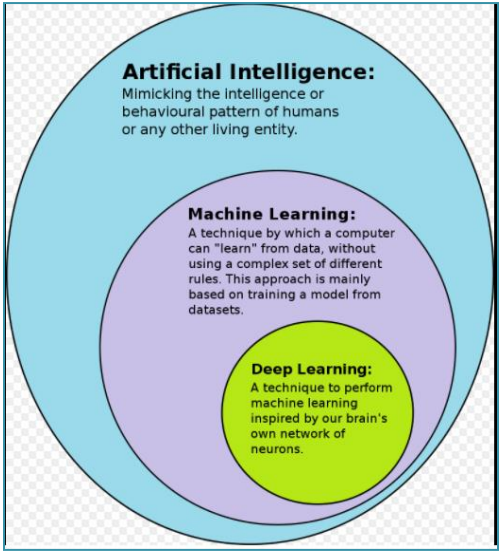
20MCA283
DEEP LEARNING

Selma Joseph, MCA LMCST

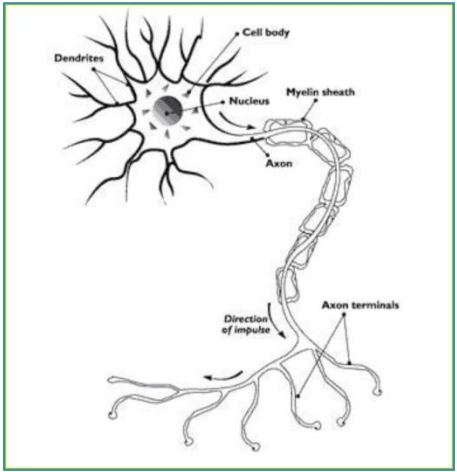

Artificial Intelligence:
Mimicking the intelligence or behavioural pattern of humans or any other living entity.
Machine Learning:
A technique by which a computer can "learn" from data, without using a complex set of different rules. This approach is mainly based on training a model from datasets.
Deep Learning:
A technique to perform machine learning inspired by our brain's own network of neurons.

Biological Neurons

Neurons and Perceptron

An Artificial Neural Network (ANN), popularly known as Neural Network is a computational model based on the structure and functions of biological neural networks. It is like an artificial human nervous system for receiving, processing, and transmitting information.

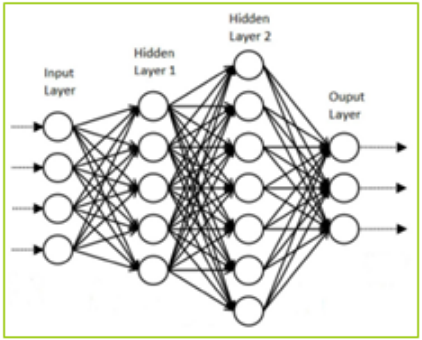
https://www.youtube.com/watch?v=qB2kXsF68uw&t=16s

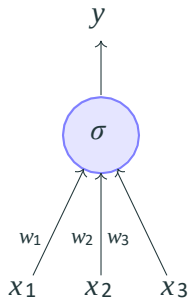


Neural networks

A neural networks consists of a number of neurons structured into different layers as in Figure where each circle represents a neuron. Basically, there are three different layers in a neural network:

1. **Input layer:** All the inputs are fed in the model through this layer.
2. **Hidden layers:** There can be more than one hidden layers which are used for processing the inputs received from the input layers.
3. **Output layer:** The data after processing is made available at the output layer.





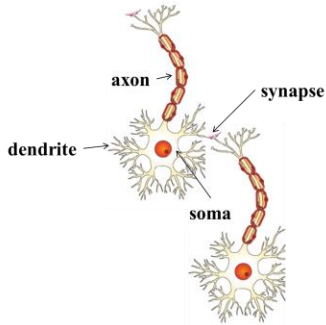
Artificial Neuron

The most fundamental unit of a deep neural network is called an *artificial neuron*

Why is it called a neuron ? Where does the inspiration come from ?

The inspiration comes from biology (more specifically, from the *brain*)

biological neurons = neural cells = neural processing units



Biological Neurons*

dendrite: receives signals from other neurons

synapse: point of connection to other neurons

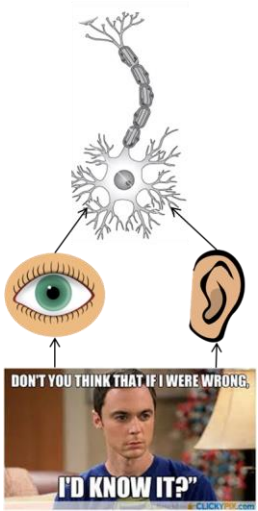
soma: processes the information

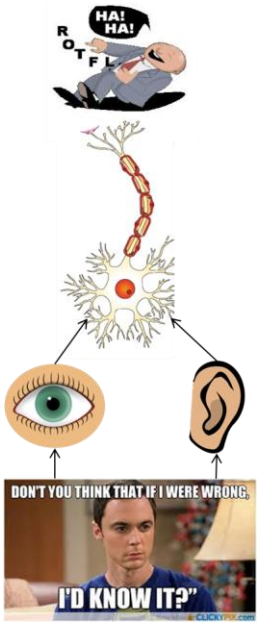
axon: transmits the output of this neuron

Let us see a very cartoonish illustration of how a neuron works

Our sense organs interact with the outside world

They relay information to the neurons



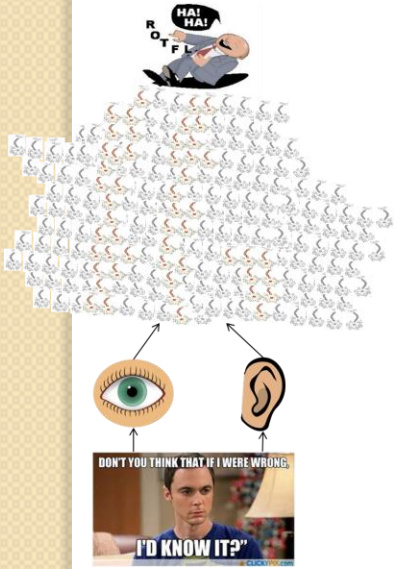


Let us see a cartoonish illustration of how a neuron works

Our sense organs interact with the outside world

They relay information to the neurons

The neurons (may) get activated and produces a response (laughter in this case)



Of course, in reality, it is not just a single neuron which does all this

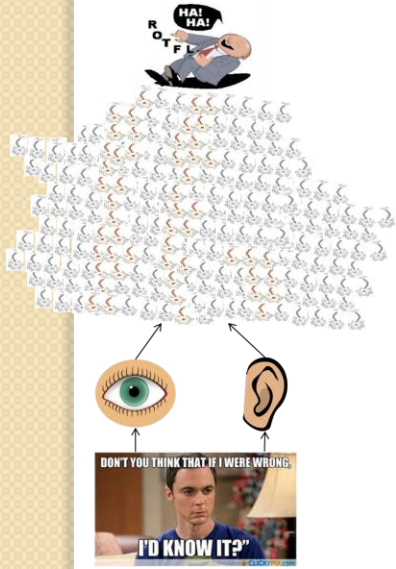
There is a massively parallel interconnected network of neurons

The sense organs relay information to the lowest layer of neurons

Some of these neurons may fire (in red) in response to this information and in turn relay information to other neurons they are connected to

These neurons may also fire (again, in red) and the process continues eventually resulting in a response (laughter in this case)

An average human brain has around 10^{11} (100 billion) neurons!



