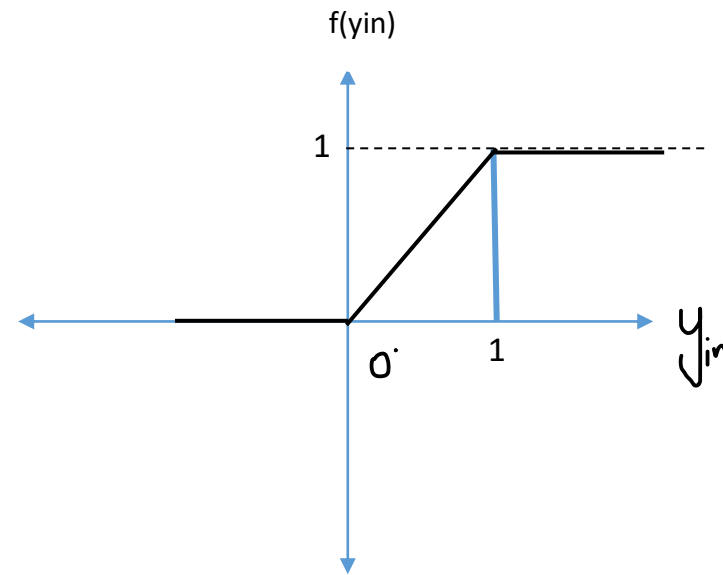


Here y lies between -1 and 1

5. RAMP function

$$\begin{aligned} f(y_{in}) &= 1 && \text{if } y_{in} > 1 \\ &= 0 && \text{if } y_{in} < 0 \\ &= y_{in} && \text{if } 0 \leq y_{in} \leq 1 \end{aligned}$$

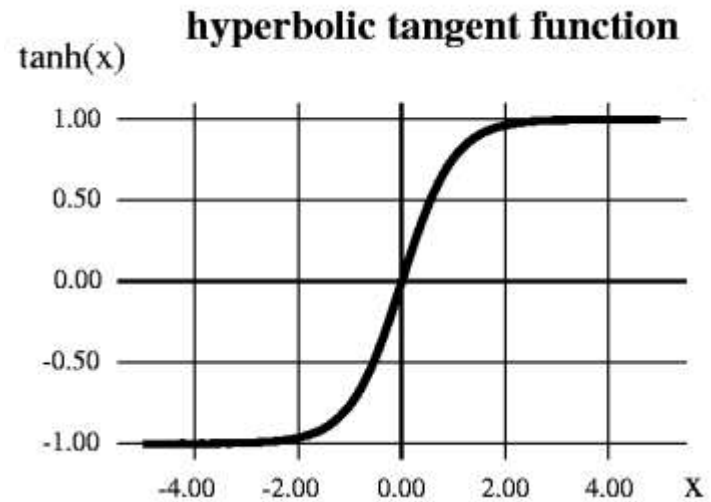
Here y lies between 0 and 1



6. Tanh—Hyperbolic tangent

mathamatical formula is

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



ReLu (**ReLU - Rectified linear units**)

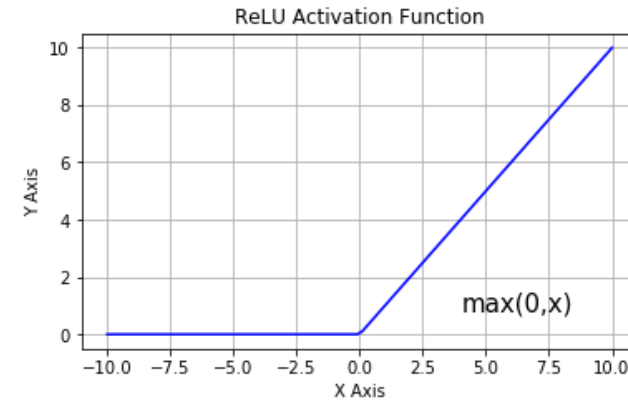
- ReLU is the most widely used activation function while designing networks today. First things first, the ReLU function is non linear, which means we can easily backpropagate the errors and have multiple layers of neurons being activated by the ReLU function.

Almost all deep learning Models use **ReLu**

Mathamatical formula is :

$$R(x) = \max(0, x)$$

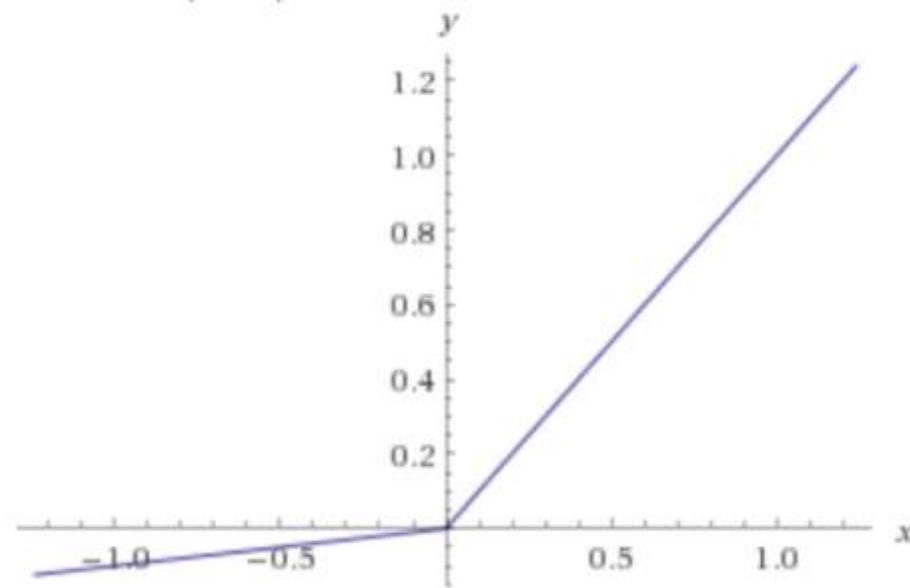
i.e if $x < 0$, $R(x) = 0$ and if $x \geq 0$, $R(x) = x$.



- *It is very simple and efficient .*
- **ReLu is only be used within Hidden layers of a Neural Network Model.**

Leaky Relu Function?

Leaky ReLU function is nothing but an improved version of the ReLU function. As we saw that for the ReLU function, the gradient is 0 for $x < 0$, which made the neurons die for activations in that region. Leaky ReLU is defined to address this problem. Instead of defining the ReLU function as 0 for x less than 0, we define it as a small linear component of x .



Here the horizontal line is replaced with a non-zero, non-horizontal line. Here a is a small value like 0.01 or so.

8. Softmax activation function

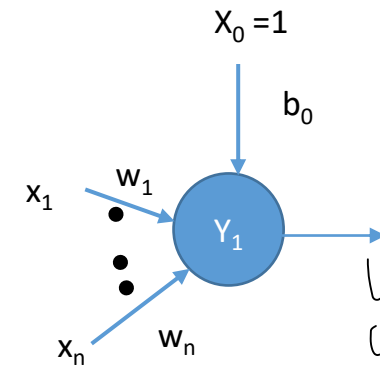
$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}} \quad \text{for } j = 1, \dots, K.$$

- The softmax function would squeeze the outputs for each class between 0 and 1 and would also divide by the sum of the outputs.
- This gives the probability of the input being in a particular class.

Terminologies of ANN

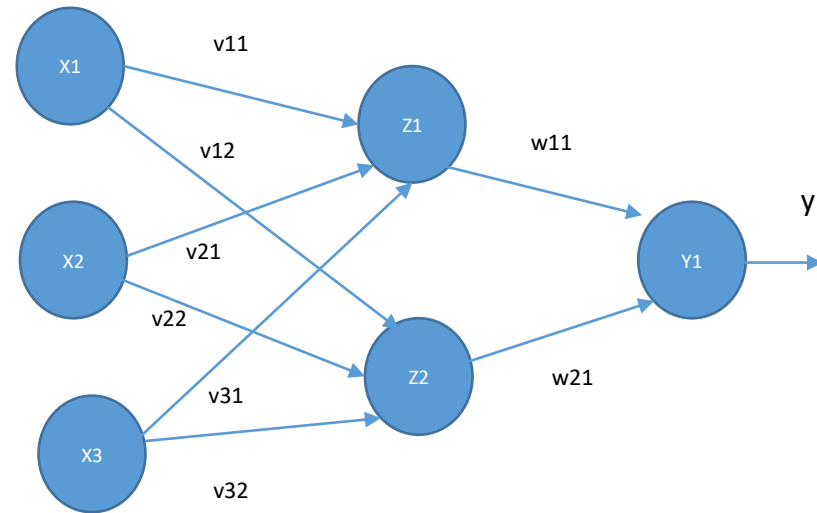
- **Weight** : Each neuron is connected to another neuron via links (weights). The weights contain information about input signal.
- **Bias** : Bias is an additional input($x_0=1$) with some weight (b_j).
Bias has an impact in calculating net input(y_{in})
If bias is positive net input increases
If bias is negative net input decreases

$$\text{Net input } y_{in} = x_0 b_j + \sum_{i=1}^n x_i w_i$$



$$V = \begin{bmatrix} V1 \\ V2 \\ V3 \end{bmatrix} = \begin{bmatrix} v11 & v12 \\ v21 & v22 \\ v31 & v32 \end{bmatrix}$$

$$W = \begin{bmatrix} w11 \\ w21 \end{bmatrix}$$



Terminologies of ANN

- **Threshold** : Threshold is a set value based upon which the final output of the network may be calculated.
The threshold value is used in activation function.
A comparison is made between the net input and the threshold to obtain the actual output.

$$f(y_{in}) = 0 \quad \text{if } y_{in} < \theta \\ = 1 \quad \text{if } y_{in} \geq \theta$$

- **Learning Rate** : It is used to control the amount of weight adjustment at each step in the training.
The learning rate α , ranging from 0 -to 1.
Determines the rate of learning at each time step

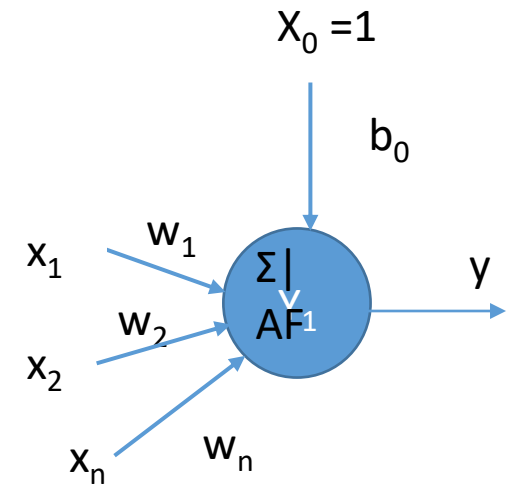
Perceptron Network

- Inputs are directly connected to the neurons in the output layer via adjustable weight values.
- The activation function used at the output layer is modified SIGN function.

output $y = 1$ if $y_{in} > +Threshold$

0 if $-Threshold \leq y_{in} \leq +Threshold$

-1 if $y_{in} < -Threshold$



Training Perceptron Network

- Learning is the process of updating the weight values.
- Perceptron can be trained by ***perceptron Learning rule***.
- Updation of weight is done by calculating the ERROR between the desired output (Target) and the calculated output (actual).

$$\text{ERROR} = \text{Target} - \text{Actual}$$

- If ERROR is zero goal has been achieved; otherwise update weight values.
- The perceptron networks are used to classify input pattern as a 'member' or 'not a member' to a particular class.

Perceptron Training Algorithm

Step1: initialise weights, learning parameter α and threshold θ .

Step2: for each training pair, activate inputs.

Step3: calculate output of the network.

1. obtain net input (y_{in})

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

2. Apply activation function over net input to get output (y)

$$\begin{aligned} y &= 1 && \text{if } y_{in} > +\theta \\ &0 && \text{if } -\theta \leq y_{in} \leq +\theta \\ &-1 && \text{if } y_{in} < -\theta \end{aligned}$$

Perceptron Training Algorithm

Step4: Compare calculated output, y and the target output, t .

Update weights if necessary.

if $y \neq t$ ERROR exists, then update weights as follows

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

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

else No need to update weight

$$w_i(\text{new}) = w_i(\text{old})$$

$$b(\text{new}) = b(\text{old})$$

Step5: If there is no change in weights for all training pair, stop training.

Else go to step 2

Perceptron Testing Algorithm

Once the training process is complete, test the performance of perceptron network.

Step1 :Initialise weights as the final weights obtained during training.


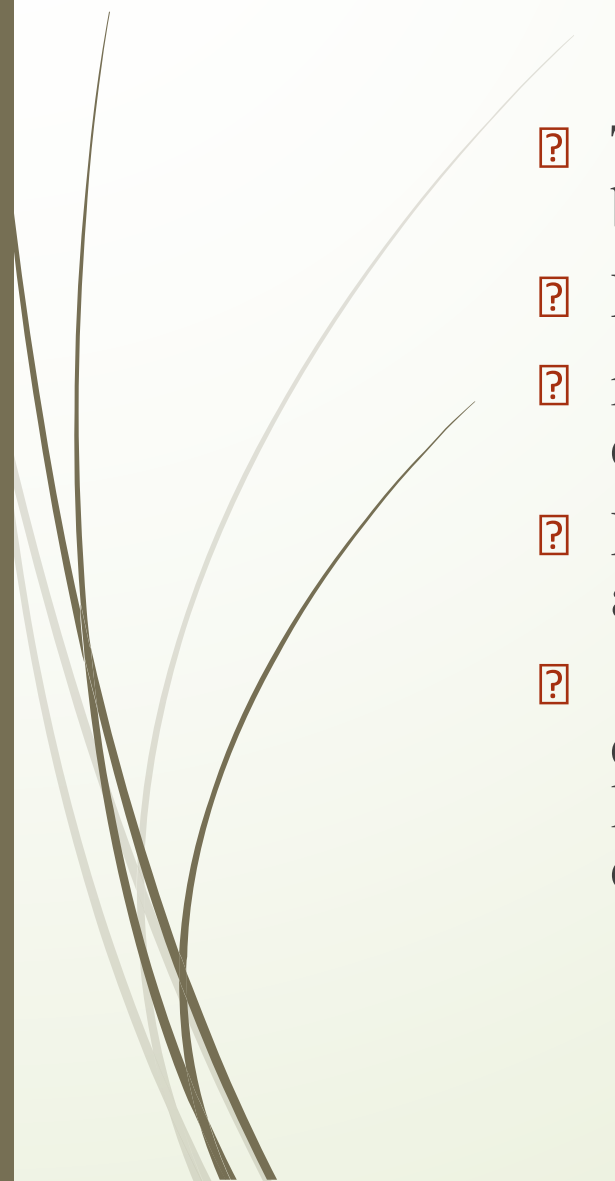
Step2 : For each input perform the following.

