Lecture 2

NNFL

Biological Neuron

- Neural networks are parallel computing devices,
- It’s aim is an attempt to make a computer model of the brain.
- The main objective is to develop a system to perform various computational tasks faster than the traditional systems.
- .

Tasks that can be done by NN

pattern recognition and

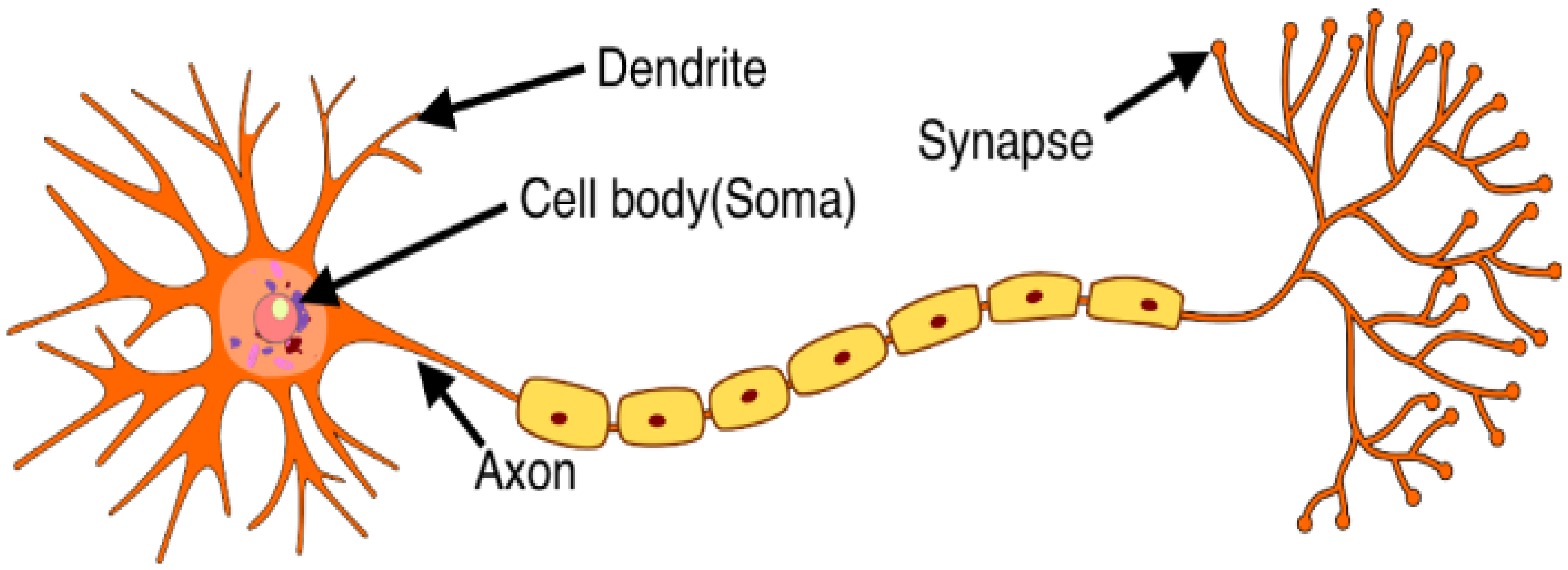
- classification,
- approximation,
- optimization, and
- data clustering

Biological neuron

- We have heard of the latest advancements in the field of **deep learning** due to the usage of different neural networks.
- At the most basic level, all such neural networks are made up of **artificial neurons**. These neurons **try to mimic the working of biological neurons**.
- Fist we will try to understand how **biological neurons work inside our brains…**