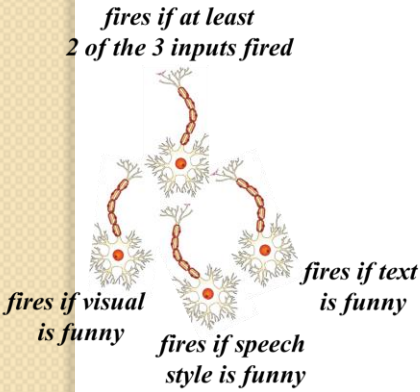
This massively parallel network also ensures that there is division of work

Each neuron may perform a certain role or respond to a certain stimulus

This massively parallel network also ensures that there is division of work

Each neuron may perform a certain role or respond to a certain stimulus

A simplified illustration

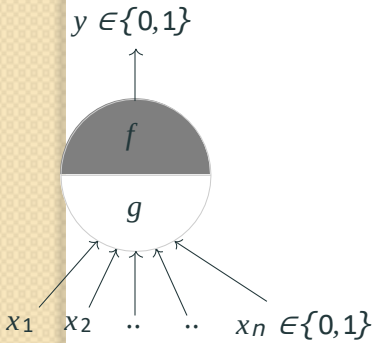


Asimplified illustration

This massively parallel network also ensures that there is division of work

Each neuron may perform a certain role or respond to a certain stimulus

Module 1.2: McCulloch Pitts Neuron



McCulloch (neuroscientist) and Pitts (logician) proposed a highly simplified computational model of the neuron (1943)

g aggregates the inputs and the function f takes a decision based on this aggregation

The inputs can be excitatory or inhibitory

$y = 0$ if any x_i is inhibitory, else

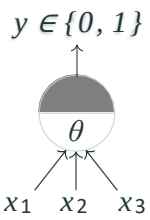
$$g(x_1, x_2, \ldots, x_n) = g(\mathbf{x}) = \sum_{i=1}^{n} x_i$$

$$y = f(g(\mathbf{x})) = 1 \quad if \quad g(\mathbf{x}) \geq \theta$$
$$= 0 \quad if \quad g(\mathbf{x}) < \theta$$

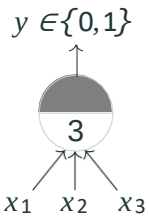
θ is called the thresholding parameter

This is called Thresholding Logic

Let us implement some boolean functions using this McCulloch Pitts (MP) neuron ...

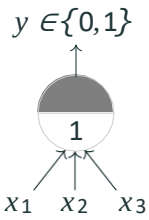


A McCulloch Pitts unit



AND function

G(x)=
X1=0
X2=0
X3=0
0>=1



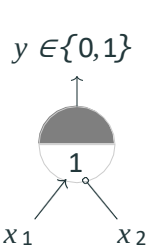
OR function

X1	X2	X3	
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

An AND function neuron would only fire when ALL the inputs are ON i.e., $\boldsymbol{g(x)} \geq 3$ here.

An OR function neuron would fire if ANY of the inputs is ON i.e., $\boldsymbol{g(x)} \geq 1$ here.

We have an inhibitory input i.e., ***x_2*** so whenever ***x_2*** is 1, the output will be 0.
The ***g(x)*** i.e., ***x_1*** + ***x_2*** would be ≥ 1 in only 3 cases:
Case 1: when ***x_1*** is 1 and ***x_2*** is 0 X1=1 X2=0
Case 2: when ***x_1*** is 1 and ***x_2*** is 1 X1=1 X2=1
Case 3: when ***x_1*** is 0 and ***x_2*** is 1 X1=0 X2=1

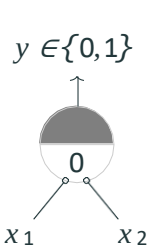


x_1 AND $!x_2$ *

x1.x2'

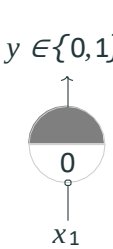
X1	x2	x1.x2'
0	0	0
0	1	0
1	0	1
1	1	0

G(x)= x1+x2
1+0=1



NOR function (X+Y)'

0	0	1
0	1	0
1	0	0
1	1	0



NOT function

*circle at the end indicates inhibitory input: if any inhibitory input is 1 the output will be 0

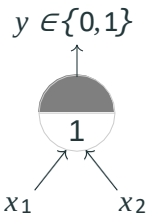
X1	0	1

- Can any boolean function be represented using a McCulloch Pitts unit ?

Can any boolean function be represented using a McCulloch Pitts unit ?

The geometric interpretation of a MP unit

...

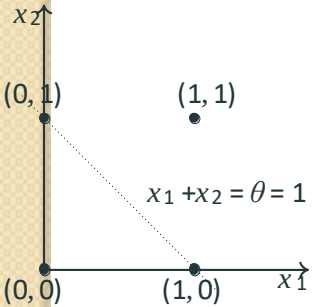


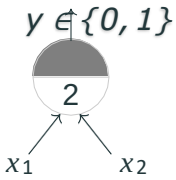
A single MP neuron splits the input points (4 points for 2 binary inputs) into two halves

All inputs which produce an output 0 will be on one side ($\sum_{i=1}^{n} x_i < \theta$) of the line and all inputs which produce an output 1 will lie on the other side ($\sum_{i=1}^{n} x_i \geq \theta$) of this line

OR function

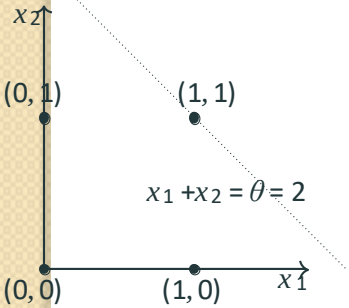
$x_1 + x_2 = \sum_{i=1}^{2} x_i \geq 1$

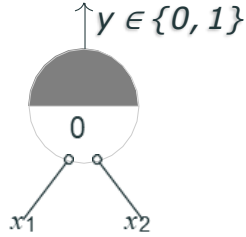




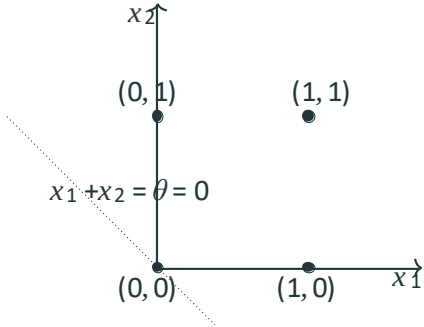
AND function

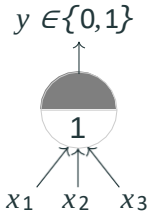
$$x_1 + x_2 = \sum_{i=1}^{2} x_i \geq 2$$





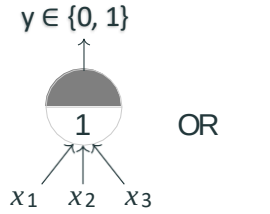
Tautology (always ON)

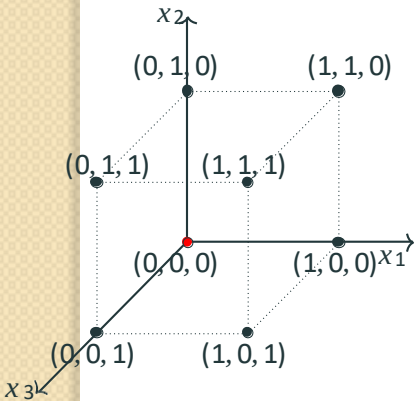




OR

What if we have more than 2 inputs?



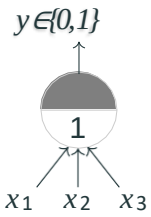


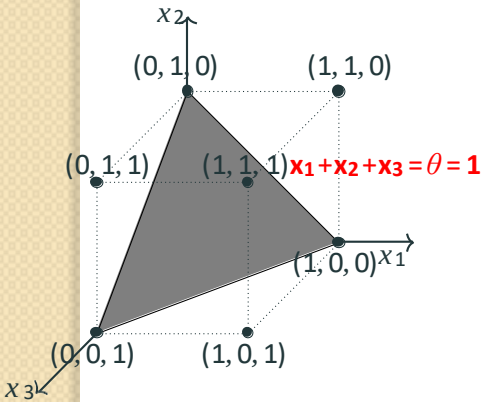
What if we have more than 2 inputs?

A 3 input OR function M-P unit has the possible inputs are …8 points — (0,0,0), (0,0,1), (0,1,0), (1,0,0), (1,0,1) etc.

Then, instead of a line we will have a plane.

For the OR function, we want a plane such that the point (0,0,0) lies on one side and the remaining 7 points lie on the other side of the plane





What if we have more than 2 inputs?

Well, instead of a line we will have a plane

For the OR function, we want a plane such that the point (0,0,0) lies on one side and the remaining 7 points lie on the other side of the plane

- A single McCulloch Pitts Neuron can be used to represent boolean functions which are linearly separable
- Linear separability (for boolean functions) : There exists a line (plane) such that all inputs which produce a 1 lie on one side of the line (plane) and all inputs which produce a 0 lie on other side of the line (plane)

Deep Learning

Contents

- Characteristics of ANN
- Back propagation
- Gradient Descent

Artificial Neural Networks

- ANN is based on the collection of connected units called Artificial Neurons.
- Each connection between artificial neurons can transmit a signal from one to another.
- The artificial neuron that receives the signal, process it ,then transmit to the neighboring neurons.