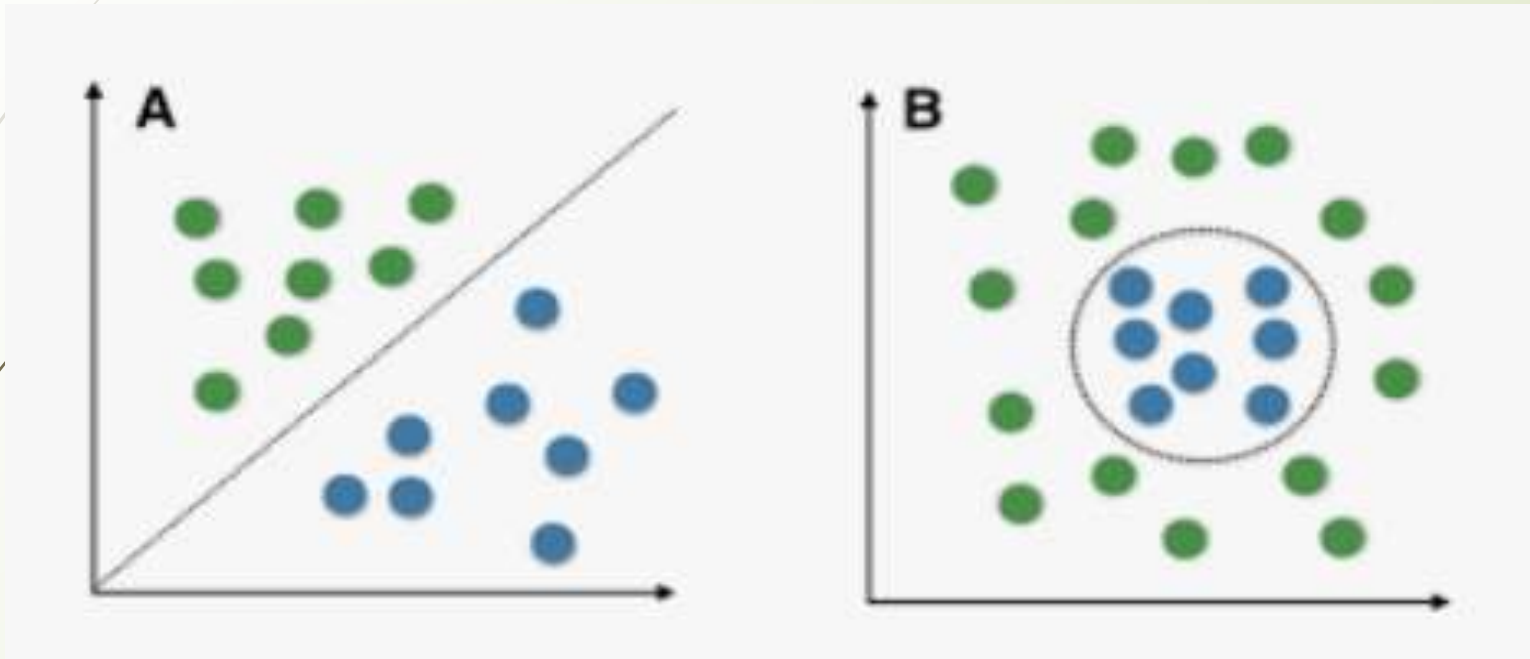
$$y_{in} = b + \sum_{i=1}^n x_i w_i$$

$$\begin{aligned} y &= 1 \quad \text{if } y_{in} > +\theta \quad (\text{Fire}) \\ &0 \quad \text{if } -\theta \leq y_{in} \leq +\theta \quad (\text{Not fire}) \\ &-1 \quad \text{if } y_{in} < -\theta \quad (\text{Not fire}) \end{aligned}$$

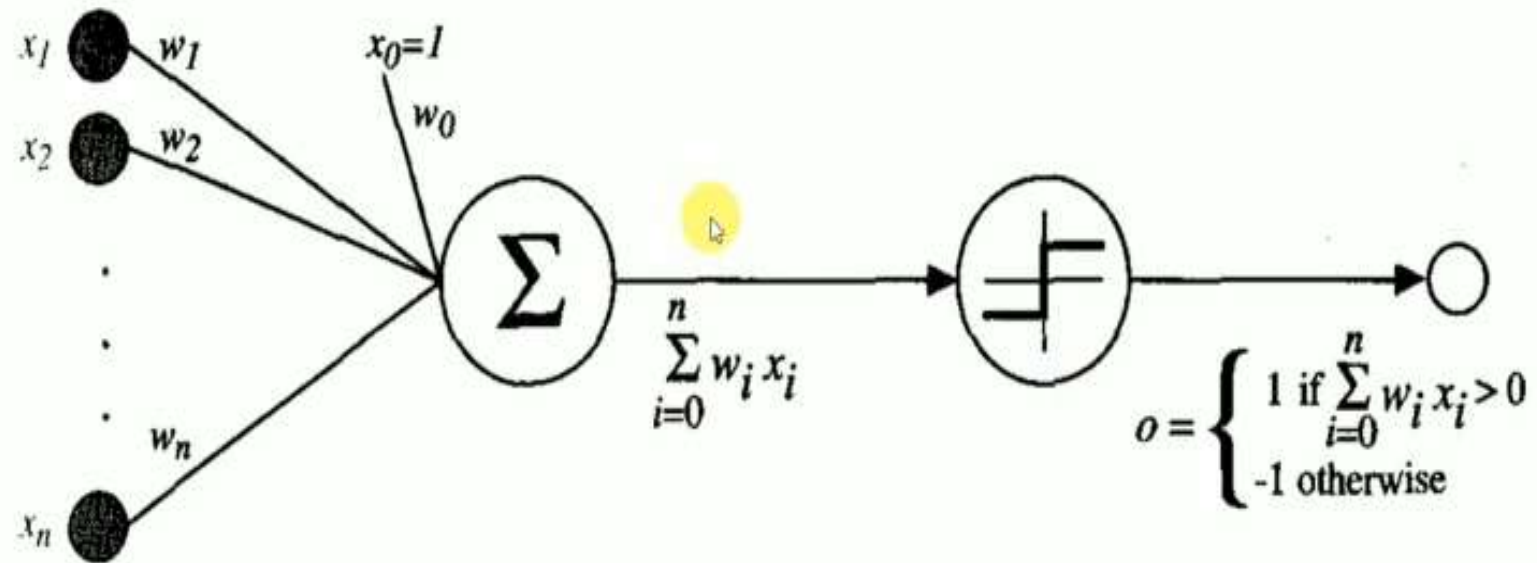


Perceptron


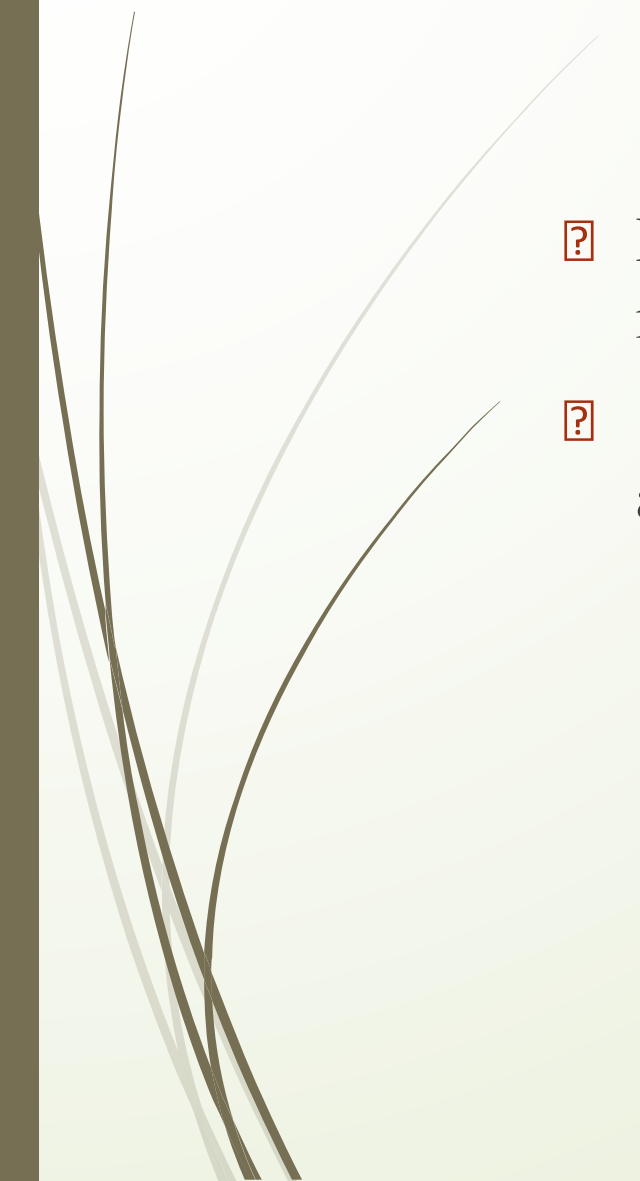
- 
- 
- ❑ The **Perceptron** is a supervised machine learning algorithm for binary classification tasks.
 - ❑ It is the simplest types of artificial neural networks.
 - ❑ it can quickly learn a linear separation in feature space for two-class classification tasks
 - ❑ It learns using the stochastic gradient descent optimization algorithm'
 - ❑ Perceptron is appropriate for those kind of problems where the classes can be separated well by a line or linear model, referred to as linearly separable. It learns a decision boundary that separates two classes using a line (called a hyperplane) in the feature space.

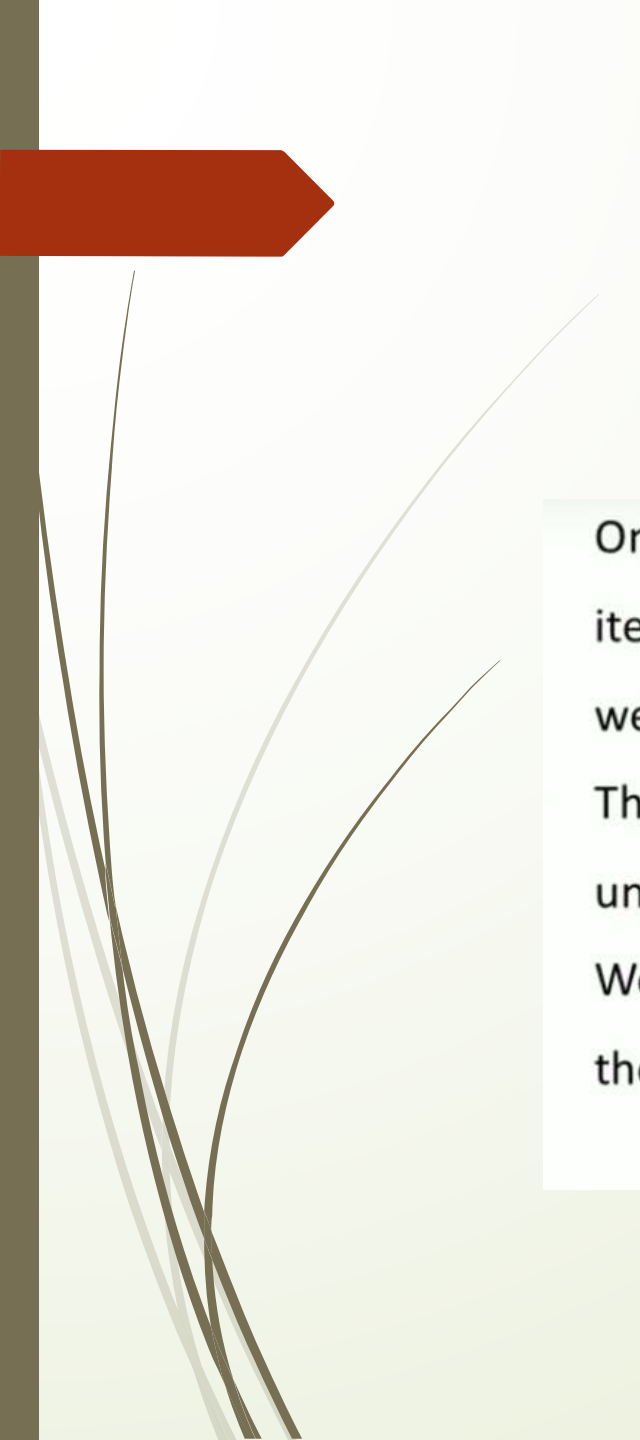


Linearly Separable



x_1, x_2, x_3, \dots are inputs
 w_1, w_2, w_3, \dots are associated weights
 w_0 is a constant

- 
- 
- ❑ It consists of a single node or neuron that takes a row of data as input and predicts a class label.
 - ❑ This is achieved by calculating the weighted sum of the inputs and a bias (set to 1).



One way to learn an acceptable weight vector is to begin with random weights, then iteratively apply the perceptron to each training example, modifying the perceptron weights whenever it misclassifies an example.

This process is repeated, iterating through the training examples as many times as needed until the perceptron classifies all training examples correctly.