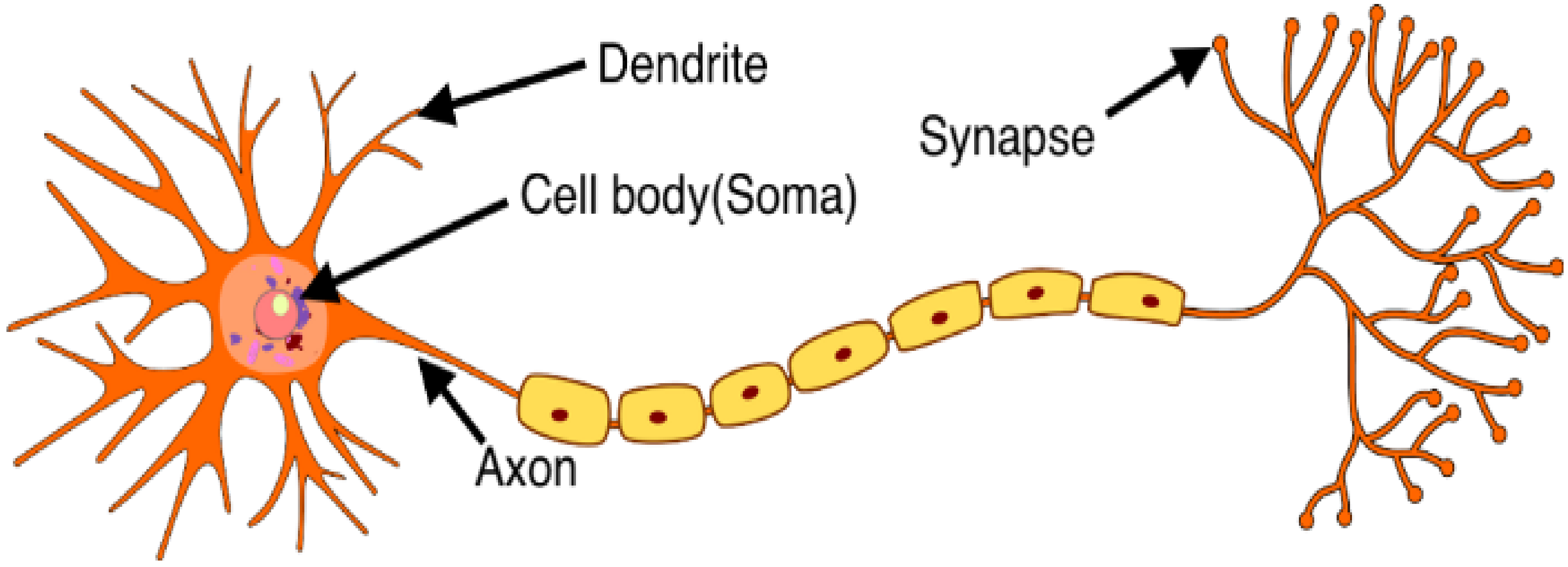
Biological neuron

- Now let us understand the ***basic parts*** of a neuron to get a deeper insight into how they actually work…



Parts of biological neuron:Receiving end

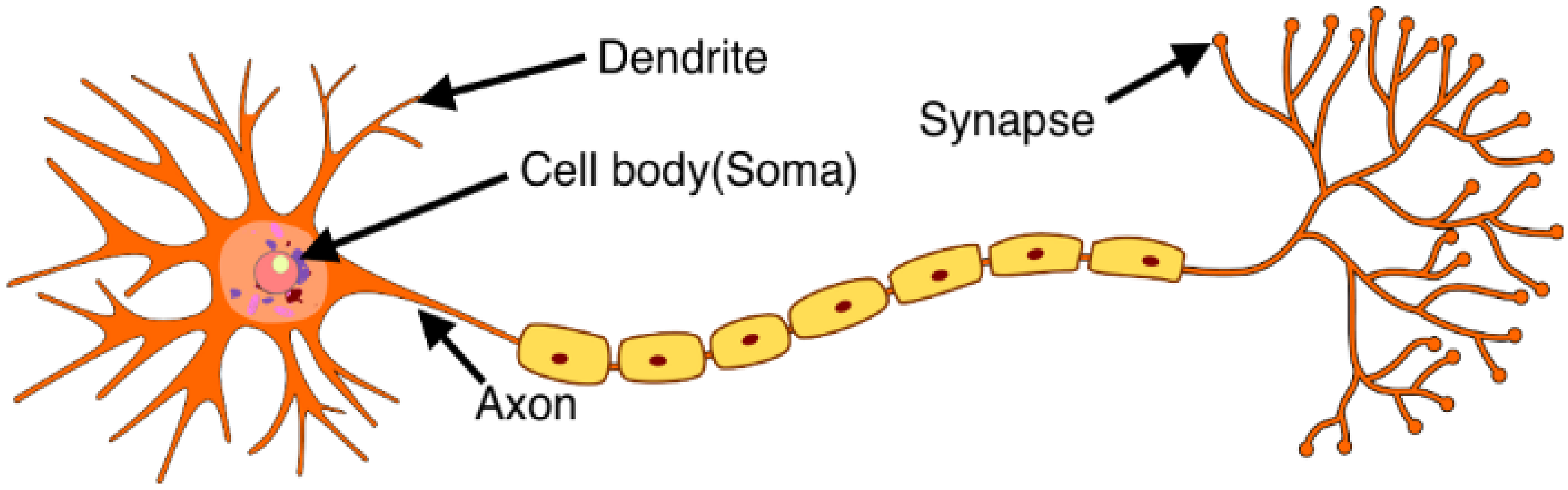
- **Dendrite**
- Dendrites are responsible for getting incoming signals from outside