Characteristics of ANN

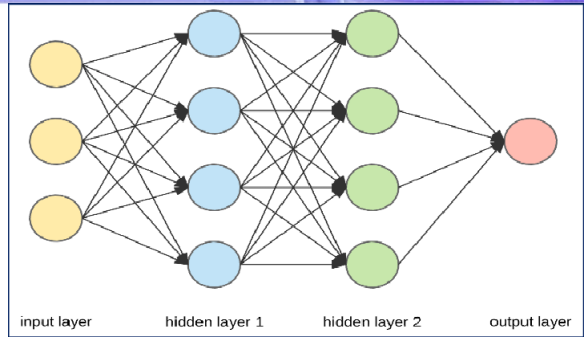
1. **Activation function** : This function defines how a neuron's combined inputs are transferred into a single output signal.
2. **The network topology**
   - It describes the number of neurons in the model as well as the number of layers and the way they are connected.
   - It determines the complexity of the task.

Characteristics of ANN

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

a) The number of layers.

- ✓ Input nodes(first layer) : The nodes that receive unprocessed signals directly from the input data.
- ✓ Hidden nodes(second layer) : The nodes that processes the signals from the input node prior to reaching output nodes.
- ✓ Output nodes : Those nodes that generate the final predicted values.



Characteristics of ANN

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

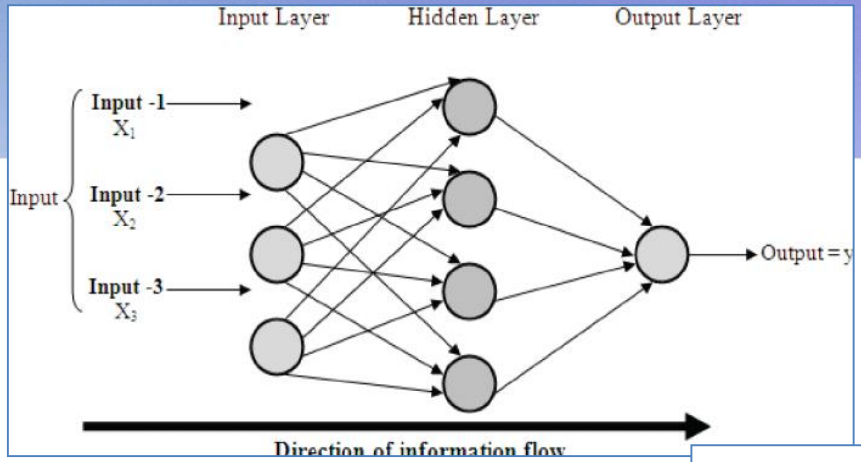
- Feed forward network : Network in which input signal is fed continuously in one direction from connection to connection until it reaches the output layer called feed forward network.
- Feedback network : Network which allows signals to travel in both directions using loops are called feedback network/recurrent network.

Characteristics of ANN

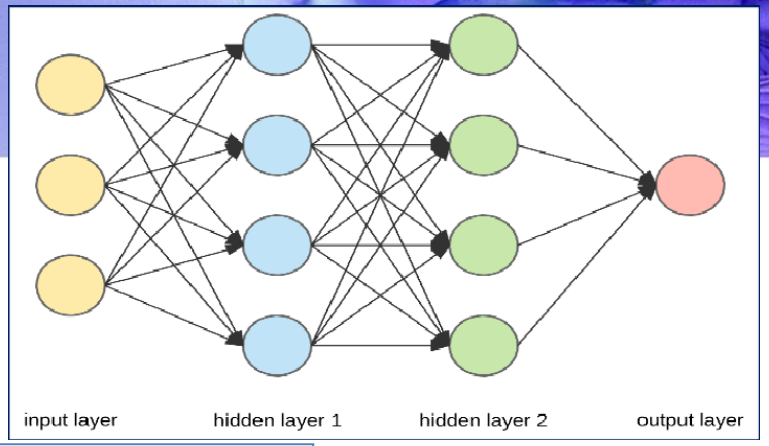
c) The number of nodes within each layer of the network

- The number of features in the input data => Number of input nodes.
- Number of output nodes => Number of outcomes to be modeled.
- Number of hidden nodes => User to decide prior to training the model. The appropriate number depends on number of input nodes, amount of training data, amount of noisy data, complexity of learning task etc.

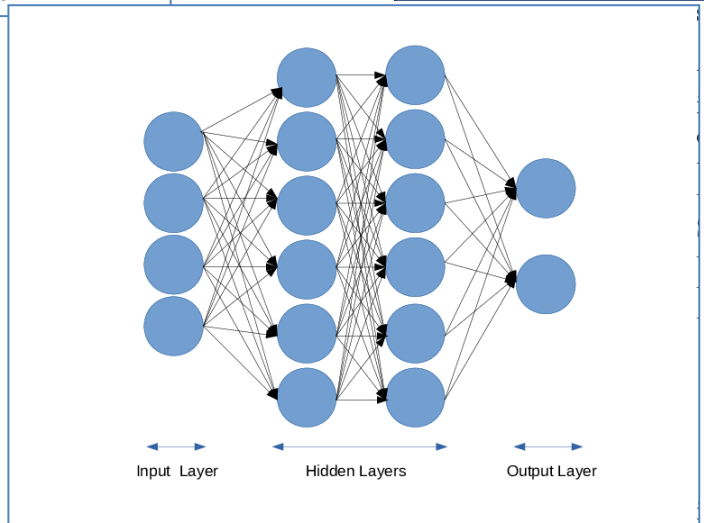
Characteristics of ANN



Network with 1 hidden layer



Network with 2 hidden layer



Network with 2 hidden layer and 2 output nodes

Characteristics of ANN

3) **Training algorithm**

Algorithms specifies how connection weights are set in order. Mainly 2 algorithms for learning a single perceptron.

1. Perceptron rule : used when training dataset is linearly separable.
2. Delta rule : used when training data set is not linearly separable.

Characteristics of ANN

4) Cost Function:

- It is a function that measures the performance of a model for given data. Also called the loss function/ objective function/scoring function/error function.
- Let *y* be the output variable. $y_1, y_2, \ldots y_n$ are actual values of *y*. Also $\hat{y}_1, \hat{y}_2, \ldots . \hat{y}_n$ are predicted values
- SSE : Sum of squares of differences between predicted and actual values of *y* denoted by

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

($i=1$: test set; y_i: predicted vaue; $\hat{y}_i$: actual value)

- MSE : Mean of the sum of squares of the differences between the predicted and actual values of *y denoted*

$$MSE = \frac{1}{n} \Sigma \underbrace{\left(y - \widehat{y} \right)^2}_{\text{The square of the difference between actual and predicted}}$$

Back propagation

- Back propagation is an algorithm for supervised learning of artificial neural networks using gradient descent.
- Given an artificial neural network and an error function, the method calculates the gradient of the error function with respect to the neural network weights.

Back propagation

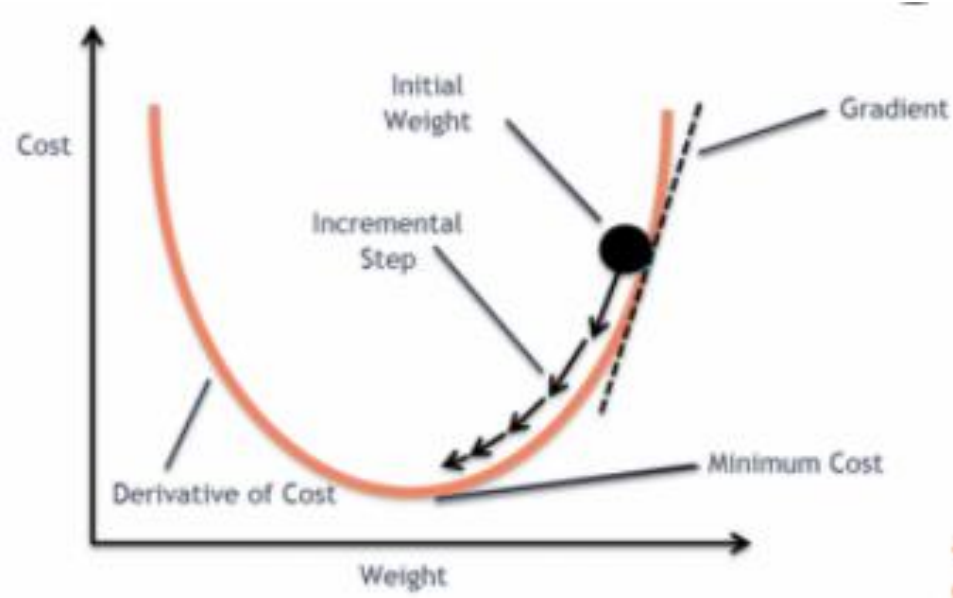
1) Initially the weights are assigned at random.
2) The algorithm iterates through main cycles of to process called
    a) Forward phase
    b) Backward phase

A Forward Phase : Neurons are activated in sequence from the input layer to the output layers. Applying neuron's weights and activation function along the way an output signal is produced from the final layer.