Weights are modified at each step according to the **perceptron training rule**, which revises the weight **w_i** associated with input **x_i** according to the rule

$$w_i \leftarrow w_i + \Delta w_i$$

$$\Delta w_i = \eta(t - o)x_i$$

Perceptron Network for Logical **AND** operation

A	B	$A \wedge B$
0	0	0
0	1	0
1	0	0
1	1	1

$w_1 = 1.2$, $w_2 = 0.6$ Threshold = 1 and Learning Rate $n = 0.5$

$w_1 = 1.2, w_2 = 0.6$ Threshold = 1 and Learning Rate $\eta = 0.5$

A	B	$A \wedge B$
0	0	0
0	1	0
1	0	0
1	1	1

1. $A=0, B=0$ and Target = 0

- $w_i.x_i = 0*1.2 + 0*0.6 = 0$
- This is not greater than the threshold of 1, so the output = 0

2. $A=0, B=1$ and Target = 0

- $w_i.x_i = 0*1.2 + 1*0.6 = 0.6$
- This is not greater than the threshold of 1, so the output = 0

$w1 = 1.2, w2 = 0.6$ Threshold = 1 and Learning Rate $n = 0.5$

A	B	$A \wedge B$
0	0	0
0	1	0
1	0	0
1	1	1

3. $A=1, B=0$ and Target = 0

- $w_i.x_i = 1*1.2 + 0*0.6 = 1.2$
- This is greater than the threshold of 1, so the output = 1

$$w_i = w_i + n(t - o)x_i$$

$$w1 = 1.2 + 0.5(0 - 1)1 = 0.7$$

$$w2 = 0.6 + 0.5(0 - 1)0 = 0.6$$

$w1 = 0.7, w2 = 0.6$ Threshold = 1 and Learning Rate $n = 0.5$

A	B	$A \wedge B$
0	0	0
0	1	0
1	0	0
1	1	1

1. $A=0, B=0$ and Target = 0

- $w_i.x_i = 0*0.7 + 0*0.6 = 0$
- This is not greater than the threshold of 1, so the output = 0

2. $A=0, B=1$ and Target = 0

- $w_i.x_i = 0*0.7 + 1*0.6 = 0.6$
- This is not greater than the threshold of 1, so the output = 0

$w_1 = 0.7$, $w_2 = 0.6$ Threshold = 1 and Learning Rate $n = 0.5$

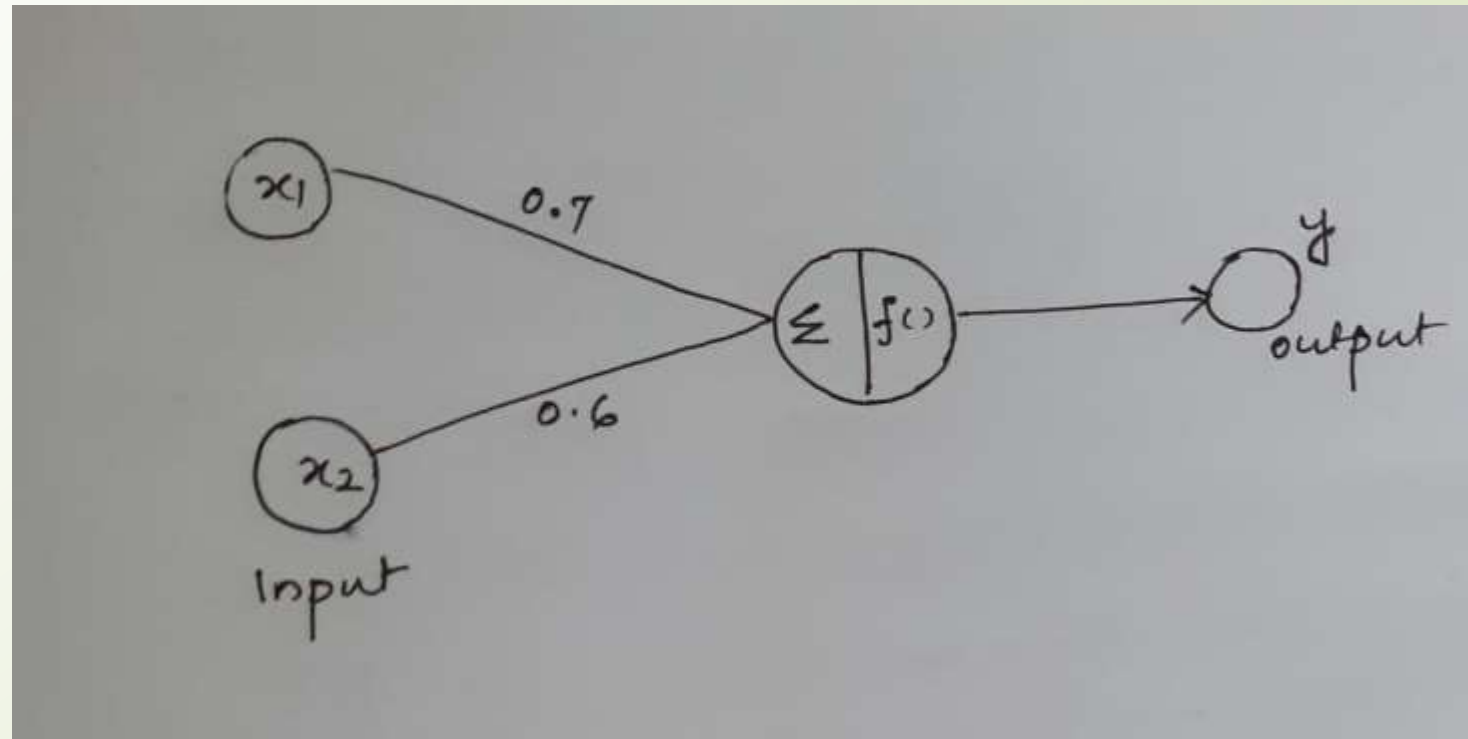
A	B	$A \wedge B$
0	0	0
0	1	0
1	0	0
1	1	1

3. $A=1$, $B=0$ and Target = 0

- $w_i.x_i = 1*0.7 + 0*0.6 = 0.7$
- This is not greater than the threshold of 1, so the output = 0

4. $A=1$, $B=1$ and Target = 1

- $w_i.x_i = 1*0.7 + 1*0.6 = 1.3$
- This is greater than the threshold of 1, so the output = 1



Design a perceptron NN for OR gate


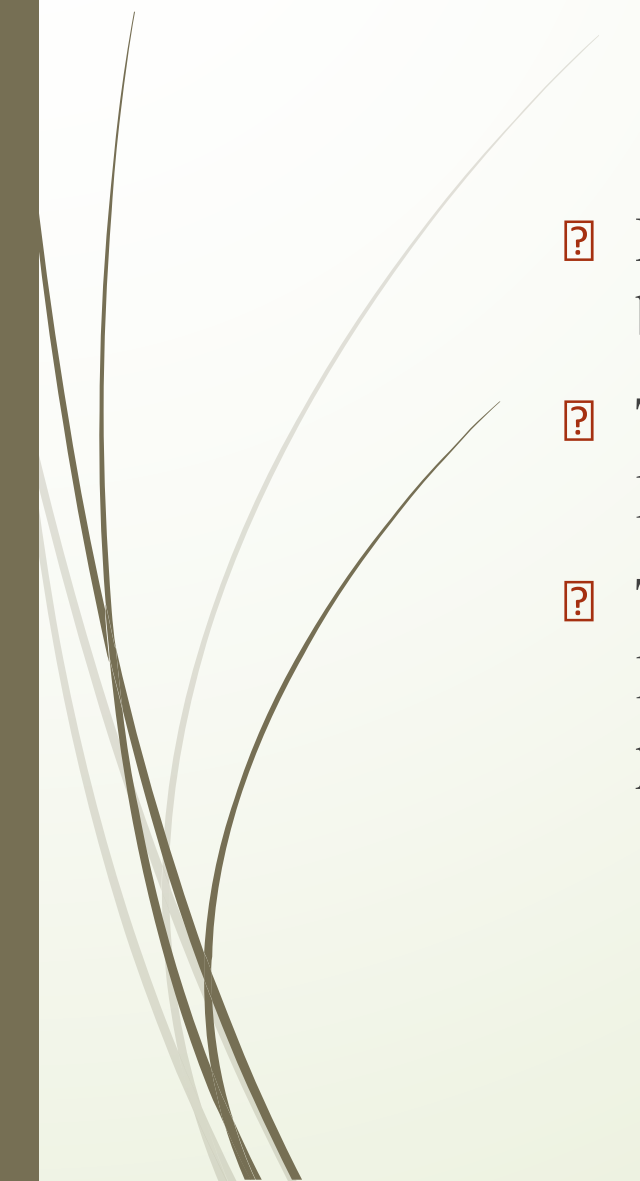
$w_1 = 0.6$, $w_2 = 0.6$ Threshold = 1 and Learning Rate $\eta = 0.5$

A	B	$Y=A+B$
0	0	0
0	1	1
1	0	1
1	1	1



How Perceptron works

- ❑ Examples from the training dataset are shown to the model one at a time, the model makes a prediction, and error is calculated.
- ❑ $\text{ERROR} = \text{Target} - \text{Actual}$
- ❑ The weights of the learning model are then updated to reduce the errors for the example. This is called the Perceptron update rule. This process, repeated for all examples in the training dataset, which is called an epoch. This process of updating the learning model using examples is then repeated for many epochs.

- 
- 
- ❑ Model weights are updated with a small change which is controlled by a hyper parameter called the learning rate.
 - ❑ Typically learning rate is set to a small value. This is to ensure learning does not occur too quickly.
 - ❑ Training is stopped when the error made by the model falls to a low level or no longer improves, or a maximum number of epochs is performed.

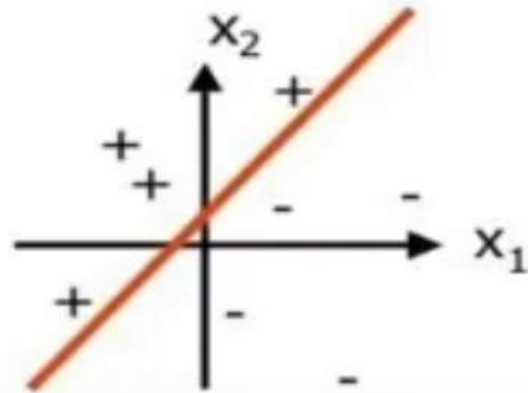


Perceptron Convergence:

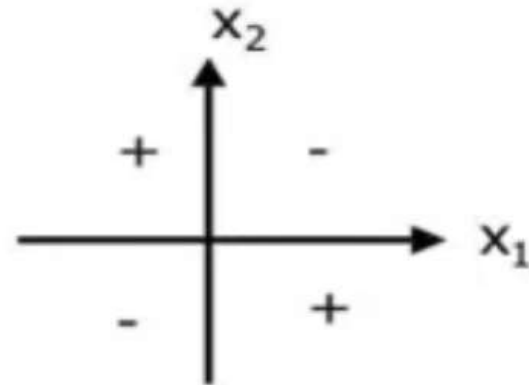
The 'Perceptron convergence theorem' states that for any data set which is linearly separable the Perceptron learning rule is guaranteed to find a solution in a finite number of steps.

Limitations of Perceptron

The Boolean function XOR(exclusive OR) is not linearly separable because its positive and negative instances cannot be separated by a line or hyper plane.



Linearly separable



Non-Linearly separable

Characteristics of ANN

1) Activation function

- This function defines how a neuron's combined input signals are transformed into a single output signal.

2) Network Topology

- It describes the no. of neurons in the model as well as the number of layers and how they are connected.

- It determines the complexity of the task

- Different forms of topology can be differentiated by the following characteristics:

a) No. of layers – Nodes are arranged in layers

First Layer - input layer

Second Layer – Hidden Layer

Last layer – Output Layer



b) Whether information in the network is allowed to travel backward

c) The number of nodes within each layer of the network

Number of input nodes - number of features in the input data

Number of output nodes - number of outcomes to be modelled

Number of hidden nodes - number depends on no. of input nodes, amount of training data, amount of noisy data, complexity of learning task, and many other factors.



3) Training Algorithm

_ it specifies how connection weights are set in order

Mainly two types algorithms are available for learning

1. Perceptron learning Rule – used when training data set is linearly separable.
2. Delta Rule – Used when training dataset is not linearly separable.



Cost function



- ❑ The cost function is the technique of evaluating “**the performance of our algorithm/model**”. It takes both predicted outputs by the model and actual outputs and calculates error in the prediction of model.
- ❑ It outputs a higher number if our predictions differ a lot from the actual values.
- ❑ Loss function: Used when we refer to the error for a single training example.
- ❑ Cost function: Used to refer to an average of the loss functions over an entire training dataset.

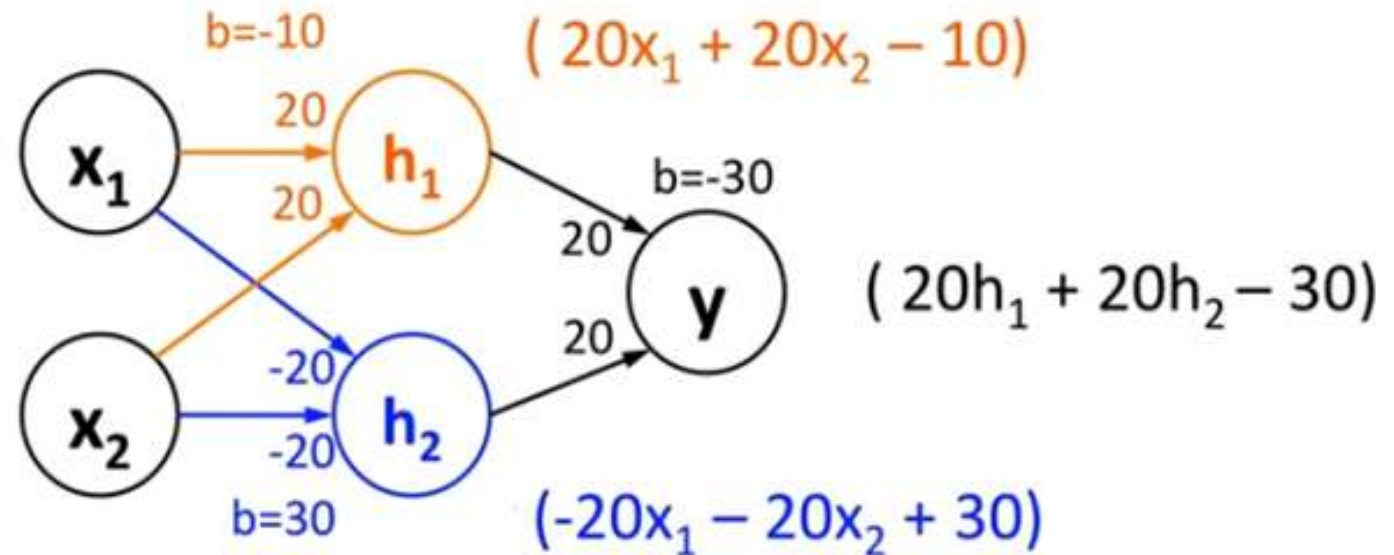
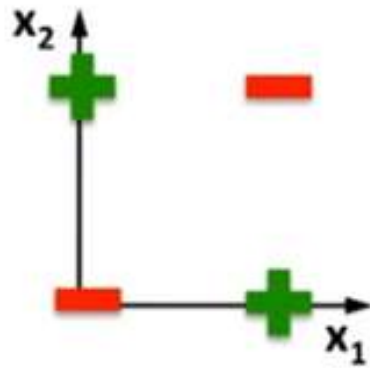


MultiLayer Perceptron(MLP)

- ❑ It is a class of feed forward artificial neural network.
- ❑ MLP consists of at least three layers of nodes:
 - ❑ an input layer,
 - ❑ a hidden layer and
 - ❑ an output layer.
- ❑ Except for the nodes in the input layer, each node uses a nonlinear activation function.
- ❑ MLP utilizes a supervised learning technique called back propagation for training.