Parts of biological neuron: processing unit

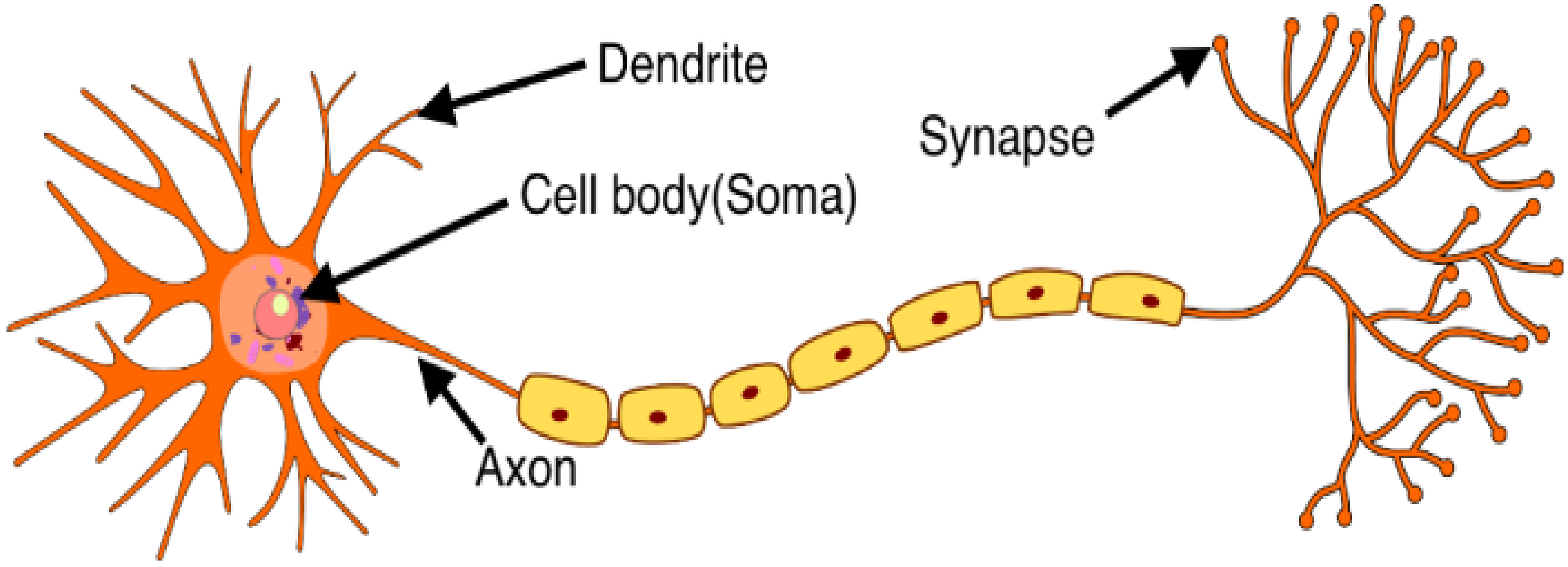
2. Soma

- Soma is the cell body responsible for the processing of input signals and deciding whether a neuron should fire an output signal