A Backward phase : Output signal combined with the true target value. Difference in results i.e.. error is propagated backward in the network to modify the connection weights

Back propagation

3) The technique used to determine how much a weight can be changed is known as Gradient Descent method.



- At any point on the surface of the curve the gradient suggest how steeply the error will be reduced or increased for a change in the weight.
- Algorithm helps to change the weights that result in the greatest reduction in error

Gradient Descent

- Imagine a blindfolded man who wants to climb to the top of a hill with the fewest steps along the way as possible. He might start climbing the hill by taking really big steps in the steepest direction, which he can do as long as he is not close to the top. As he comes closer to the top, however, his steps will get smaller and smaller to avoid overshooting it. This process can be described mathematically using the gradient. Instead of climbing up a hill, think of gradient descent as hiking down to the bottom of a valley.

WHAT IS A GRADIENT?

In machine learning, a gradient is a derivative of a function that has more than one input variable known as the slope of a function.In mathematical terms, the gradient simply measures the change in all weights with regard to the change in error.

Gradient Descent

- Gradient Descent is an iterative optimization algorithm for finding the local minimum of a function.
- To find the local minimum of a function using Gradient Descent, take steps proportional to the negative of the gradient(move away from the gradient) of the function at the current point.
- It takes steps proportional to the positive of the gradient (moving towards the gradient), we will approach a local maximum of the function. This procedure is called gradient ascent.

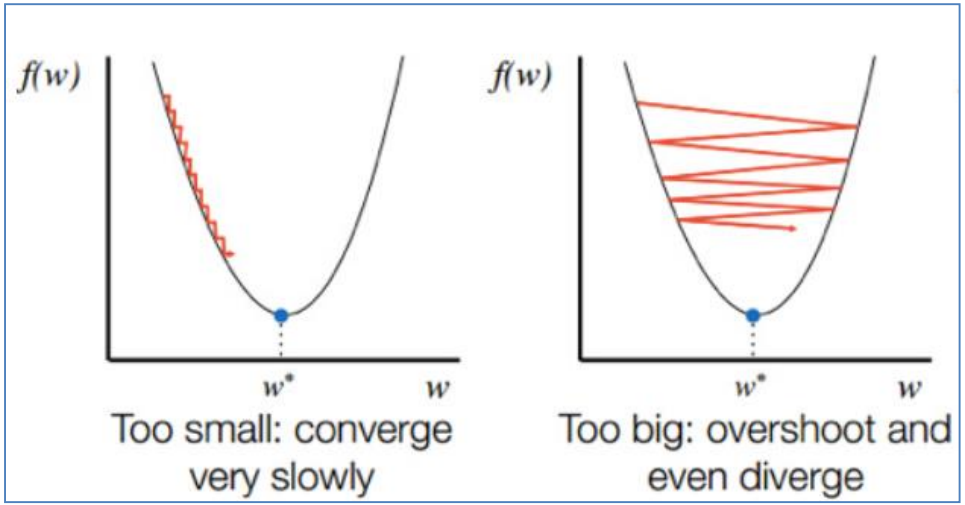
Gradient Descent

- The goal of the gradient Descent algorithm is to minimize the given function.

  1) Compute the gradient(slope), the first order derivative of the function at that point.

  2) Make a step in the direction opposite to the direction of the gradient.
- We have now in the direction to move in, next is to decide the size of the step.
- Learning rate (η)– A tuning parameter in the optimization process and it decides the length of the steps.

Gradient Descent



- If the learning rate is too high it might overshoot the minima and keep bouncing without reaching the minima.
- If the learning rate is too small the training might turn out to be too long.
- Choose an optimal learning rate so that model converges to the minimum.

Types of gradient Descent:

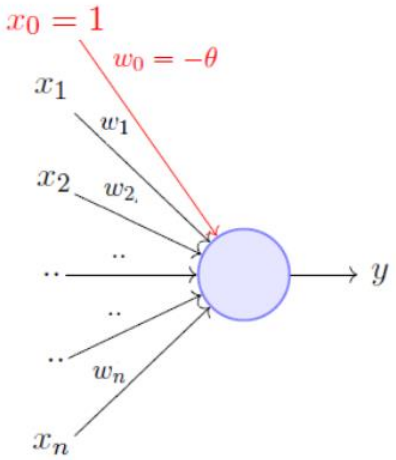
- **Batch Gradient Descent:** This is a type of gradient descent which processes all the training examples for each iteration of gradient descent. But if the number of training examples is large, then batch gradient descent is computationally very expensive and not preferred. Instead, we prefer to use stochastic gradient descent or mini-batch gradient descent.
- **Stochastic Gradient Descent:** This is a type of gradient descent which processes 1 training example per iteration. Hence, 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.
- Mini Batch gradient descent: This is a type of gradient descent which works faster than both batch gradient descent and stochastic gradient descent. Here b examples where b<m are processed per iteration. 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.

Types of gradient Descent:

Batch Gradient Descent	Stochastic Gradient Descent (SGD)	Mini batch gradient descent
Use **all** training samples for one forward pass and then adjust weights	Use **one** (randomly picked) sample for a forward pass and then adjust weights	Use **a batch of** (randomly picked) samples for a forward pass and then adjust weights

Deep Learning

Perceptron



A more accepted convention,

$$y = 1 \quad if \sum_{i=0}^{n} w_i * x_i \geq 0$$

$$= 0 \quad if \sum_{i=0}^{n} w_i * x_i < 0$$

$$where, \quad x_0 = 1 \quad and \quad w_0 = -\theta$$

A single perceptron can only be used to implement **linearly separable** functions. It takes both real and Boolean inputs and associates a set of **weights** to them, along with a **bias.**

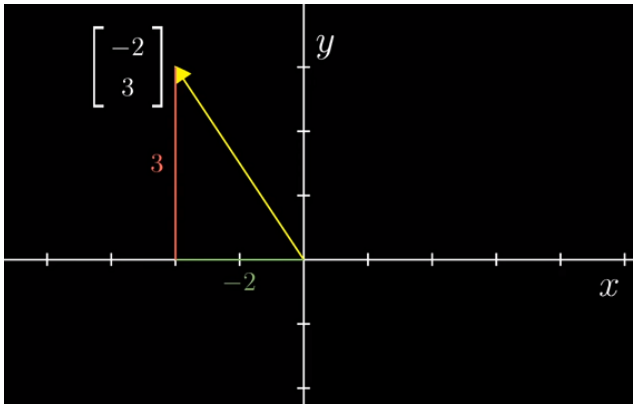
Basics Of Linear Algebra

- **Vector**

A vector is anything that sits anywhere in space, has a magnitude and a direction. For CS guy, a vector is just a data structure used to store some data

Vector Representations

A 2-dimensional vector can be represented on a 2D plane as



Dot Product Of Two Vectors

The dot product of these vectors (w.x) could be computed as follows:

$$\mathbf{w} = [w_0, w_1, w_2, ..., w_n]$$

$$\mathbf{x} = [1, x_1, x_2, ..., x_n]$$

$$\mathbf{w} \cdot \mathbf{x} = \mathbf{w}^{\mathrm{T}}\mathbf{x} = \sum_{i=0}^{n} w_i * x_i$$

The decision boundary line which a perceptron gives out that separates positive examples from the negative ones is really just w . x = 0.