Solving XOR with a Neural Net

Linear classifiers cannot solve this



$$(20 * \text{red} + 20 * \text{red} - 10) \approx 0$$

$$(20 * \text{red} + 20 * \text{red} - 10) \approx 1$$

$$(20 * \text{green} + 20 * \text{green} - 10) \approx 1$$

$$(20 * \text{green} + 20 * \text{green} - 10) \approx 1$$

$$(-20 * \text{red} - 20 * \text{red} + 30) \approx 1$$

$$(-20 * \text{red} - 20 * \text{red} + 30) \approx 0$$

$$(-20 * \text{green} - 20 * \text{green} + 30) \approx 1$$

$$(-20 * \text{green} - 20 * \text{green} + 30) \approx 1$$

$$(20 * \text{red} + 20 * \text{blue} - 30) \approx 0$$

$$(20 * \text{red} + 20 * \text{blue} - 30) \approx 0$$

$$(20 * \text{red} + 20 * \text{blue} - 30) \approx 1$$

$$(20 * \text{red} + 20 * \text{blue} - 30) \approx 1$$

Backpropagation Networks (BPN)

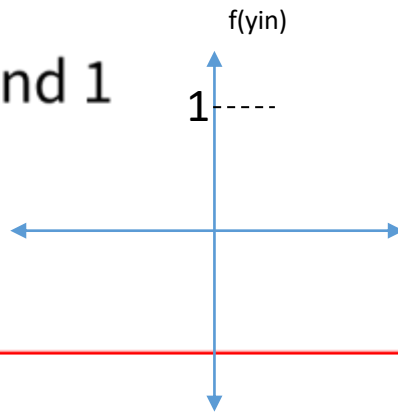
- Backpropagation network is a multilayer neural network containing *an input layer, one or more hidden layers and an output layer*.
- This network is also called multilayer perceptron.
- Neurons present in the hidden layers and output layer have bias inputs.
- The *activation functions are either binary sigmoid or bipolar sigmoid*.

Backpropagation Networks

Binary Sigmoid

$$y = f(y_{in}) = \frac{1}{1 + e^{-y_{in}}}$$

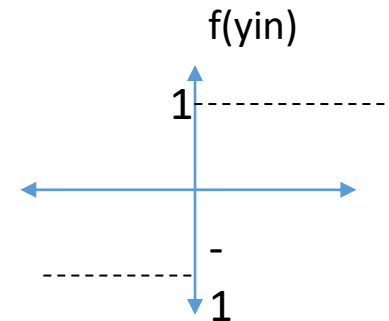
Here y lies between 0 and 1



Bipolar Sigmoid

$$y = f(y_{in}) = \frac{1 - e^{-y_{in}}}{1 + e^{-y_{in}}}$$

Here y lies between -1 and 1



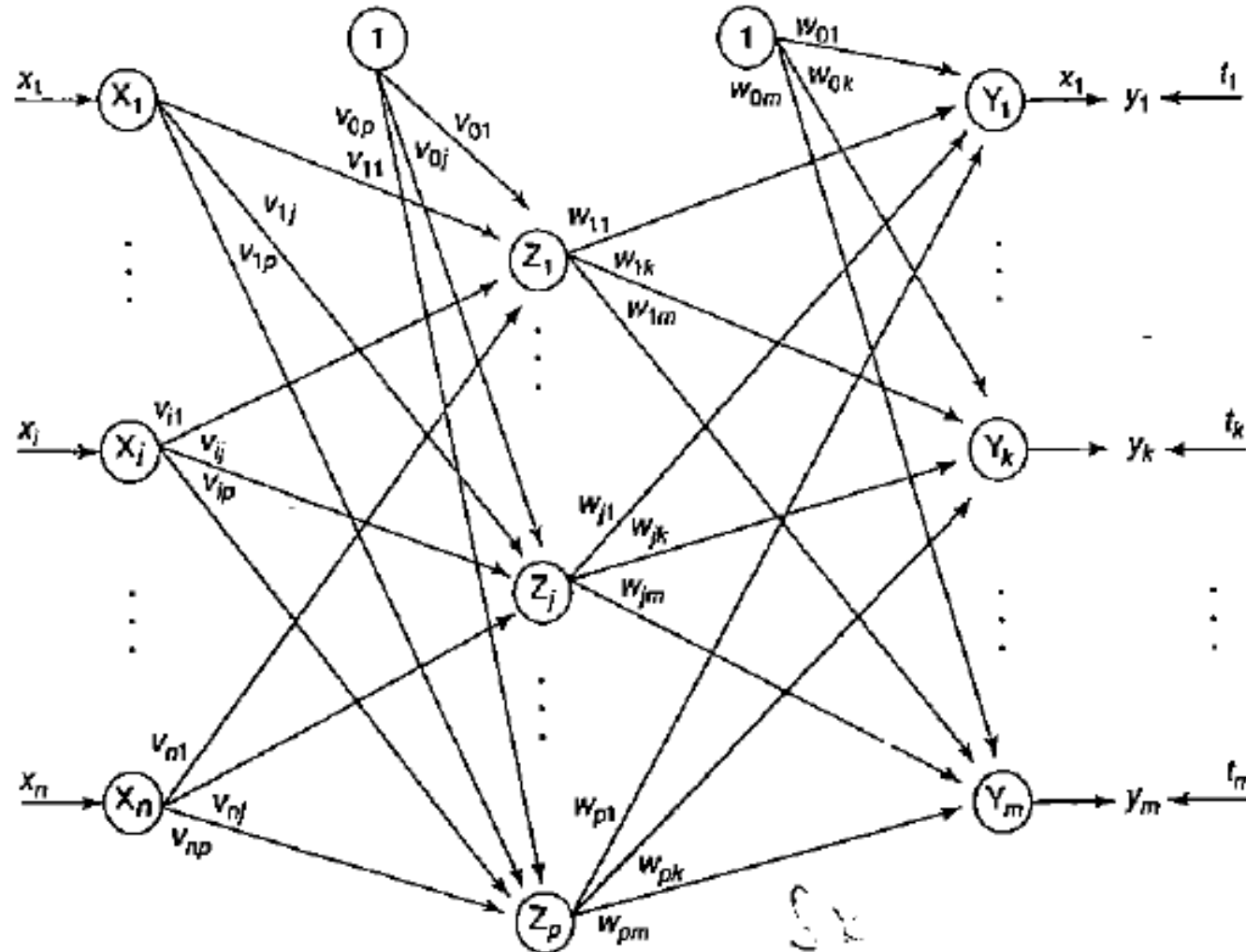
Backpropagation Networks (BPN): Learning Rule

- Learning algorithm provides procedure for changing weights.
- The basic concept for this weight update algorithm is the Gradient descent method used in simple perceptron networks with differentiable units.
- To update weights, error must be calculated.
- In BPN the error is propagated back to the hidden unit.
- The error at output layer can be calculated easily (ie, target – actual).
- The calculation of error at hidden layer is difficult because output of hidden layer is not known .

Backpropagation Networks (BPN): Learning Rule

- When the hidden layers are increased, training become more complex.
- Back propagation algorithm works in three phases.
 - Feed forward phase
 - Backpropagation of errors
 - Weight and bias updation.

Backpropagation Networks (BPN): Learning Rule



Architecture of a back-propagation network.

x – Training Input vector(x_1, x_2, \dots, x_n)

t - Target output vector(t_1, t_2, \dots, t_m)

α – learning rate

X_i – i /p unit i

v_{0j} – bias in j th hidden unit

w_{0k} – bias on k th o/p unit

Z_j – j th Neron in the Hidden Layer

v_{ij} – weight of link from i^{th} I/P unit to j^{th} Node in hidden layer.

w_{jk} - weight of link from j^{th} Node in the hidden layer to k^{th} Node in the output layer.

Backpropagation Networks (BPN): Learning Rule

1. Net input to Z_j

$$z_{inj} = v_{oj} + \sum_{i=1}^n x_i v_{ij}$$

2. Output of Z_j

$$z_j = f(z_{inj}) = \frac{1}{1+e^{-z_{inj}}} \text{ or } \frac{1-e^{-z_{inj}}}{1+e^{-z_{inj}}}$$

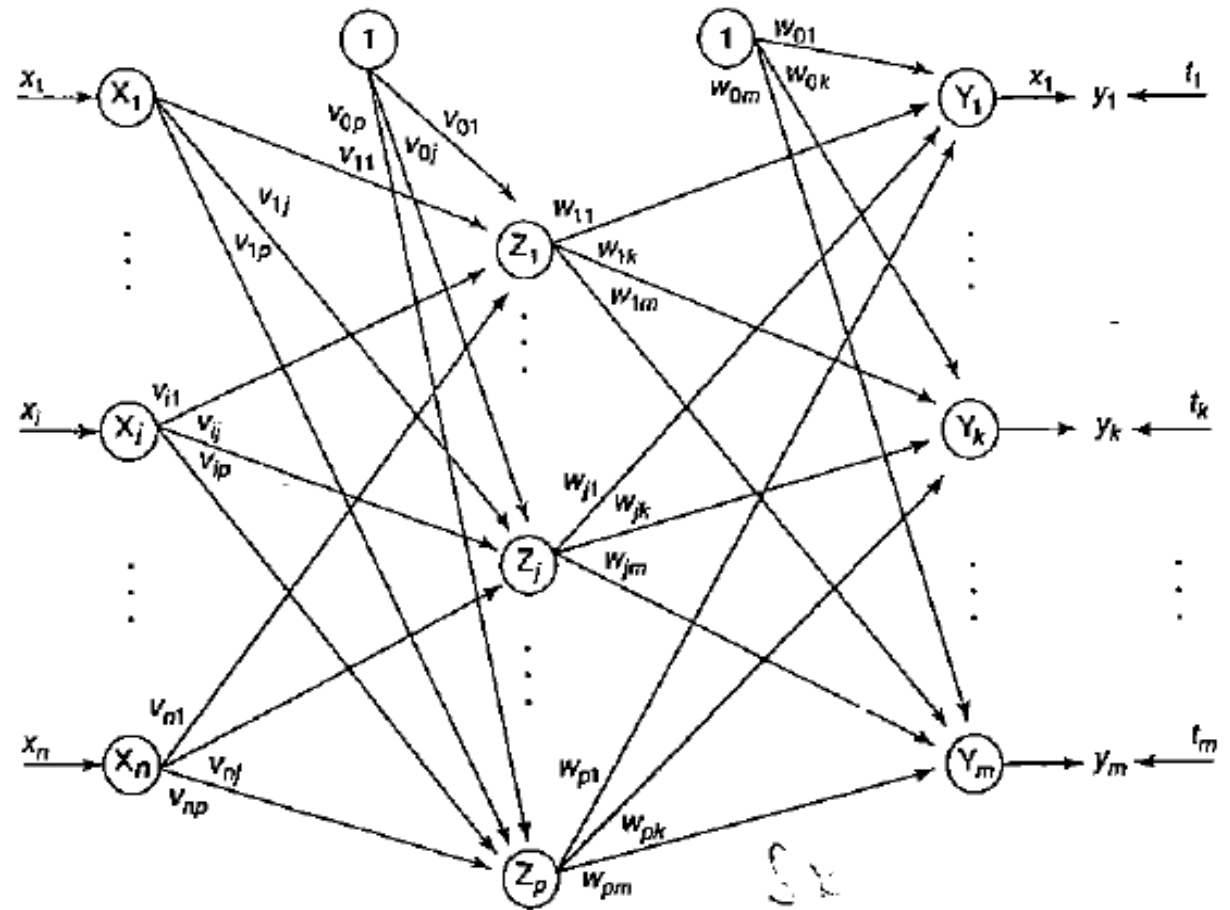
3. Y_k - k^{th} o/p unit

4. net input to Y_k

$$y_{ink} = w_{ok} + \sum_{j=1}^p z_j w_{jk}$$

5. output of Y_k

$$y_k = f(y_{ink}) = \frac{1}{1+e^{-y_{ink}}} \text{ or } \frac{1-e^{-y_{ink}}}{1+e^{-y_{ink}}}$$



Architecture of a back-propagation network.

Backpropagation Networks

Training Algorithm

1. Feed forward of input training pattern – calculate the net input of each neuron.
2. Calculation of back propagation of error – calculate errors for weight updation
3. Updation of weights – weights updation of all units (including bias)

Testing Algorithm

Computation of feedforward phase only – calculation of output value

Backpropagation Networks : Training Algorithm

- 1. Initialize weights and learning parameter (to some small value).
 2. Perform steps 3 -10 when stopping condition is false
 3. Perform steps 4 -9 for each training pair

Phase 1 Feedforward phase

4. Each input unit receives input signal x_i and send it to hidden unit($i=1$ to n).
5. Each hidden unit $Z_j(j=1$ to $p)$ calculate the net input.

$$Z_{inj} = v_{oj} + \sum_{i=1}^n x_i v_{ij}$$

5. Also calculate the output of hidden units

$$z_j = f(z_{inj}) = \frac{1}{1+e^{-z_{inj}}} \text{ or } \frac{1-e^{-z_{inj}}}{1+e^{-z_{inj}}}$$

6. For each output unit Y_k calculate the net input

$$y_{ink} = w_{0k} + \sum_{j=1}^p z_j w_{jk}$$

Also calculate the output of output units

$$y_k = f(y_{ink}) = \frac{1}{1+e^{-y_{ink}}} \text{ or } \frac{1-e^{-y_{ink}}}{1+e^{-y_{ink}}}$$

Backpropagation Networks : Training Algorithm

Phase II Backpropagating Error

7. Calculate error at output layer and send this error back to hidden layer.

Error at output node k is denoted by δ_k or Err_k

$$\delta_k = (t_k - y_k) y_k (1 - y_k)$$

if bipolar activation function

$$\delta_k = (t_k - y_k) 0.5(1 + y_k) (1 - y_k)$$

8. Calculate the error at the hidden layer.

Error at j^{th} hidden layer neuron = δ_j or Err_j

$$\begin{aligned}\delta_j &= z_j(1 - z_j) \delta_{\text{inj}} \\ &= z_j(1 - z_j) \sum_{k=1}^m \delta_k w_{jk}\end{aligned}$$

$$\delta_j = 0.5(1 + z_j) (1 - z_j) \sum_{k=1}^m \delta_k w_{jk}$$

Backpropagation Networks : Training Algorithm

Phase III :Weights and Bias updations

9. For each neuron in the Hidden Layer

$$v_{ij}(\text{new}) = v_{ij}(\text{old}) + \alpha \delta_j x_i$$

$$v_{0j}(\text{new}) = v_{0j}(\text{old}) + \alpha \delta_j$$

For each neuron in the Output Layer

$$w_{jk}(\text{new}) = w_{jk}(\text{old}) + \alpha \delta_k z_j$$

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

10. Check for the stopping condition. The stopping condition may be certain number of epochs or calculated output = target output

Equations (based on the following n/w)