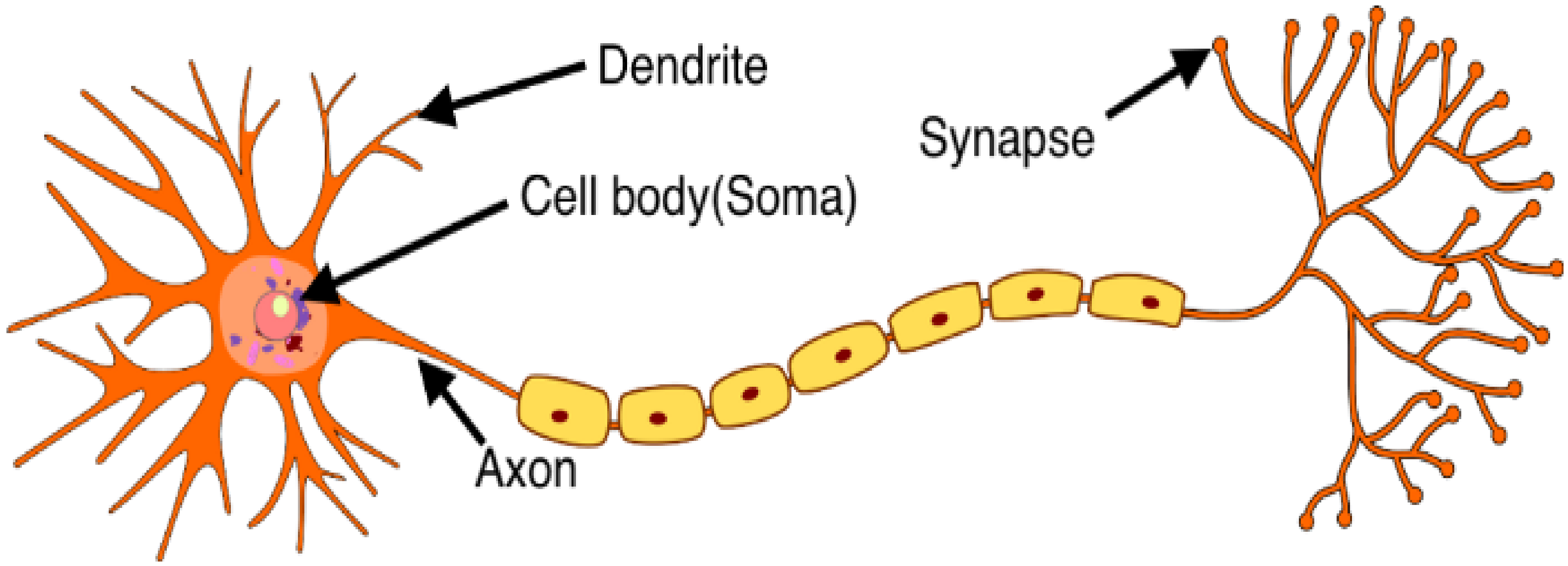
Parts of biological neuron

- **3. Axon** It is just like a cable through which neurons send the information.