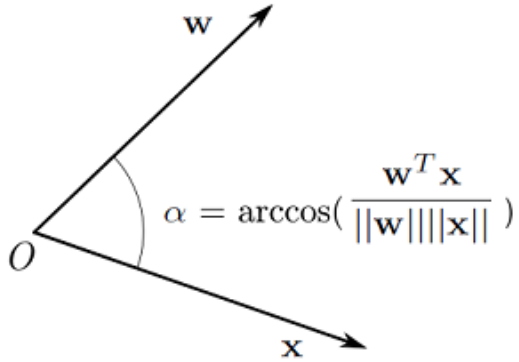
Basics Of Linear Algebra

Angle Between Two Vectors

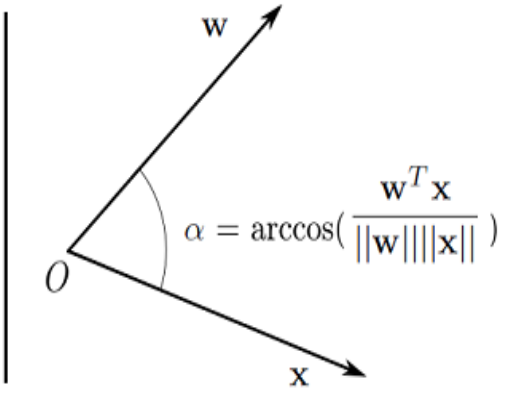
The angle between the vectors and their individual magnitudes

$$\mathbf{w}^T\mathbf{x} = ||\mathbf{w}||||\mathbf{x}|| cos\alpha$$

$$cos\alpha = \frac{\mathbf{w}^T\mathbf{x}}{||\mathbf{w}||||\mathbf{x}||}$$



$$cos\alpha = \frac{\mathbf{w}^T\mathbf{x}}{||\mathbf{w}||||\mathbf{x}||}$$



Perceptron Learning Algorithm

Algorithm: Perceptron Learning Algorithm

$P \leftarrow$ *inputs with label* 1;
$N \leftarrow$ *inputs with label* 0;
Initialize **w** randomly;
while !*convergence* **do**
  Pick random $\mathbf{x} \in P \cup N$;
  if $\mathbf{x} \in P$ *and* $\mathbf{w}.\mathbf{x} < 0$ **then**
    $\mathbf{w} = \mathbf{w} + \mathbf{x}$;
  end
  if $\mathbf{x} \in N$ *and* $\mathbf{w}.\mathbf{x} \geq 0$ **then**
    $\mathbf{w} = \mathbf{w} - \mathbf{x}$;
  end
end
//the algorithm converges when all the inputs are classified correctly

We initialize w with some random vector. We then iterate over all the examples in the data, (P U N) both positive and negative examples. Now if an input x belongs to P, ideally what should the dot product w.x be? It should be greater than or equal to 0 because that's the only thing what our perceptron wants at the end of the day so let's give it that. And if x belongs to N, the dot product MUST be less than 0. So if you look at the if conditions in the while loop:

Perceptron Learning Algorithm

Why would this work?

When x belongs to P, we want w.x > 0, basic perceptron rule. i.e when x belongs to P, the angle between w and x should be less than 90 degrees? The angle between w and x should be less than 90 because the cosine of the angle is proportional to the dot product. So whatever the w vector may be, as long as it makes an angle less than 90 degrees with the positive example data vectors (x E P) and an angle more than 90 degrees with the negative example data vectors (x E N).

So whatever the w vector may be, as long as it makes an angle less than 90 degrees with the positive example data vectors (x E P) and an angle more than 90 degrees with the negative example data vectors (x E N)

$$cos\alpha = \frac{\mathbf{w}^T\mathbf{x}}{||\mathbf{w}||||\mathbf{x}||} \quad \Big| \quad cos\alpha \propto \mathbf{w}^T\mathbf{x}$$

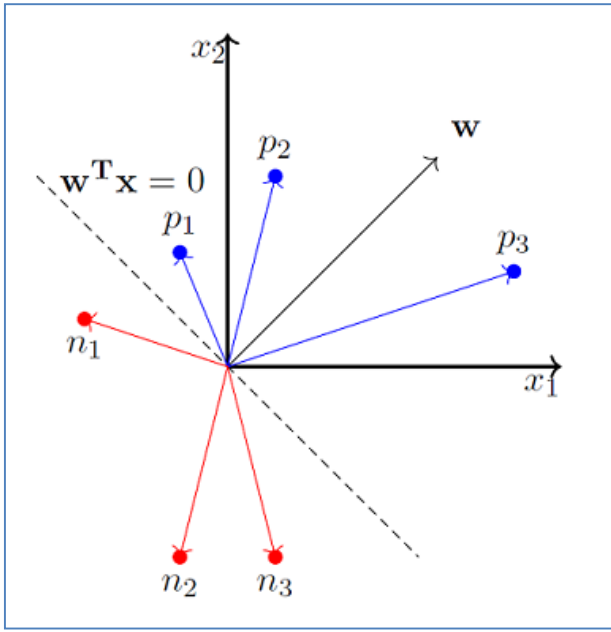
$$\text{So if } \mathbf{w}^T\mathbf{x} > 0 \Rightarrow cos\alpha > 0 \Rightarrow \alpha < 90$$

$$\text{Similarly, if } \mathbf{w}^T\mathbf{x} < 0 \Rightarrow cos\alpha < 0 \Rightarrow \alpha > 90$$

Perceptron Learning Algorithm

Why would this work?

(α_{new}) when $\mathbf{w_{new}} = \mathbf{w} + \mathbf{x}$	(α_{new}) when $\mathbf{w_{new}} = \mathbf{w} - \mathbf{x}$
$cos(\alpha_{new}) \propto \mathbf{w_{new}}^T \mathbf{x}$	$cos(\alpha_{new}) \propto \mathbf{w_{new}}^T \mathbf{x}$
$\propto (\mathbf{w} + \mathbf{x})^T \mathbf{x}$	$\propto (\mathbf{w} - \mathbf{x})^T \mathbf{x}$
$\propto \mathbf{w}^T \mathbf{x} + \mathbf{x}^T \mathbf{x}$	$\propto \mathbf{w}^T \mathbf{x} - \mathbf{x}^T \mathbf{x}$
$\propto cos\alpha + \mathbf{x}^T \mathbf{x}$	$\propto cos\alpha - \mathbf{x}^T \mathbf{x}$
$cos(\alpha_{new}) > cos\alpha$	$cos(\alpha_{new}) < cos\alpha$



Gradient Descent Algorithm

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). It subtracts the value because we want to minimise the function (to maximise it would be adding). This process can be written as:

- There's an important parameter η which scales the gradient and thus controls the step size. In machine learning, it is called learning rate and have a strong influence on performance.
- 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) or even to diverge completely.

Gradient Descent Algorithm

Gradient Descent method's steps are:

1. Choose a starting point (initialization)
2. Calculate gradient at this point
3. Make a scaled step in the opposite direction to the gradient

   (objective: minimize)

4. Repeat points 2 and 3 until one of the criteria is met:
5. Maximum number of iterations reached
6. Step size is smaller than the tolerance.

This function takes 5 parameters:

1. **starting point** - In practice, it is often a random initialisation
2. **gradient function** - has to be specified before-hand
3. **learning rate** - scaling factor for step sizes
4. maximum number of iterations
5. tolerance to conditionally stop the algorithm (a default value is 0.01)

Gradient Descent Algorithm

The objective learning algorithm is to determine the best possible values for the parameters(w & b), such that the overall loss of the model is minimized as such as possible.

Initialize w, b;
Iterate over data;
Compute $\hat{y}$
Compute L(w,b)
$W_{t+1}=W_t - \eta\Delta W_t$
$b_{t+1}=b_t - \eta\Delta b_t$

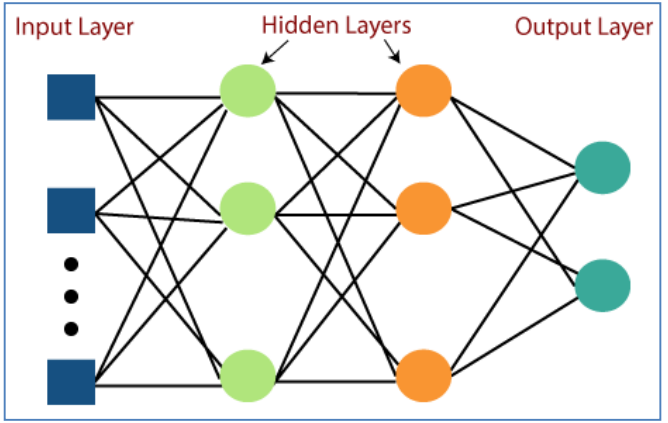
where $\Delta W_t = \frac{\partial L(W,b)}{\partial w}$ at $W=W_t, b=b_t$

$\Delta b_t = \frac{\partial L(W,b)}{\partial b}$ at $W=W_t, b=b_t$

Multilayer perceptron/Network of perceptron

- Multi-Layer perceptron defines the most complex architecture of artificial neural networks. It is substantially formed from multiple layers of the perceptron. MLP networks are used for supervised learning format. A typical learning algorithm for MLP networks is also called **back propagation's algorithm**.