Net i/p hidden layer

$$z_{in1} = v_{01} + x_1 v_{11} + x_2 v_{21}$$

$$z_{in2} = v_{02} + x_1 v_{12} + x_2 v_{22}$$

Output Hidden layer

$$z_1 = \frac{1}{1 + e^{-z_{in1}}}$$

$$z_2 = \frac{1}{1 + e^{-z_{in2}}}$$

Net i/p output layer

$$y_{in1} = w_{01} + z_1 w_{11} + z_2 w_{21}$$

$$y_{in2} = w_{02} + z_1 w_{12} + z_2 w_{22}$$

Output of output layer

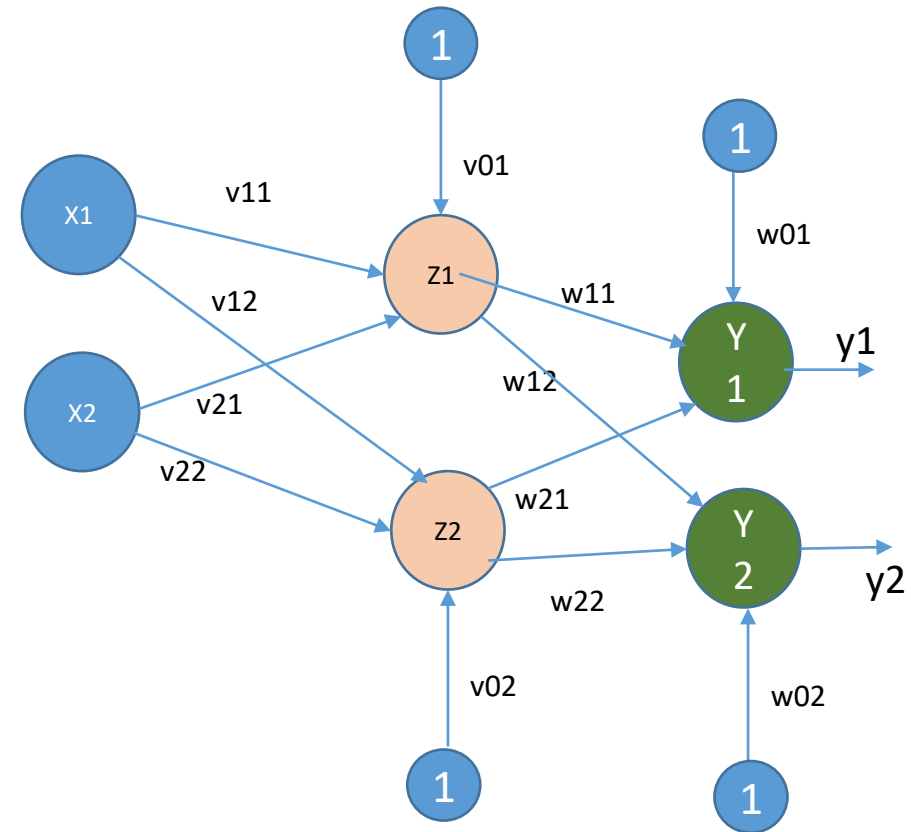
$$y_1 = \frac{1}{1 + e^{-y_{in1}}}$$

$$y_2 = \frac{1}{1 + e^{-y_{in2}}}$$

Error output layer

$$\delta_1 = (t_1 - y_1) y_1 (1 - y_1)$$

$$\delta_2 = (t_2 - y_2) y_2 (1 - y_2)$$



Error hidden layer

$$\delta_{z1} = z_1 (1 - z_1) [\delta_1 w_{11} + \delta_2 w_{12}]$$

$$\delta_{z2} = z_2 (1 - z_2) [\delta_1 w_{21} + \delta_2 w_{22}]$$

Backpropagation Networks : Training Algorithm

Phase III :Weights and Bias updatation

9. For neurons in the Hidden Layer

$$v_{11}(\text{new}) = v_{11}(\text{old}) + \alpha \delta_{z1} x_1$$

$$v_{12}(\text{new}) = v_{12}(\text{old}) + \alpha \delta_{z2} x_1$$

$$v_{21}(\text{new}) = v_{21}(\text{old}) + \alpha \delta_{z1} x_2$$

$$v_{22}(\text{new}) = v_{22}(\text{old}) + \alpha \delta_{z2} x_2$$

$$v_{01}(\text{new}) = v_{01}(\text{old}) + \alpha \delta_{z1}$$

$$v_{02}(\text{new}) = v_{02}(\text{old}) + \alpha \delta_{z2}$$

For neurons in the Output Layer

$$w_{11}(\text{new}) = w_{11}(\text{old}) + \alpha \delta_1 z_1$$

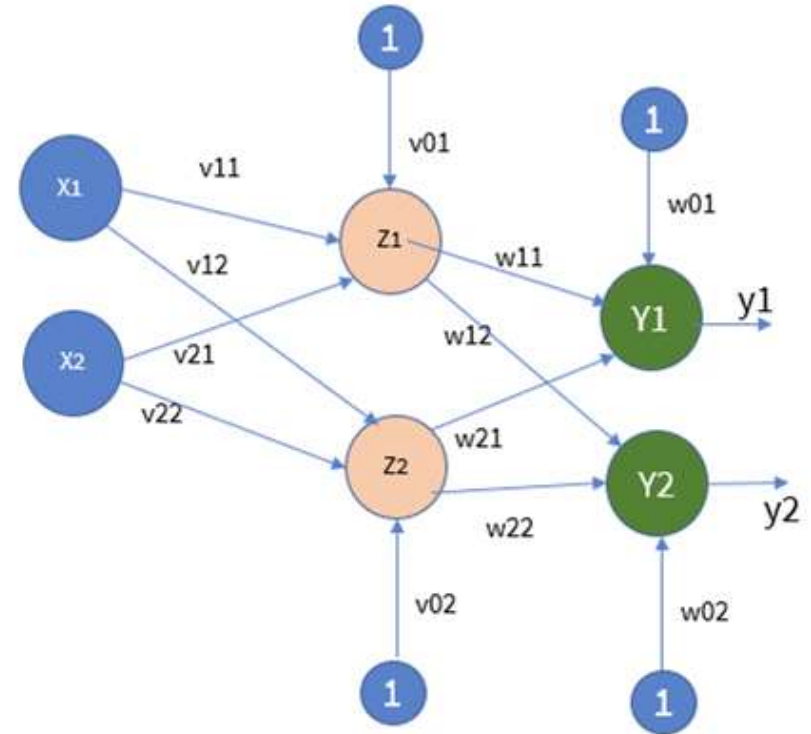
$$w_{12}(\text{new}) = w_{12}(\text{old}) + \alpha \delta_2 z_1$$

$$w_{21}(\text{new}) = w_{21}(\text{old}) + \alpha \delta_1 z_2$$

$$w_{22}(\text{new}) = w_{22}(\text{old}) + \alpha \delta_2 z_2$$

$$w_{01}(\text{new}) = w_{01}(\text{old}) + \alpha \delta_1$$

$$w_{02}(\text{new}) = w_{02}(\text{old}) + \alpha \delta_2$$



Use BPN find new weights for the following network.

Input pattern is [0,1] and output is 1, learning rate is 0.25. use binary sigmoid activation function.

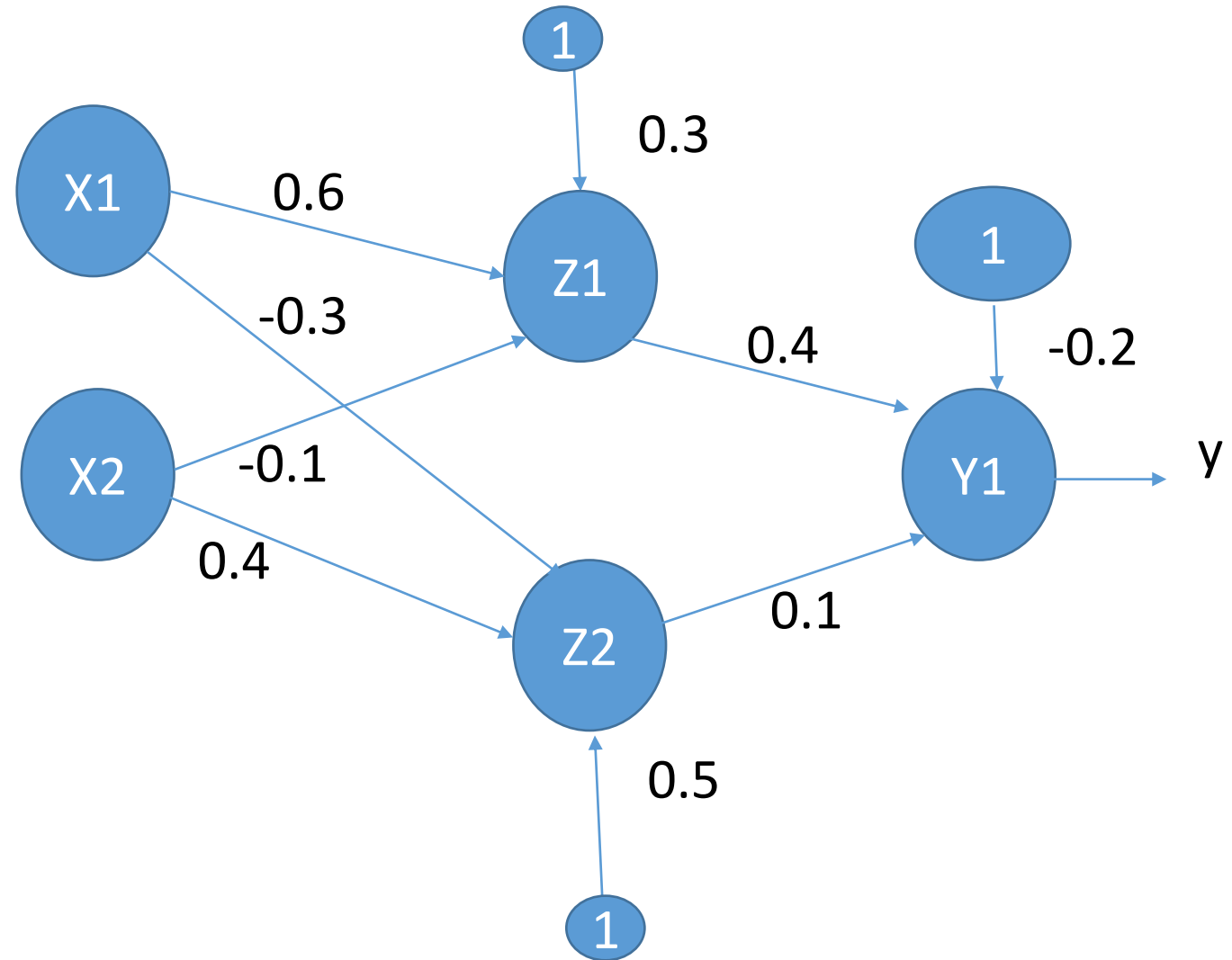
Given

$[x_1 \ x_2] = [0 \ 1] \quad t=1$

$\alpha = 0.25$

$$\begin{bmatrix} v_{01} & v_{02} \\ v_{11} & v_{12} \\ v_{21} & v_{22} \end{bmatrix} = \begin{bmatrix} 0.3 & 0.5 \\ 0.6 & -0.3 \\ -0.1 & 0.4 \end{bmatrix}$$

$$\begin{bmatrix} w_{01} \\ w_{11} \\ w_{21} \end{bmatrix} = \begin{bmatrix} -0.2 \\ 0.4 \\ 0.1 \end{bmatrix}$$



Phase 1 Forward Phase

Net input at hidden layer

$$Z1_{(in)} = v_{01} + x_1 v_{11} + x_2 v_{21} = 0.3 + 0 * 0.6 + 1 * -0.1 = 0.2$$

$$Z2_{(in)} = v_{02} + x_1 v_{12} + x_2 v_{22} = 0.5 + 0 * -0.3 + 1 * 0.4 = 0.9$$

Output Hidden layer

$$Z1_{(out)} = \frac{1}{1 + e^{-zin1}}$$

$$= \frac{1}{1 + e^{-0.2}} = 0.5498$$

$$Z2_{(out)} = \frac{1}{1 + e^{-zin2}}$$

$$= \frac{1}{1 + e^{-0.9}} = 0.711$$

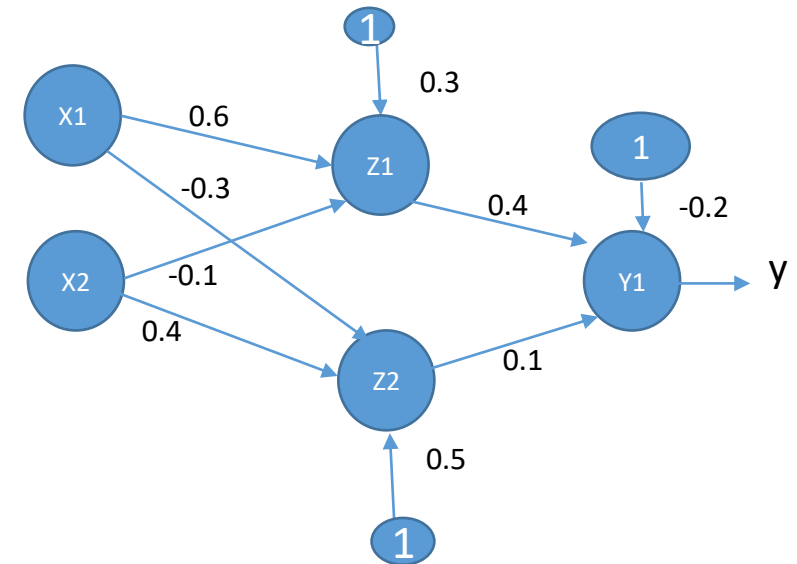
Given

$$[x1 \ x2] = [0 \ 1] \quad t=1$$

$$\alpha = 0.25$$

$$\begin{bmatrix} v_{01} & v_{02} \\ v_{11} & v_{12} \\ v_{21} & v_{22} \end{bmatrix} = \begin{bmatrix} 0.3 & 0.5 \\ 0.6 & -0.3 \\ -0.1 & 0.4 \end{bmatrix}$$

$$\begin{bmatrix} w_{01} \\ w_{11} \\ w_{21} \end{bmatrix} = \begin{bmatrix} -0.2 \\ 0.4 \\ 0.1 \end{bmatrix}$$



Phase 1 Forward Phase

Net input at output layer

$$\begin{aligned} Y1_{(in)} &= w_{01} + z_1 w_{11} + z_2 w_{21} \\ &= -0.2 + 0.5498 * 0.4 + 0.711 * 0.1 \\ &= 0.0910 \end{aligned}$$

Output of output layer

$$Y1_{(out)} = \frac{1}{1 + e^{-y1(in)}} = \frac{1}{1 + e^{-0.0910}} = \mathbf{0.5227}$$