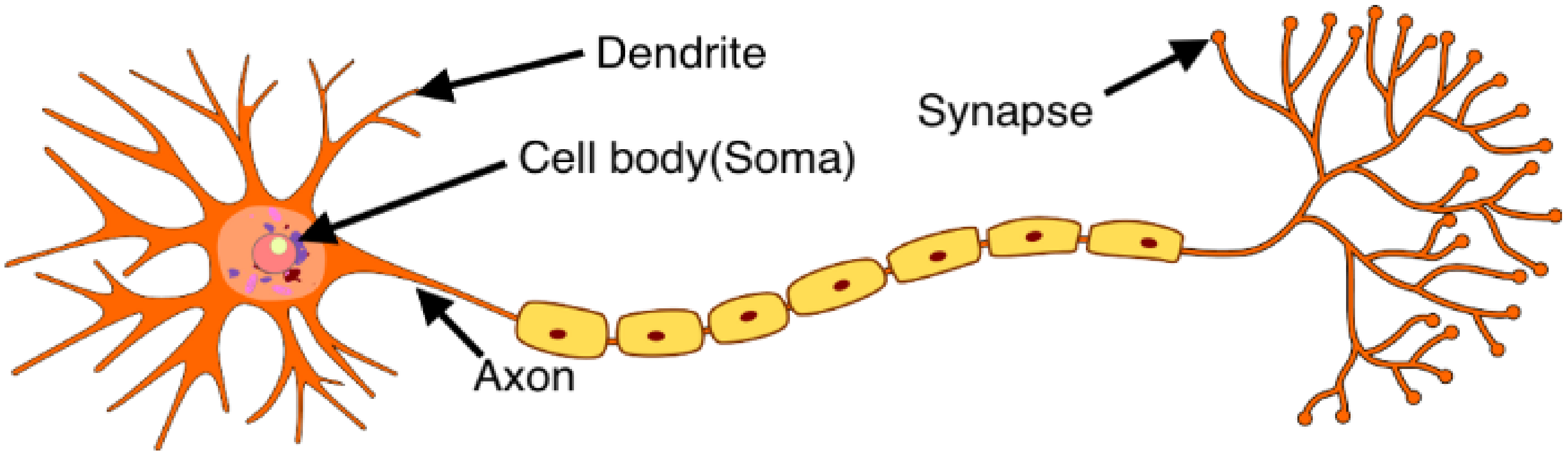
Parts of biological neuron

- **4. Synapse** – It is the connection between the axon and other neuron dendrite



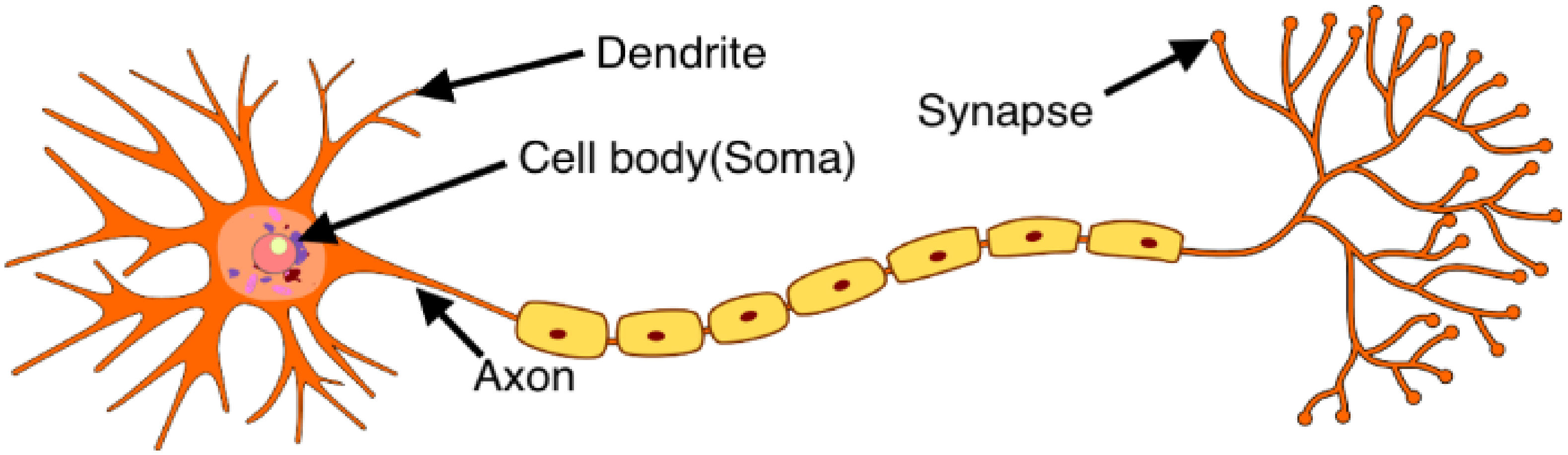
Working of biological neuron

- Incoming signals can be either
- **excitatory** — which means they tend to make the neuron **fire** (generate an electrical impulse)
- or **inhibitory** — which means that they tend to keep the neuron from firing.