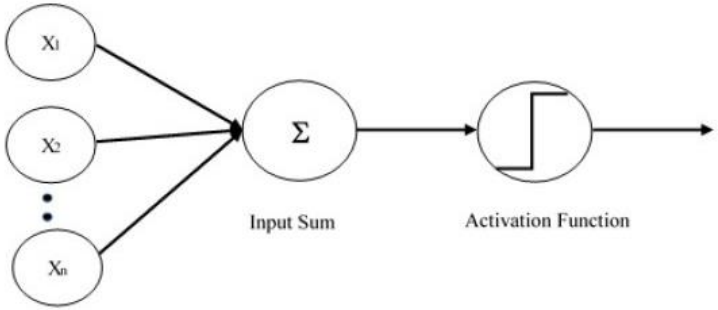
A multilayer perceptron (MLP) is a feed forward artificial neural network that generates a set of outputs from a set of inputs. An MLP is characterized by several layers of input nodes connected as a directed graph between the input nodes connected as a directed graph between the input and output layers. MLP uses backpropagation for training the network.



Single Layer vs Multilayer Perceptron

Single layer perceptron is a simple Neural Network which contains only one layer. The single layer computation of perceptron is the calculation of sum of input vector with the value multiplied by corresponding vector weight. The displayed output value will be the input of an activation function.

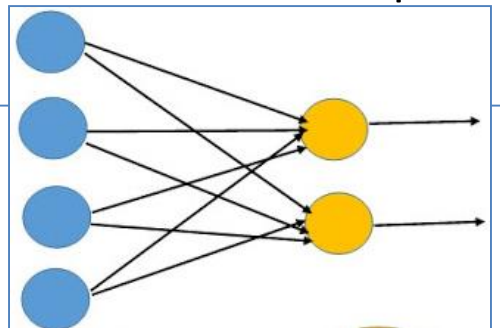
The perceptron consists of 4 parts.
Input values or One input layer
Weights and Bias
Net sum
Activation Function



Single Layer vs Multilayer Perceptron

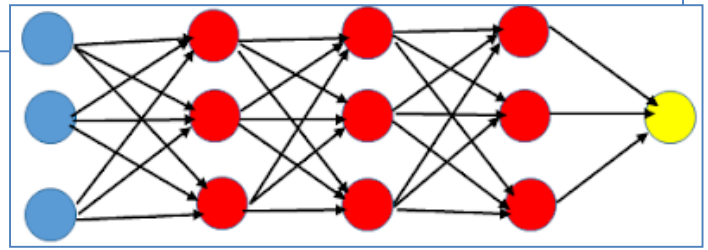
- **Single-layer Perceptron**

Single Layer Perceptron has just two layers of input and output. It only has single layer hence the name single layer perceptron. It does not contain Hidden Layers as that of Multilayer perceptron. Input nodes are connected fully to a node or multiple nodes in the next layer. A node in the next layer takes a weighted sum of all its inputs



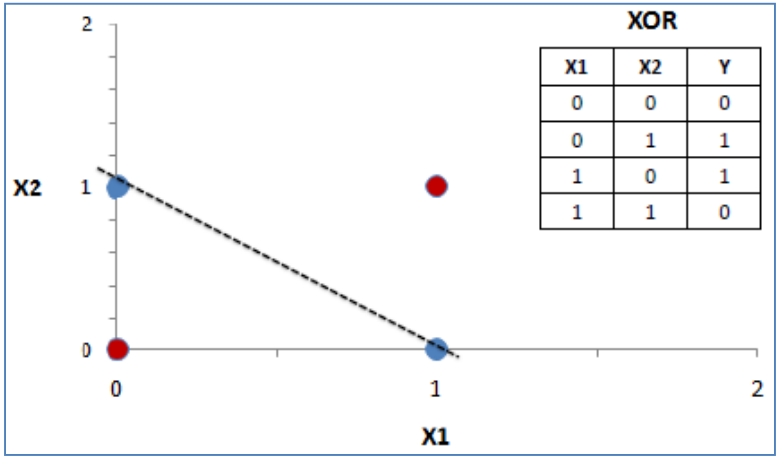
Multi-Layer Perceptron (MLP)

A multilayer perceptron is a type of feed-forward artificial neural network that generates a set of outputs from a set of inputs. An MLP is a neural network connecting multiple layers in a directed graph, which means that the signal path through the nodes only goes one way. The MLP network consists of input, output, and hidden layers. Each hidden layer consists of numerous perceptron's which are called hidden layers or hidden unit.



Learning XOR Gate

An XOR (exclusive OR gate) is a digital logic gate that gives a true output only when both its inputs differ from each other. The truth table for an XOR gate is shown below:



XOR

X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	0

The goal of the neural network is to classify the input patterns according to the above truth table. If the input patterns are plotted according to their outputs, it is seen that these points are not linearly separable. A single neuron is not able to solve an XOR problem

Solving XOR problem Example

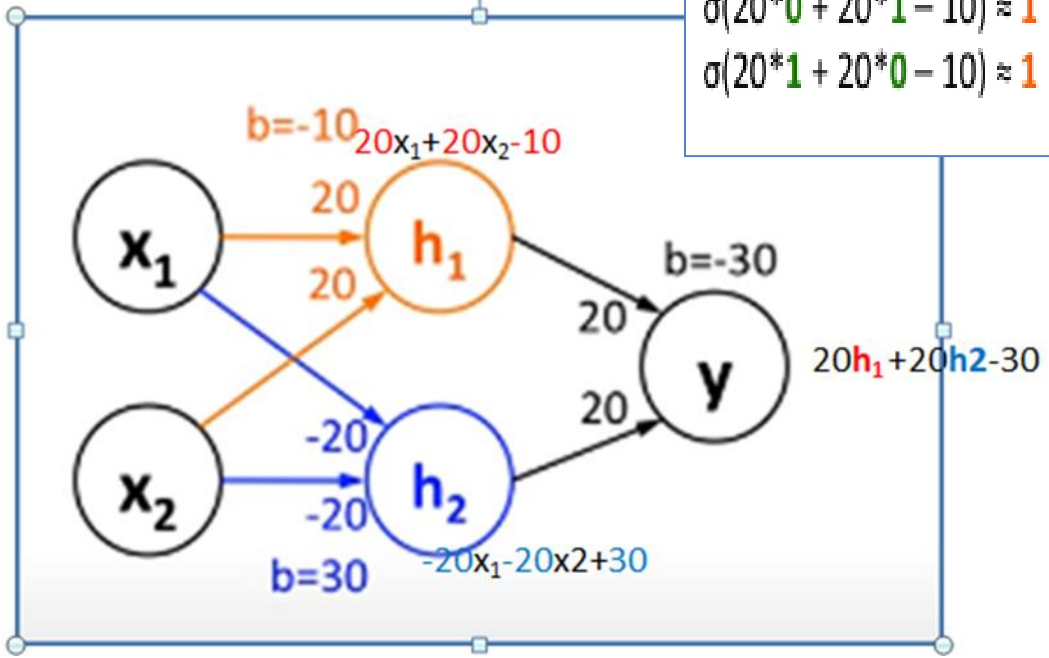
Consider the following network with 2 inputs and 1 hidden layer.