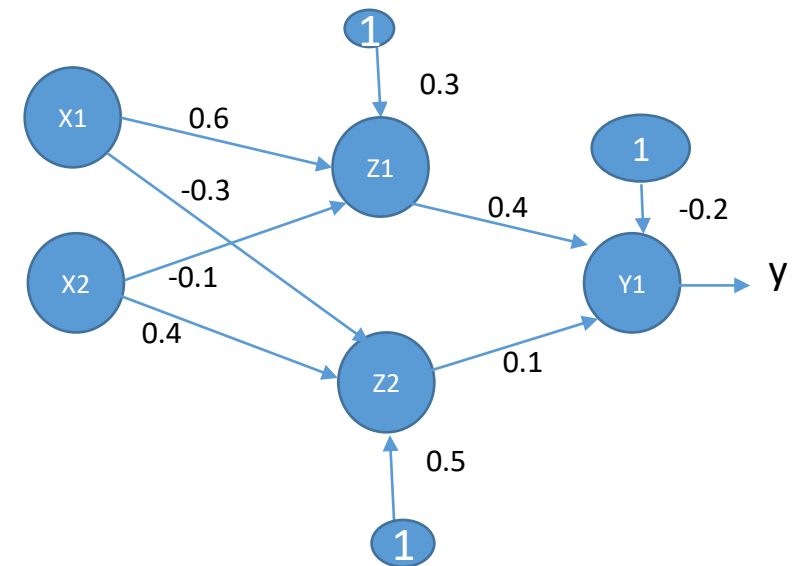
Given

$$[x1 \ x2] = [0 \ 1] \quad t=1$$

$$\alpha = 0.25$$

$$\begin{bmatrix} v_{01} & v_{02} \\ v_{11} & v_{12} \\ v_{21} & v_{22} \end{bmatrix} = \begin{bmatrix} 0.3 & 0.5 \\ 0.6 & -0.3 \\ -0.1 & 0.4 \end{bmatrix}$$

$$\begin{bmatrix} w_{01} \\ w_{11} \\ w_{21} \end{bmatrix} = \begin{bmatrix} -0.2 \\ 0.4 \\ 0.1 \end{bmatrix}$$



Phase 2. Error Calculation

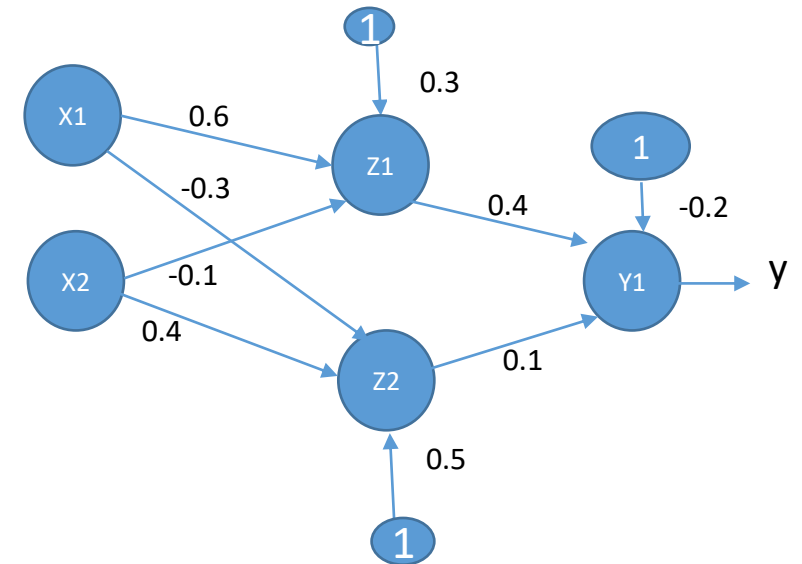
Error output layer

$$\begin{aligned}\text{Err}(Y1) &= (t_1 - y_1) y_1 (1 - y_1) \\ &= (1 - 0.5227) * 0.5227 * (1 - 0.5227) \\ &= 0.11908\end{aligned}$$

$$Y1 = 0.5227$$

$$Z1 = 0.5498$$

$$Z2 = 0.711$$



Phase 2. Error Calculation

Error hidden layer

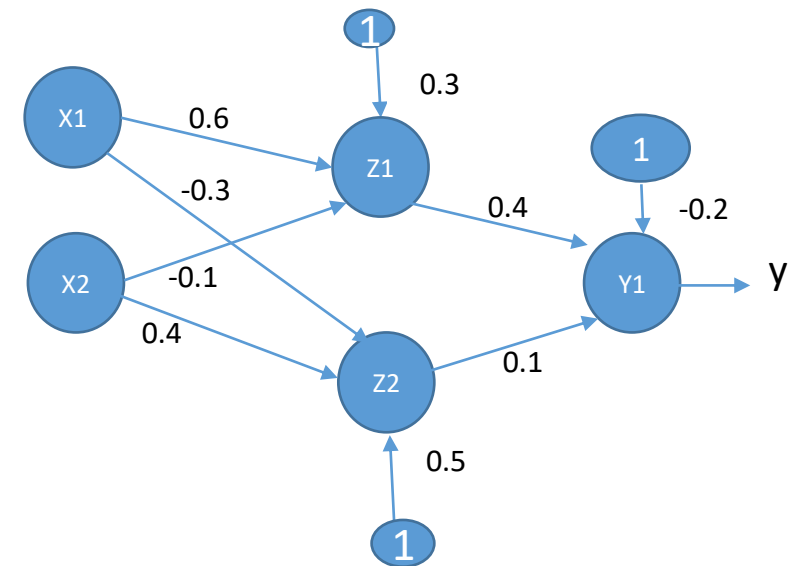
$$\begin{aligned}\text{Err}(Z1) &= z_1(1 - z_1) [\text{Err}(Y1) * w_{11}] \\ &= 0.5498(1-0.5498)[0.11908 * 0.4] \\ &= 0.01178\end{aligned}$$

$$\begin{aligned}\text{Err}(Z2) &= z_2(1 - z_2) [\text{Err}(Y1) * w_{21}] \\ &= 0.711(1-0.711)[0.11908 * 0.1] \\ &= 0.00245\end{aligned}$$

$$Y1 = 0.5227$$

$$Z1 = 0.5498$$

$$Z2 = 0.711$$



Phase 3. Weight Updation

For neurons in the Hidden Layer

$$\begin{aligned}v_{11}(\text{new}) &= v_{11}(\text{old}) + \alpha * \text{Err}(z1) * x_1 \\ &= 0.6 + 0.25 * 0.01178 * 0 = 0.6\end{aligned}$$

$$\begin{aligned}v_{21}(\text{new}) &= v_{21}(\text{old}) + \alpha * \text{Err}(z1) * x_2 \\ &= -0.1 + 0.25 * 0.01178 * 1 = -0.09706\end{aligned}$$

$$\begin{aligned}v_{12}(\text{new}) &= v_{12}(\text{old}) + \alpha * \text{Err}(z2) * x_1 \\ &= -0.3 + 0.25 * 0.00245 * 0 = -0.3\end{aligned}$$

$$\begin{aligned}v_{22}(\text{new}) &= v_{22}(\text{old}) + \alpha * \text{Err}(z2) * x_2 \\ &= 0.4 + 0.25 * 0.00245 * 1 = 0.4006125\end{aligned}$$

$$\begin{aligned}v_{01}(\text{new}) &= v_{01}(\text{old}) + \alpha * \text{Err}(Z1) \\ &= 0.3 + 0.25 * 0.01178 = 0.302945\end{aligned}$$

$$\begin{aligned}v_{02}(\text{new}) &= v_{02}(\text{old}) + \alpha * \text{Err}(Z2) \\ &= 0.5 + 0.25 * 0.00245 = 0.5006125\end{aligned}$$

$$Y1 = 0.5227$$

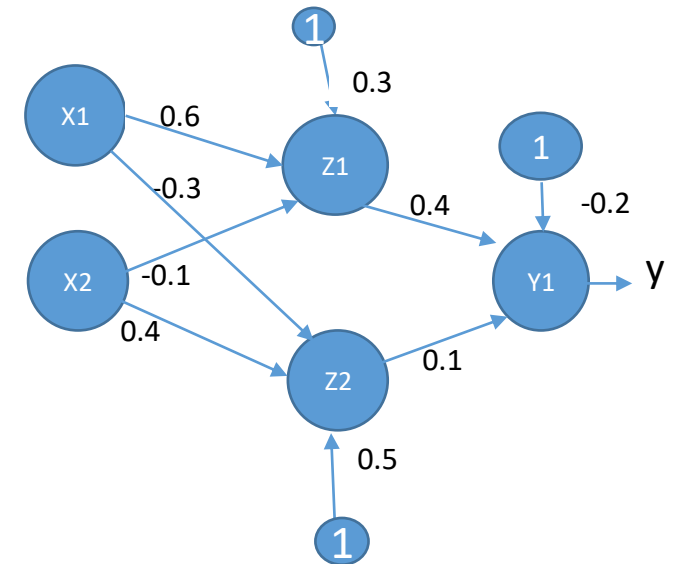
$$Z1 = 0.5498$$

$$Z2 = 0.711$$

$$\delta_{z1} = 0.01178$$

$$\delta_{z2} = 0.00245$$

$$\delta_1 = 0.11908$$



For neurons in the Output Layer

$$\begin{aligned}w_{11}(\text{new}) &= w_{11}(\text{old}) + \alpha * \text{Err}(Y1) * Z1_{(\text{out})} \\&= 0.4 + 0.25 * 0.11908 * 0.5498 \\&= 0.41637\end{aligned}$$

$$\begin{aligned}w_{21}(\text{new}) &= w_{21}(\text{old}) + \alpha * \text{Err}(Y1) * Z2_{(\text{out})} \\&= 0.1 + 0.25 * 0.11908 * 0.711 \\&= 0.12117\end{aligned}$$

$$\begin{aligned}w_{01}(\text{new}) &= w_{01}(\text{old}) + \alpha * \text{Err}(Y1) \\&= -0.2 + 0.25 * 0.11908 \\&= -0.17023\end{aligned}$$