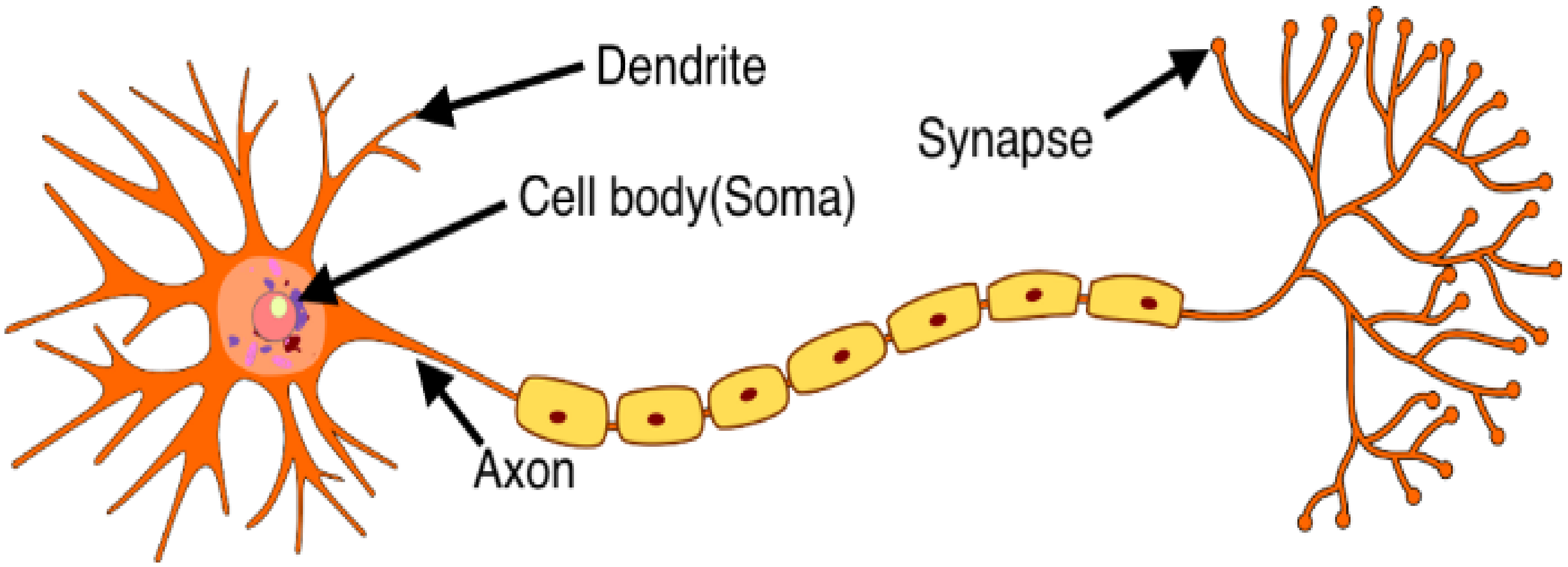
Working of biological neuron

- Most neurons receive many input signals throughout their dendritic trees.
- A single neuron may have **more than one set of dendrites** and may receive **many thousands** of input signals.



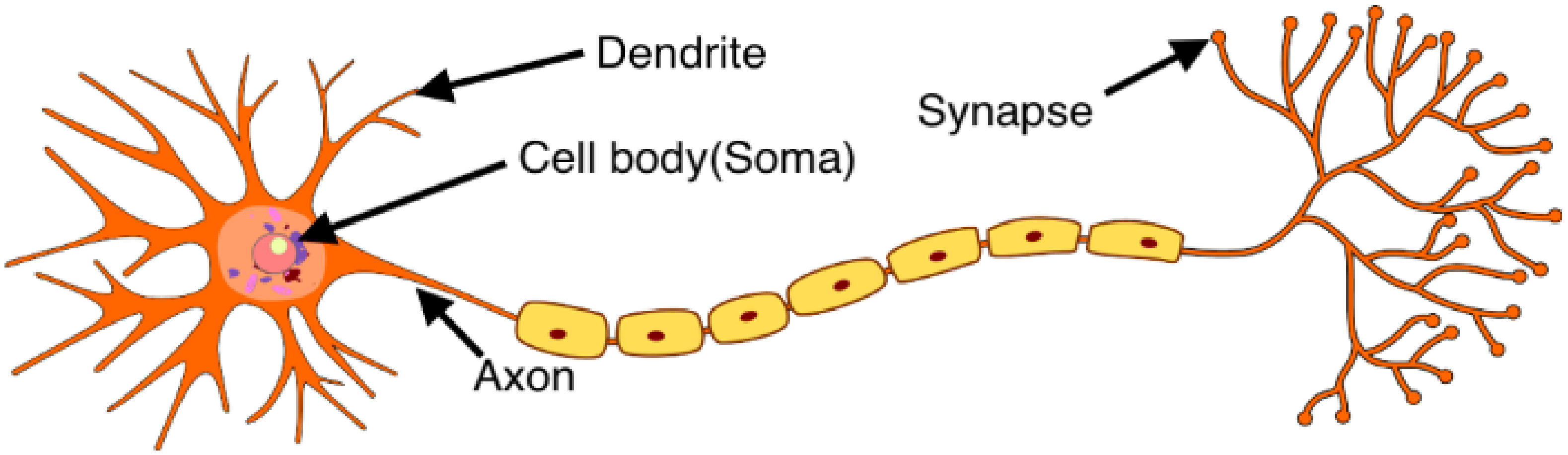
Working of biological neuron

- Whether or not a neuron is excited into firing an impulse depends on the **sum of all of the excitatory and inhibitory signals** it receives.