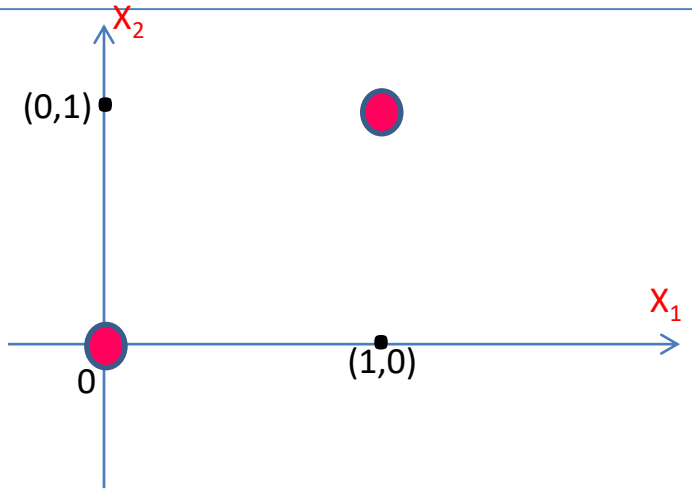
$\sigma(20*0 + 20*0 - 10) \approx 0$ $\sigma(-20*0 - 20*0 + 30) \approx 1$ $\sigma(20*0 + 20*1 - 30) \approx 0$

$\sigma(20*1 + 20*1 - 10) \approx 1$ $\sigma(-20*1 - 20*1 + 30) \approx 0$ $\sigma(20*1 + 20*0 - 30) \approx 0$

$\sigma(20*0 + 20*1 - 10) \approx 1$ $\sigma(-20*0 - 20*1 + 30) \approx 1$ $\sigma(20*1 + 20*1 - 30) \approx 1$

$\sigma(20*1 + 20*0 - 10) \approx 1$ $\sigma(-20*1 - 20*0 + 30) \approx 1$ $\sigma(20*1 + 20*1 - 30) \approx 1$

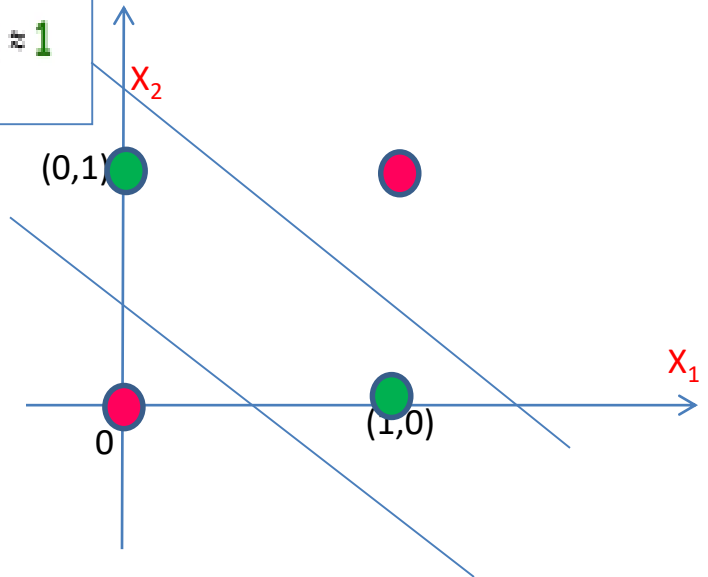




Solving XOR problem
Example

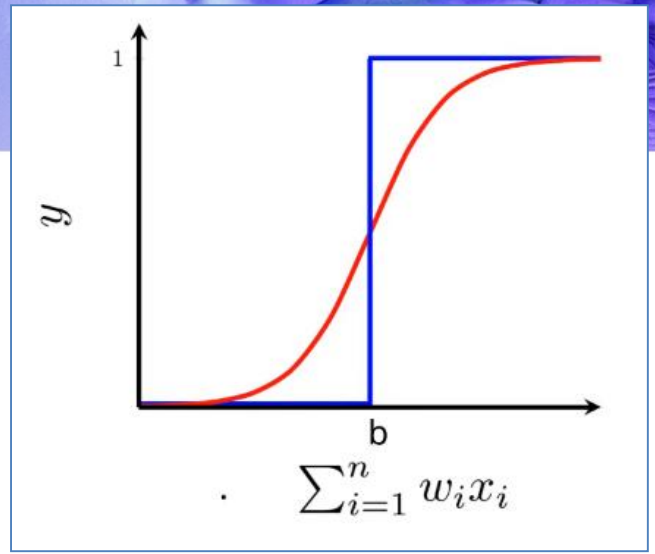
$\sigma(20*0 + 20*0 - 10) \approx 0$	$\sigma(-20*0 - 20*0 + 30) \approx 1$	$\sigma(20*0 + 20*1 - 30) \approx 0$
$\sigma(20*1 + 20*1 - 10) \approx 1$	$\sigma(-20*1 - 20*1 + 30) \approx 0$	$\sigma(20*1 + 20*0 - 30) \approx 0$
$\sigma(20*0 + 20*1 - 10) \approx 1$	$\sigma(-20*0 - 20*1 + 30) \approx 1$	$\sigma(20*1 + 20*1 - 30) \approx 1$
$\sigma(20*1 + 20*0 - 10) \approx 1$	$\sigma(-20*1 - 20*0 + 30) \approx 1$	$\sigma(20*1 + 20*1 - 30) \approx 1$

MLP solves the XOR problem efficiently by visualizing the data points in multi-dimensions and thus constructing an n-variable equation to fit in the output values(hyperplane).



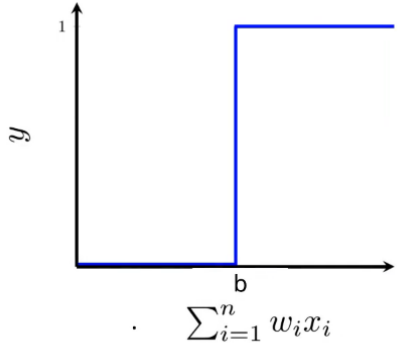
Sigmoid Neuron

- The building block of the deep neural networks is called the sigmoid neuron. Sigmoid neurons are similar to perceptrons, but they are slightly modified such that the output from the sigmoid neuron is much smoother than the step functional output from perceptron. the Sigmoid family of functions in Deep Learning of which many of the functions are S-shaped. One such function is the logistic function(it is one smooth continuous function) and this function is defined by the below equation:



Sigmoid neuron function result

Step function result



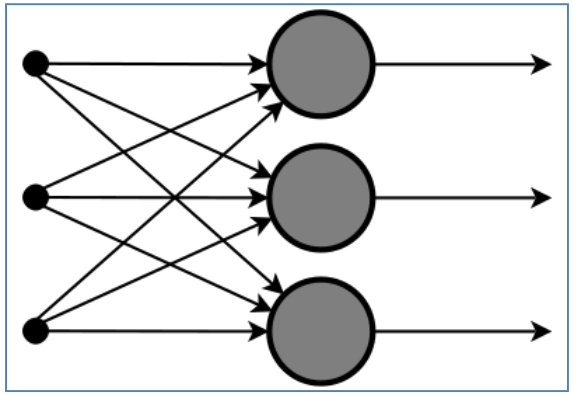
$$y = \frac{1}{1 + e^{-(wx+b)}}$$

Feed forward Neural Networks.

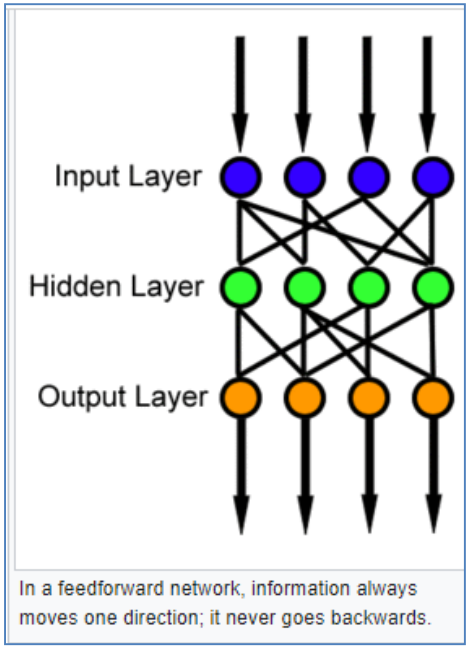
- "The process of receiving an input to produce some kind of output to make some kind of prediction is known as Feed Forward." Feed Forward neural network is the core of many other important neural networks such as convolution neural network. In the feed-forward neural network, there are not any feedback loops or connections in the network. Here is simply an input layer, a hidden layer, and an output layer. There can be multiple hidden layers which depend on what kind of data you are dealing with. The number of hidden layers is known as the depth of the neural network. The deep neural network can learn from more functions. Input layer first provides the neural network with data and the output layer then make predictions on that data which is based on a series of functions.
- This neural network is one of the simplest forms of ANN, where the data or the input travels in one direction. The data passes through the input nodes and exit on the output nodes. This neural network may or may not have the hidden layers. In simple words, it has a front propagated wave and no backpropagation by using a classifying activation function usually.

Feed forward Neural Networks.

- Below is a Single layer feed-forward network. Here, the sum of the products of inputs and weights are calculated and fed to the output. The output is considered if it is above a certain value i.e threshold and the neuron fires with an activated output and if it does not fire, the deactivated value is emitted.



Application of Feedforward neural networks are found in computer vision and speech recognition where classifying the target classes is complicated.