$$Y1 = 0.5227$$

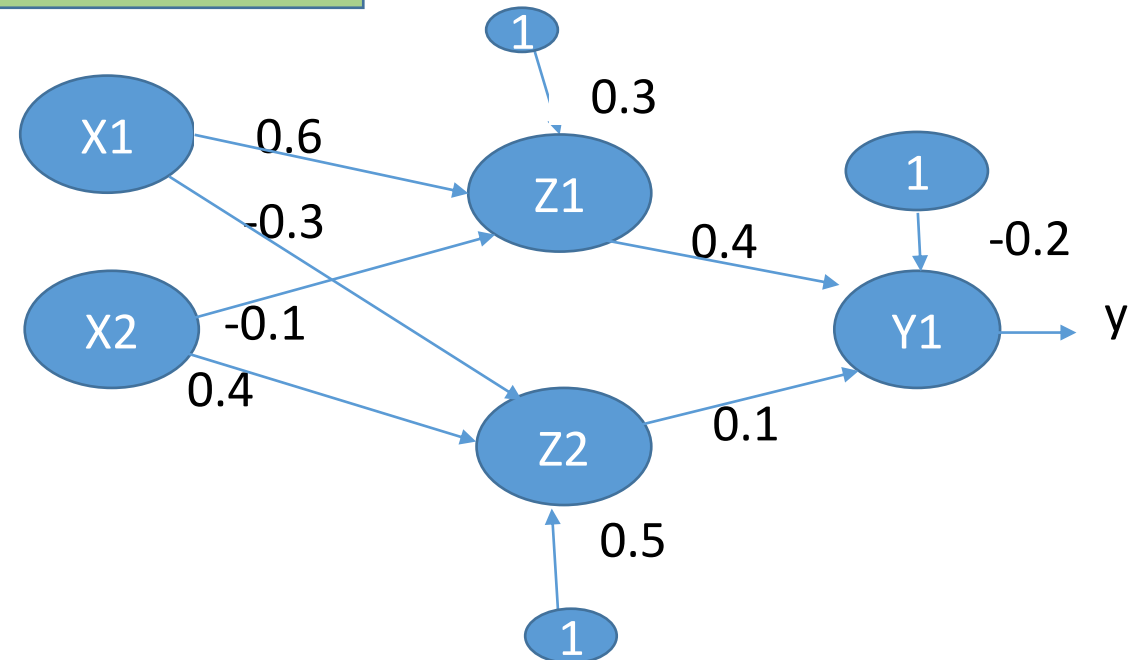
$$Z1 = 0.5498$$

$$Z2 = 0.711$$

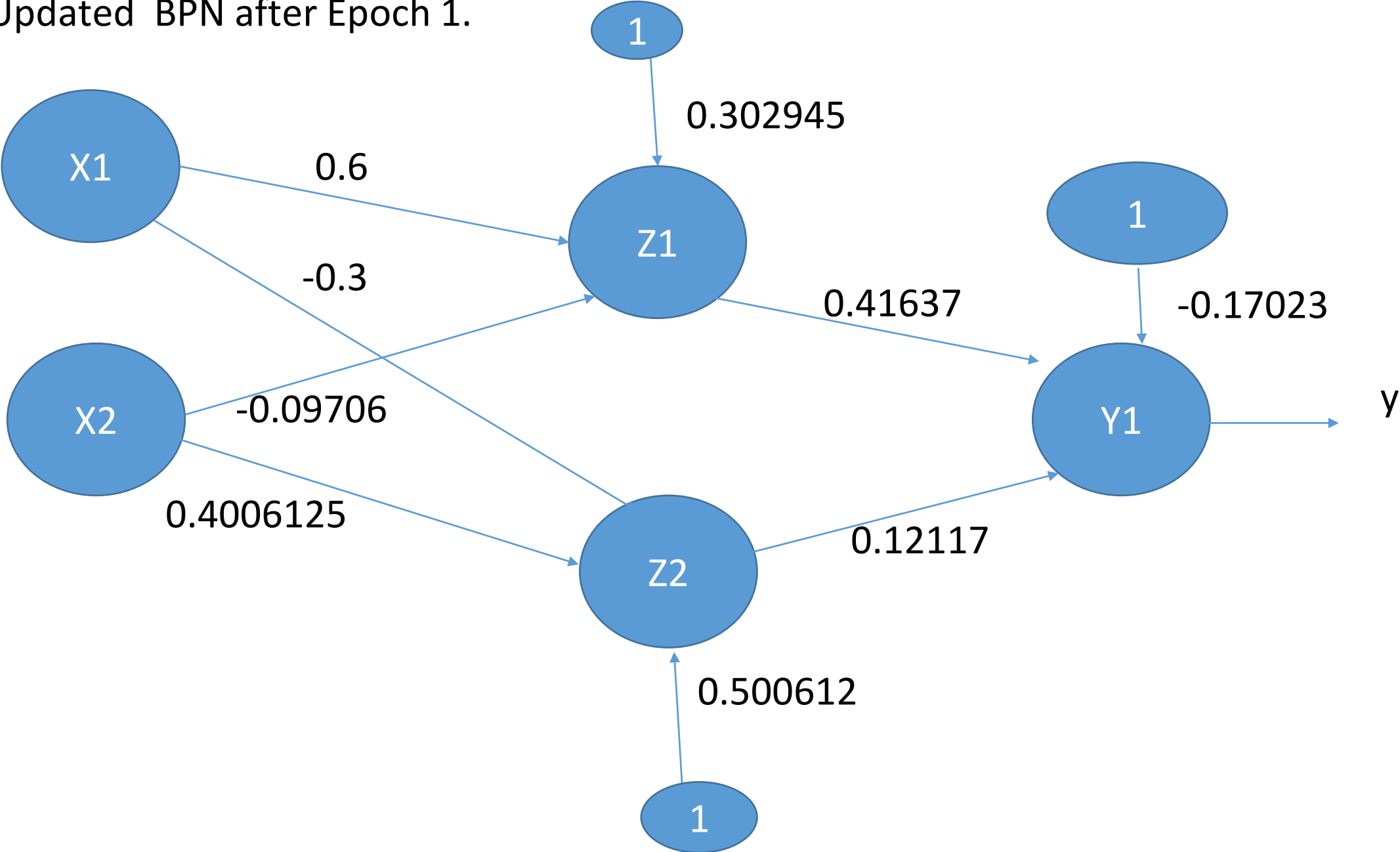
$$\delta_{z1} = 0.01178$$

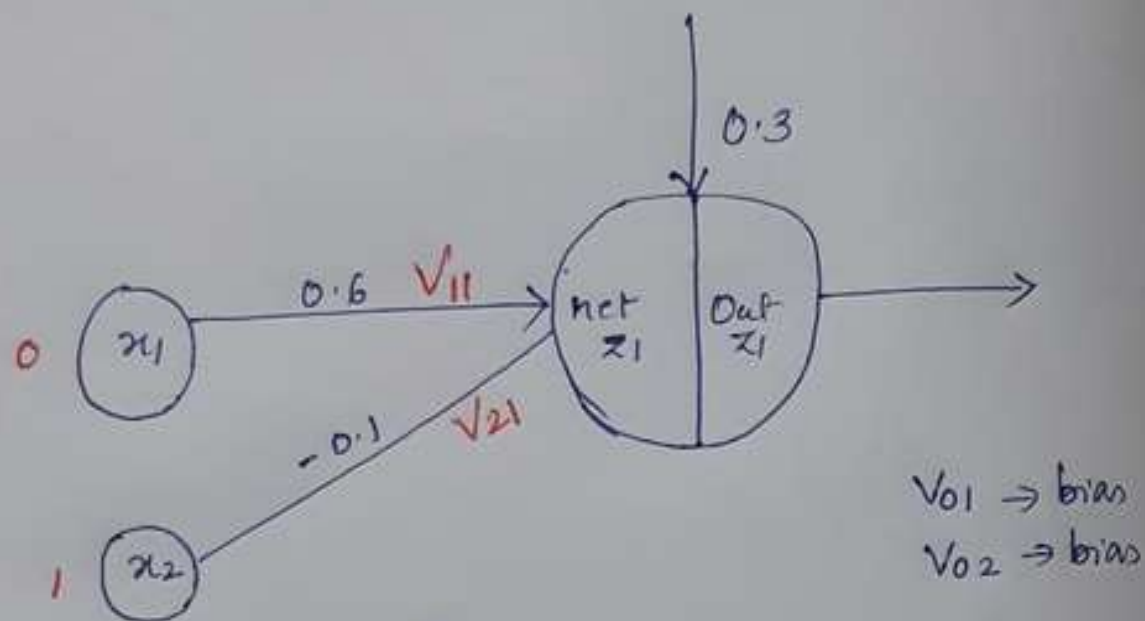
$$\delta_{z2} = 0.00245$$

$$\delta_1 = 0.11908$$



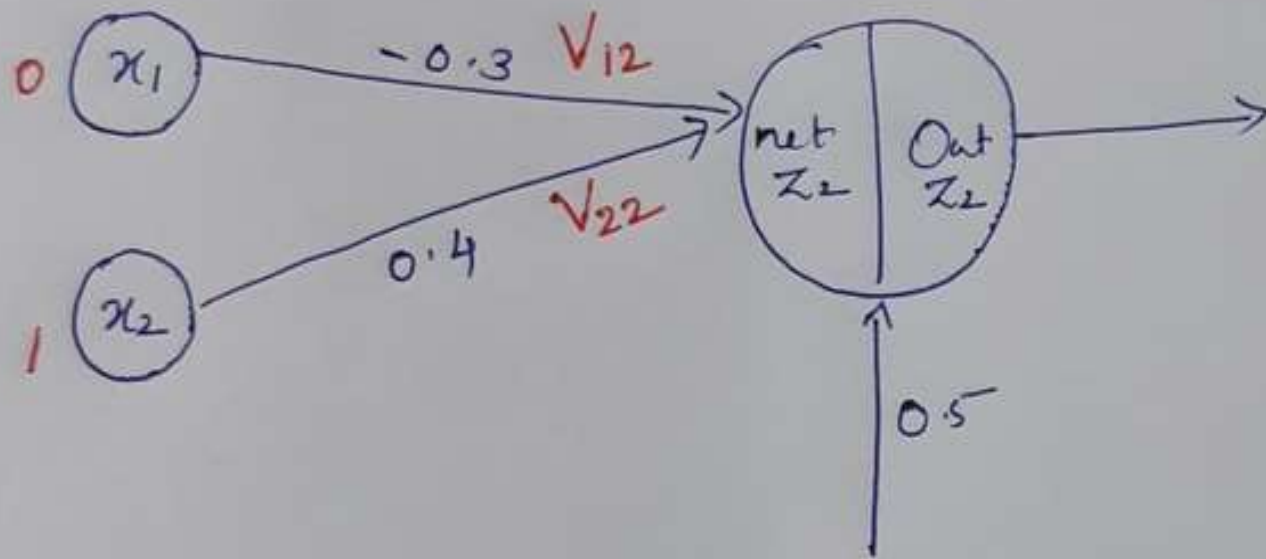
Updated BPN after Epoch 1.





Net input at hidden layer

$$\begin{aligned}
 z_{1(\text{in})} &= 0.3 + x_1 V_{11} + x_2 V_{21} \\
 &= 0.3 + 0 \times 0.6 + 1 \times -0.1 = 0.2
 \end{aligned}$$

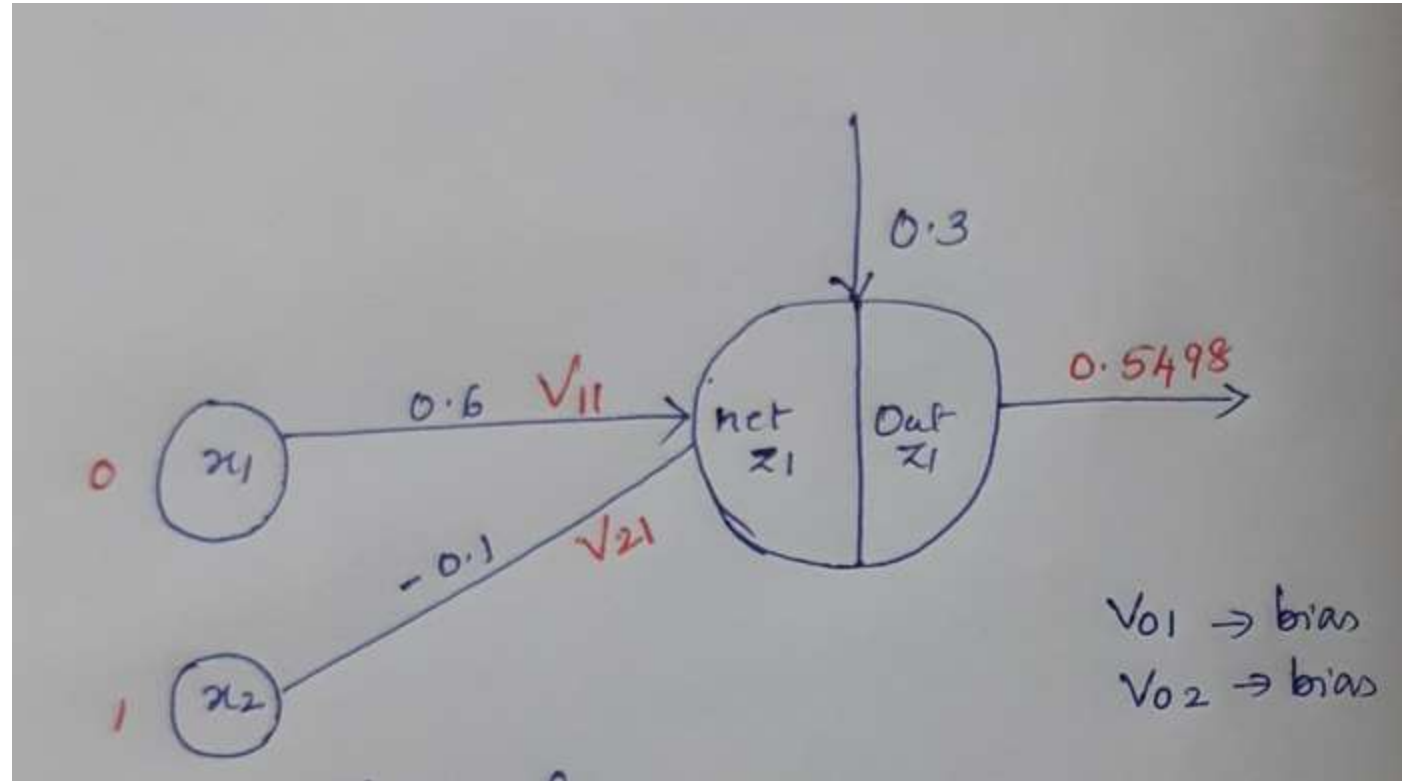


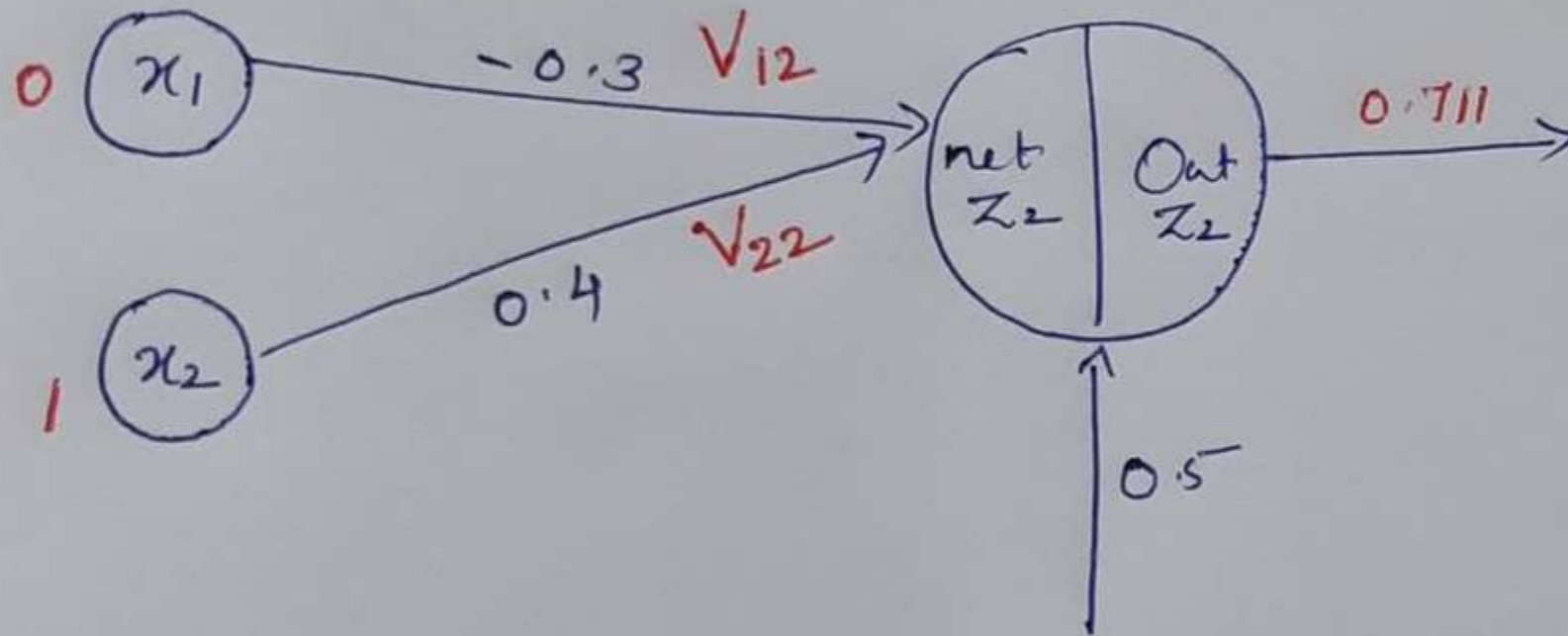
$$\begin{aligned} Z_2(in) &= 0.5 + x_1 V_{12} + x_2 \times V_{22} \\ &= 0.5 + 0 \times -0.3 + 1 \times 0.4 = 0.9 \end{aligned}$$