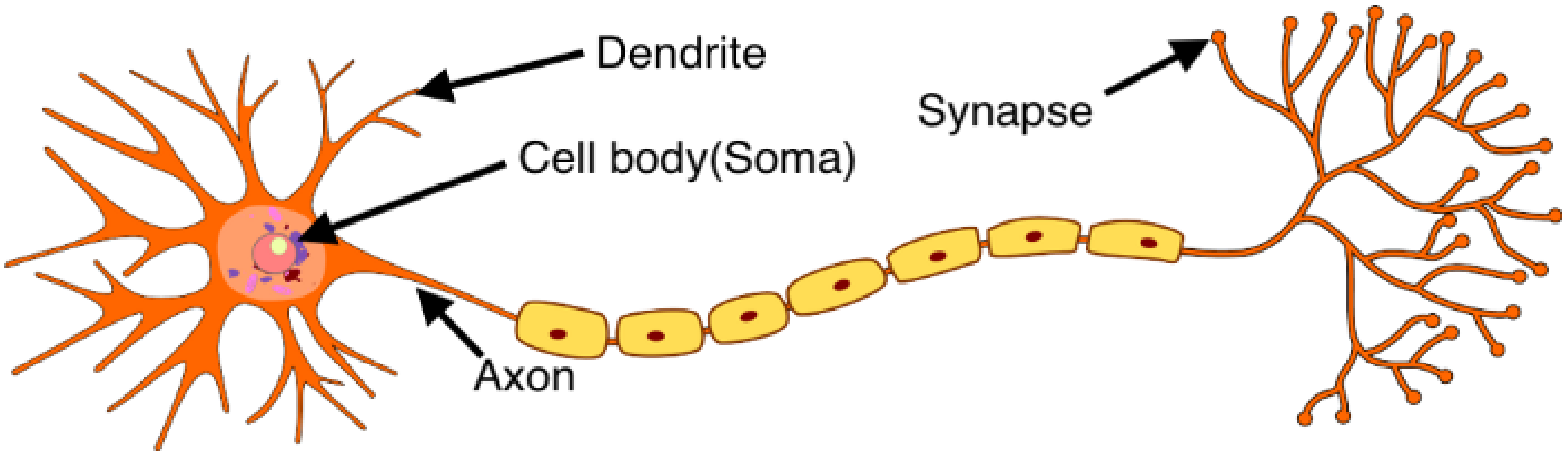
Working of biological neuron

- The processing of this information happens in **soma** which is neuron cell body. If the neuron does end up firing, the nerve impulse, or **action potential**, is conducted down the axon.