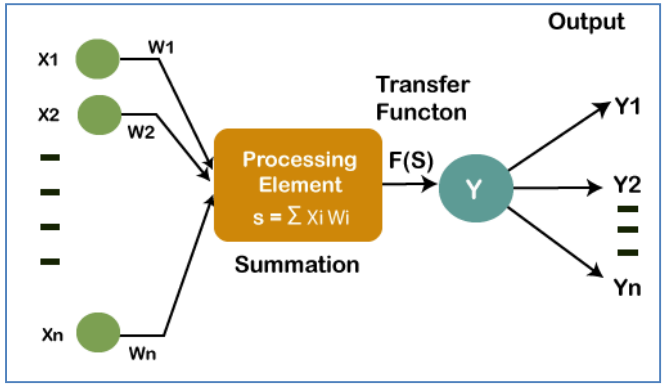


In a feedforward network, information always moves one direction; it never goes backwards.

Working of Artificial Neural Networks-Summary

A perceptron can be defined as a neural network with a single layer that classifies the linear data. It further constitutes four major components, which are as follows;

- Inputs
- Weights and Bias
- Summation Functions
- Activation or transformation function



Working of Artificial Neural Networks-Summary

- The main logic behind the concept of Perceptron is as follows:
- The inputs (x) are fed into the input layer, which undergoes multiplication with the allotted weights (w) followed by experiencing addition in order to form weighted sums. Then these inputs weighted sums with their corresponding weights are executed on the pertinent activation function.
- Weights and Bias
- As and when the input variable is fed into the network, a random value is given as a weight of that particular input, such that each individual weight represents the importance of that input in order to make correct predictions of the result.
- However, bias helps in the adjustment of the curve of activation function so as to accomplish a precise output.

Working of Artificial Neural Networks-Summary

- **Summation Function**

After the weights are assigned to the input, it then computes the product of each input and weights. Then the weighted sum is calculated by the summation function in which all of the products are added.

- **Activation Function**

The main objective of the activation function is to perform a mapping of a weighted sum upon the output. The transformation function comprises of activation functions such as tanh, ReLU, sigmoid, etc.

The activation function is categorized into two main parts:

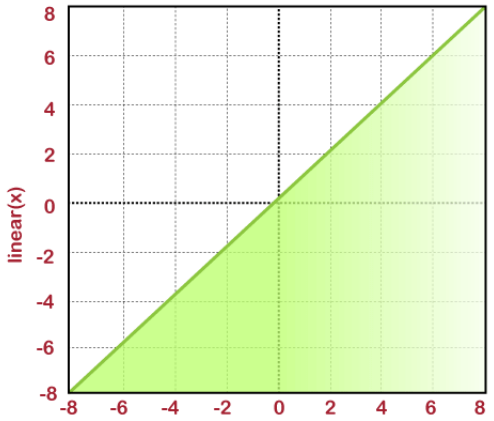
✓ Linear Activation Function

✓ Non-Linear Activation Function

Working of Artificial Neural Networks-Summary

- **Linear Activation Function**

In the linear activation function, the output of functions is not restricted in between any range. Its range is specified from -infinity to infinity. For each individual neuron, the inputs get multiplied with the weight of each respective neuron, which in turn leads to the creation of output signal proportional to the input.



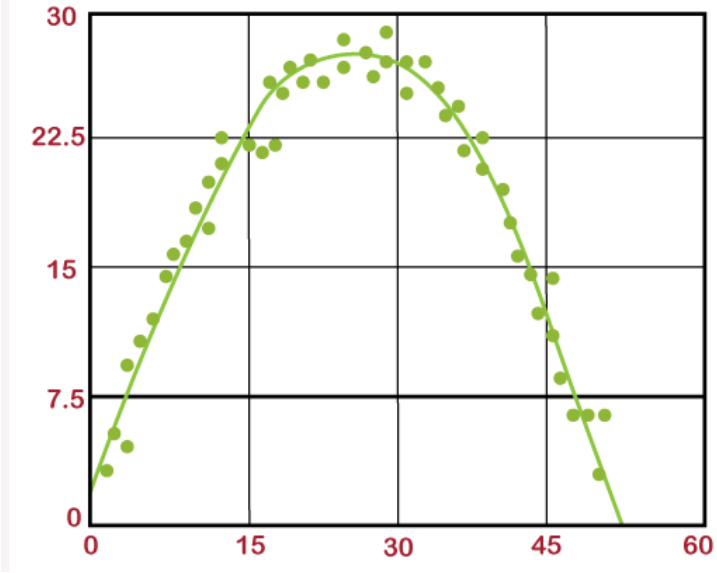
Working of Artificial Neural Networks-Summary

- **Non- linear function**

These are one of the most widely used activation function. It helps the model in generalizing and adapting any sort of data in order to perform correct differentiation among the output. It solves the following problems faced by linear activation functions:

Since the non-linear function comes up with derivative functions, so the problems related to backpropagation has been successfully solved.

For the creation of deep neural networks, it permits the stacking up of several layers of the neurons.

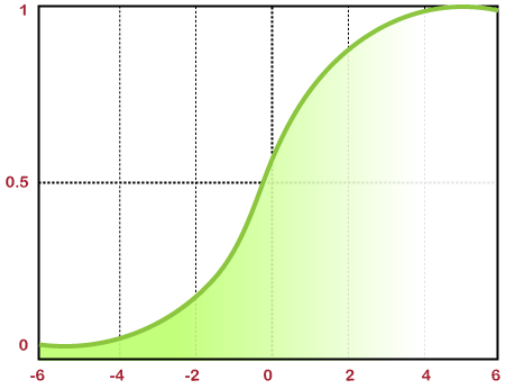


Working of Artificial Neural Networks-Summary

The non-linear activation function is further divided into the following parts:

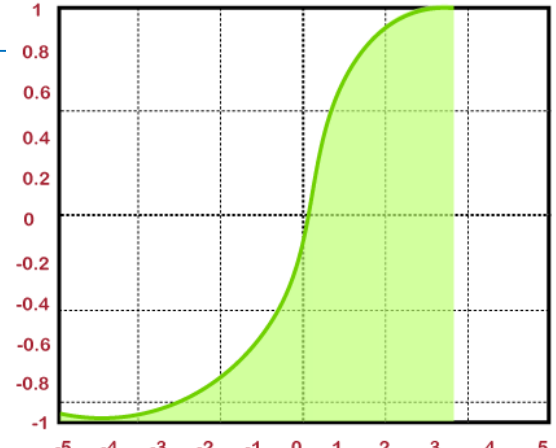
1. Sigmoid or Logistic Activation Function

It provides a smooth gradient by preventing sudden jumps in the output values. It has an output value range between 0 and 1 that helps in the normalization of each neuron's output. It's value ranges between 0 and 1 due to which it is highly preferred by binary classification whose result is either 0 or 1.



2. Tanh or Hyperbolic Tangent Activation Function

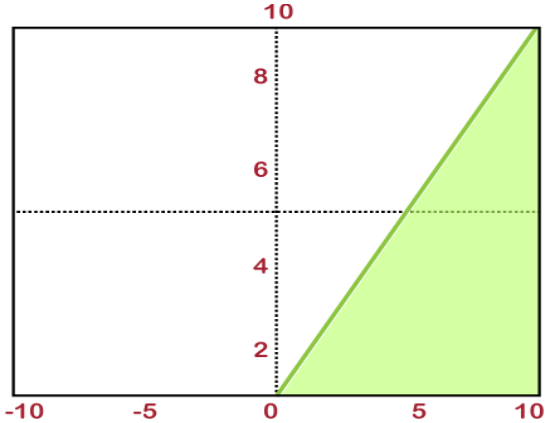
The tanh activation function works much better than that of the sigmoid function, or simply we can say it is an advanced version of the sigmoid activation function. Since it has a value range between -1 to 1, so it is utilized by the hidden layers in the neural network, and because of this reason, it has made the process of learning much easier.



Working of Artificial Neural Networks-Summary

3. ReLU(Rectified Linear Unit) Activation Function

ReLU is one of the most widely used activation function by the hidden layer in the neural network. Its value ranges from 0 to infinity. It clearly helps in solving out the problem of backpropagation. It tends out to be more expensive than the sigmoid, as well as the tanh activation function. It allows only a few neurons to get activated at a particular instance that leads to effectual as well as easier computations.



4. Softmax Function

It is one of a kind of sigmoid function whereby solving the problems of classifications. It is mainly used to handle multiple classes for which it squeezes the output of each class between 0 and 1, followed by dividing it by the sum of outputs. This kind of function is specially used by the classifier in the output layer.