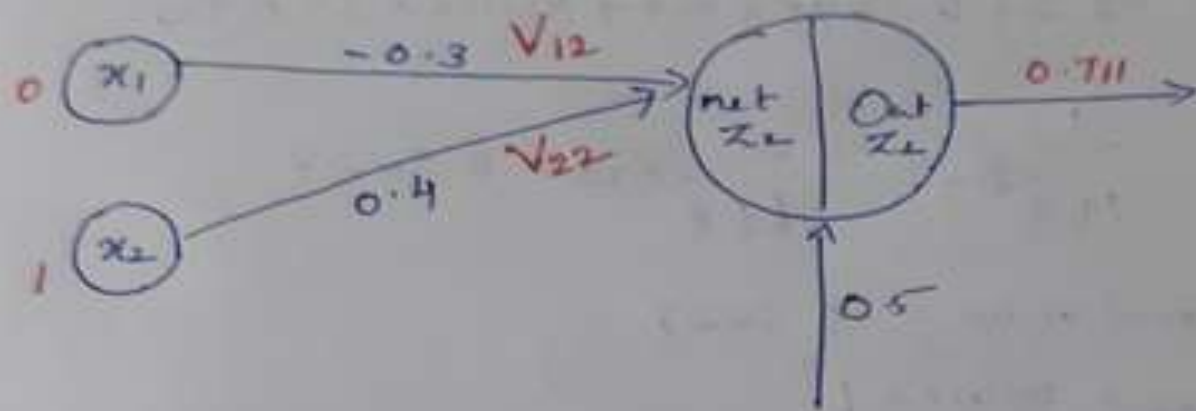
Output at Hidden Layer

$$Z1_{out} = \frac{1}{1 + e^{-Z1_{in}}} = \frac{1}{1 + e^{-0.2}} = 0.5498$$

$$Z2_{out} = \frac{1}{1 + e^{-Z2_{in}}} = \frac{1}{1 + e^{-0.9}} = 0.711$$







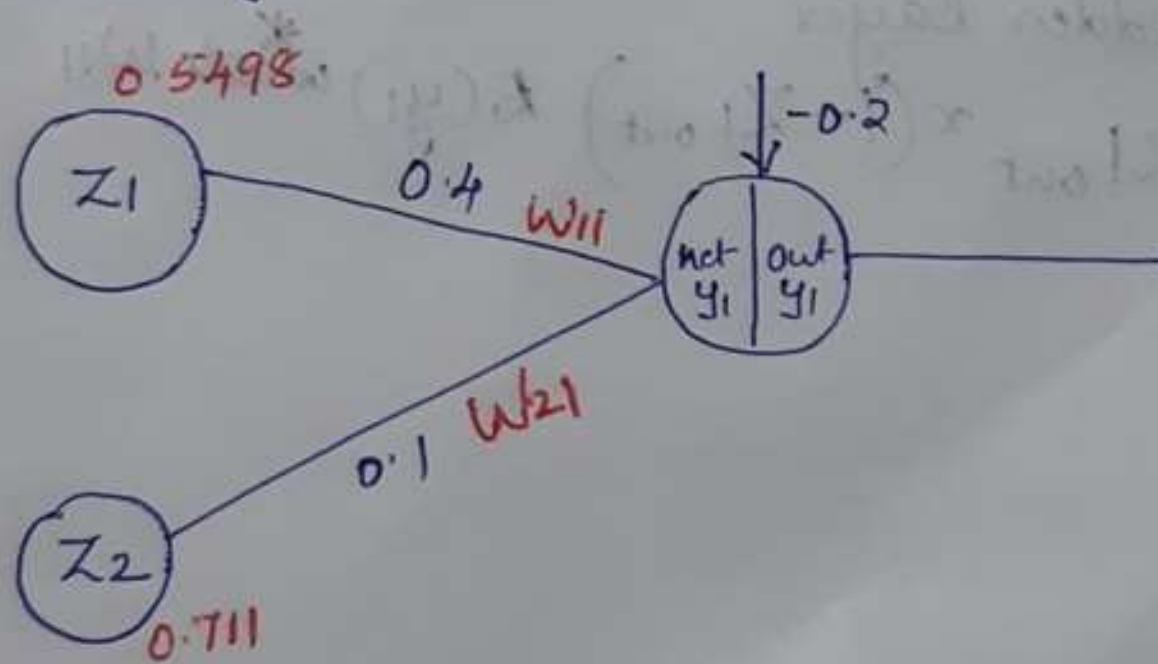
$$Z_2(\text{in}) = 0.5 + x_1 V_{12} + x_2 V_{22} \\ = 0.5 + 0 \times -0.3 + 1 \times 0.4 = 0.9$$

Output at Hidden Layer

$$Z_{1\text{out}} = \frac{1}{1 + e^{-Z_{1\text{in}}}} = \frac{1}{1 + e^{-0.2}} = 0.5498$$

$$Z_{2\text{out}} = \frac{1}{1 + e^{-Z_{2\text{in}}}} = \frac{1}{1 + e^{-0.9}} = 0.711$$

output Layer



$$y_{1(in)} = -0.2 + 0.5498 \times 0.4 + 0.711 \times 0.1 = 0.910$$

$$y_1(ut) = \frac{1}{1 + e^{-y_{1in}}} = \frac{1}{1 + e^{-0.910}} = 0.5227$$

Observed value = 0.5227

Target value = 1

$$y_{1(in)} = -0.2 + 0.5498 \times 0.4 + 0.711 \times 0.1 = 0.910$$

$$y_1(ut) = \frac{1}{1 + e^{-y_{1(in)}}} = \frac{1}{1 + e^{-0.910}} = 0.5227$$

Observed value = 0.5227

Target value = 1

Error at Output layer

$$\begin{aligned} E(y_i) &= (y_{i\text{target}} - y_{i\text{out}}) * y_{i\text{out}} * (1 - y_{i\text{out}}) \\ &= (1 - 0.5227) * 0.5227 * (1 - 0.5227) \\ &= \underline{\underline{0.11908}} \end{aligned}$$

Error at Hidden Layer

$$E(z_1) = z_{1\text{out}} * (1 - z_{1\text{out}}) * E(y_1) * w_{11}$$

$$= 0.5498 * (1 - 0.5498) * 0.11908 * 0.4..$$

$$= 0.01178$$

Hey

$$E(z_2) = z_{2\text{out}} * (1 - z_{2\text{out}}) * E(y_1) * w_{21}$$

$$= 0.00245$$

Weight updations in hidden layer

$$V_{11}(\text{new}) = V_{11}(\text{old}) + \alpha * E(z_1) * x_1$$

$$V_{21}(\text{new}) = V_{21}(\text{old}) + \alpha * E(z_1) * x_2$$

$$V_{11}(\text{new}) = 0.6 + 0.25 * 0.01178 * 0 = 0.6$$

$$V_{21}(\text{new}) = -0.1 + 0.25 * 0.01178 * 1 = -0.09706$$

$$V_{12}(\text{new}) = V_{12}(\text{old}) + \alpha * E(z_2) * x_1$$

$$V_{22}(\text{new}) = V_{22}(\text{old}) + \alpha * E(z_2) * x_2$$

$$V_{12}(\text{new}) = -0.3$$

$$V_{22}(\text{new}) = 0.4006125$$

New Bias values

$$\begin{aligned} V_{01}(\text{new}) &= V_{01}(\text{old}) + \alpha * E(z_1) \\ &= 0.3 + 0.25 * 0.01178 = 0.302945 \end{aligned}$$

$$V_{02}(\text{new}) = V_{02}(\text{old}) + \alpha * E(z_2) = 0.5006125$$

Weight updations in Output Layer

$$\begin{aligned}W_{11}(\text{new}) &= W_{11}(\text{old}) + \alpha E(y_1) * Z_1 \\&= 0.4 + 0.25 \times 0.11908 * 0.5498 \\&= 0.41637\end{aligned}$$

$$W_{21}(\text{new}) = W_{21}(\text{old}) + \alpha E(y_1) * Z_2 = 0.12117$$

new Bias values

$$\begin{aligned}W_{01}(\text{new}) &= W_{01}(\text{old}) + \alpha E(y_1) \\&= -0.2 + 0.25 * 0.11908 = \underline{\underline{-0.17023}}\end{aligned}$$

Loss function and cost function

- During training, we predict the output of a model for different inputs and compare the predicted output with actual output in our training set.
- The *difference in actual and predicted output is termed as loss over that input.*
- *The summation of losses across all inputs is termed as cost function.*

$$\sum_{i=1}^n (y_i - \hat{y}_i)^2$$

- Selection of a loss and cost functions depends on the kind of output we are targeting. For classification we use cross entropy cost function.

Back propagation

- **The goal of training is to minimize the cost function.**
- **This is achieved using back propagation algorithm.**
- **Back propagation algorithm allows the gradients to back propagate through the network and then these are used to adjust weights and biases to move the solution space towards the direction of reducing cost function.**

Gradient Descent

- The goal of all supervised machine learning algorithms is to best estimate a target function (f) that maps input data (x) onto output variables (y).
- Machine learning algorithms require a process of optimization to find the set of coefficients that result in the best estimate of the target function.
- Gradient descent method can be used to optimize coefficients and is used when the parameters cannot be calculated analytically (e.g. using linear algebra) and must be searched for by an optimization algorithm.

Gradient Descent Algorithm

- GD is an optimization algorithm to find the minimum of a function.
- Start with a random point function and move in the negative direction of the gradient of the function to reach the local/global minima.
- Gradient Descent Algorithm iteratively calculates the next point using gradient at the current position, then scales it (by a learning rate) and subtracts obtained value from the current position (makes a step).
- This process can be written as:

$$w_{i+1} = w_i - \alpha \frac{\partial L}{\partial w_i}$$

- Here α is called **learning rate** .It scales the gradient and thus controls the step size. The smaller learning rate the longer GD converges, or may reach maximum iteration before reaching the optimum point.
- If learning rate is too big the algorithm may not converge to the optimal point (jump around).

Gradient Descent Algorithm

- Gradient Descent method's steps are:
 1. choose a starting point (initialisation)
 2. calculate gradient at this point
 3. make a scaled step in the opposite(negative) direction to the gradient (objective: minimise)
 4. repeat points 2 and 3 until one of the criteria is met:
 - maximum number of iterations reached
 - step size is smaller than the tolerance.

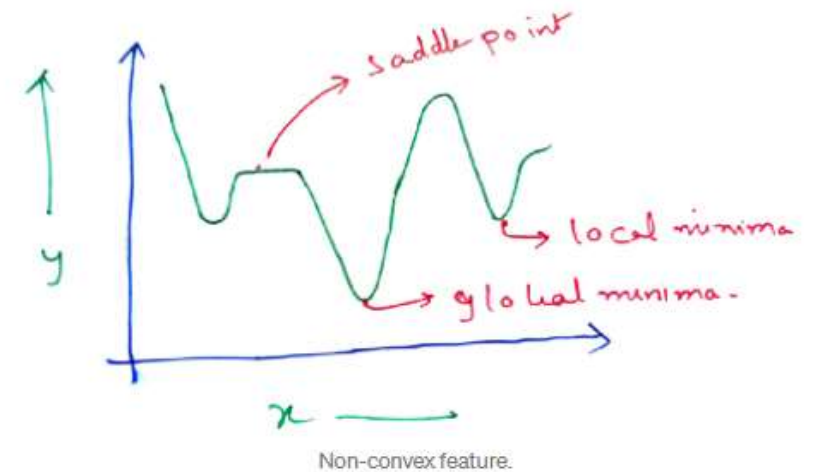
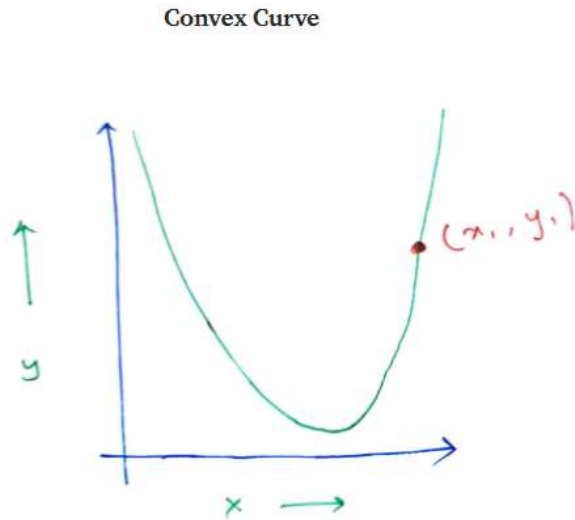
Gradient Descent Algorithm

- Gradient descent algorithm does not work for all functions. There are two specific requirements. A function has to be:
 - **differentiable**
 - **convex**
- **Differentiable** : If a function is differentiable, it has a derivative for each point in its domain.

Gradient Descent Algorithm

- To check mathematically if a univariate function is convex is to calculate the second derivative and check if its value is always bigger than 0.

$$\frac{d^2 f(x)}{dx^2} > 0$$



Gradient Descent Procedure

- The goal is to continue to try different values for the coefficients, evaluate their cost and select new coefficients that have a slightly better (lower) cost.
- Repeating this process enough times will lead to the values of the coefficients that result in the minimum cost

Gradient Descent

1. Batch Gradient Descent:

- This is a type of gradient descent which processes all the training examples for each iteration of gradient descent.
- But if the number of training examples is large, then batch gradient descent is computationally very expensive.
- If the number of training examples is large, then batch gradient descent is not preferred.
- Instead, we prefer to use stochastic gradient descent or mini-batch gradient descent.

Gradient Descent

2. Stochastic Gradient Descent:

- This is a type of gradient descent which processes 1 training example per iteration.
- The parameters are being updated even after one iteration in which only a single example has been processed.
- Hence this is quite faster than batch gradient descent.
- But again, when the number of training examples is large, even then it processes only one example which can be additional overhead for the system as the number of iterations will be quite large.

Gradient Descent

3. Mini Batch gradient descent:

This is a type of gradient descent which works faster than both batch gradient descent and stochastic gradient descent.

So even if the number of training examples is large, it is processed in batches of b training examples in one go. Thus, it works for larger training examples and that too with lesser number of iterations.

Sigmoid Neuron

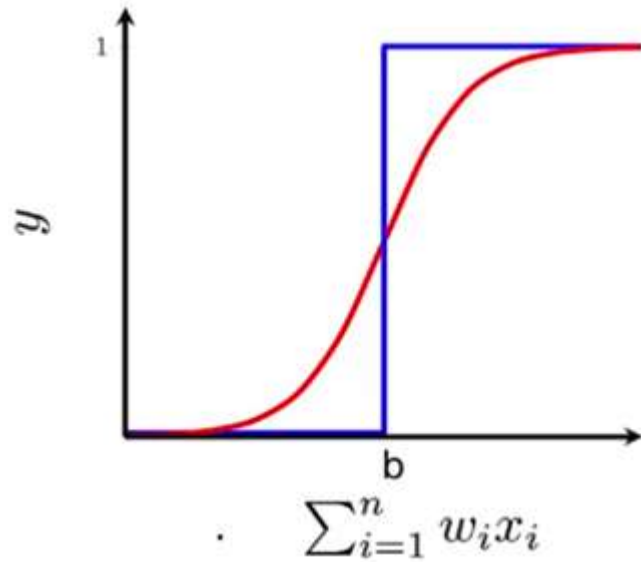
- Sigmoid neurons are similar to Perceptrons, but modified so that small changes in their weights and bias cause only a small change in their output.
- The sigmoid neuron has inputs, x_1, x_2, \dots .
- These inputs can also take on any values *between* 0 and 1. sigmoid neuron has weights for each input, w_1, w_2, \dots and an overall bias, b .
- In sigmoid neurons the output function (sigmoid activation function) is much smoother than the step function.
- In the sigmoid neuron, a small change in the input only causes a small change in the output as opposed to the stepped output.
- The most commonly used function is the logistic function.

$$y = \sum_{i=1}^n w_i x_i \geq b$$

Salary (in thousands)	Can buy a car?
80	1
20	0
65	1
15	0
30	0
49	0
51	1
87	1



Sigmoid Neuron



$$y = \frac{1}{1 + e^{-(w_0 + \sum_{i=1}^n w_i x_i)}}$$

- The output y is not binary but a real value between 0 and 1 which can be interpreted as a probability

- Can be used for both classification and regression task
- Has real valued inputs and outputs.
- Smooth at boundaries

Limitations of Perceptron

- Can learn only linearly separable data
- Hash decision boundary
- Has specific learning algorithm
- Only classification task can be done.