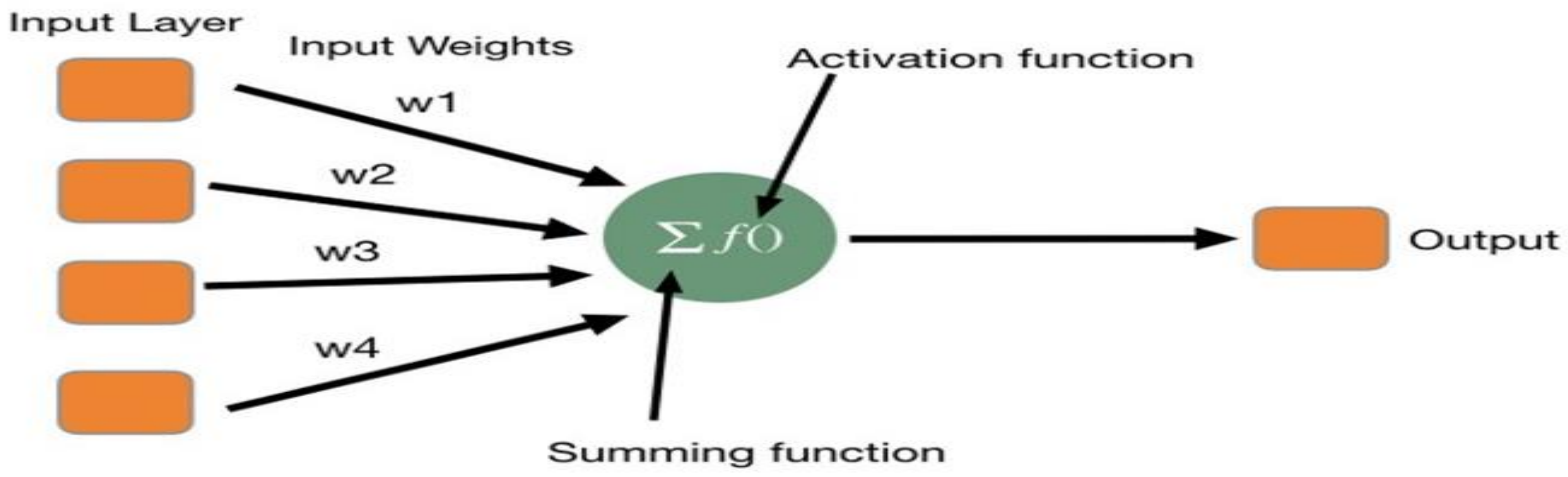
Working of biological neuron

- Towards its end, the axon splits up into many branches and develops bulbous swellings known as **axon terminals** (or **nerve terminals**). These axon terminals make connections on target cells.



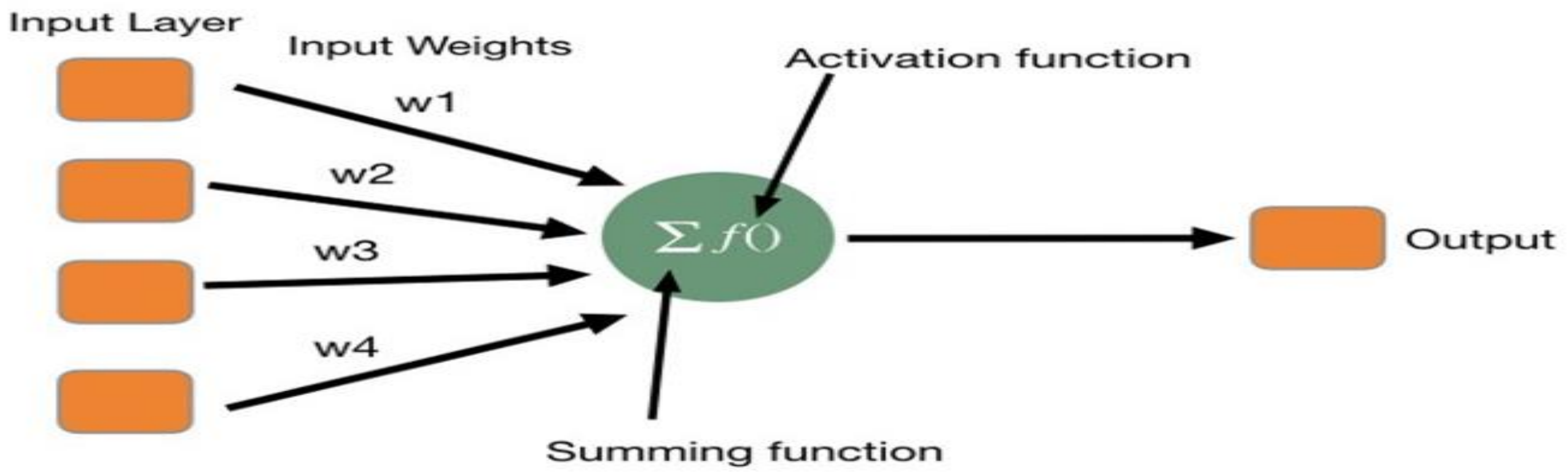
Each artificial neuron has the following main functions:

- Takes inputs from the input layer Weighs them separately and sums them up
- Pass this sum through a nonlinear function to produce output.



Artificial neuron

- A neuron is **a mathematical function modeled** on the working of biological neurons
- It is an elementary unit in an artificial neural network.Inputs are first **multiplied by weights**, then **summed** and passed through **a nonlinear function** to produce output



Artificial neuron

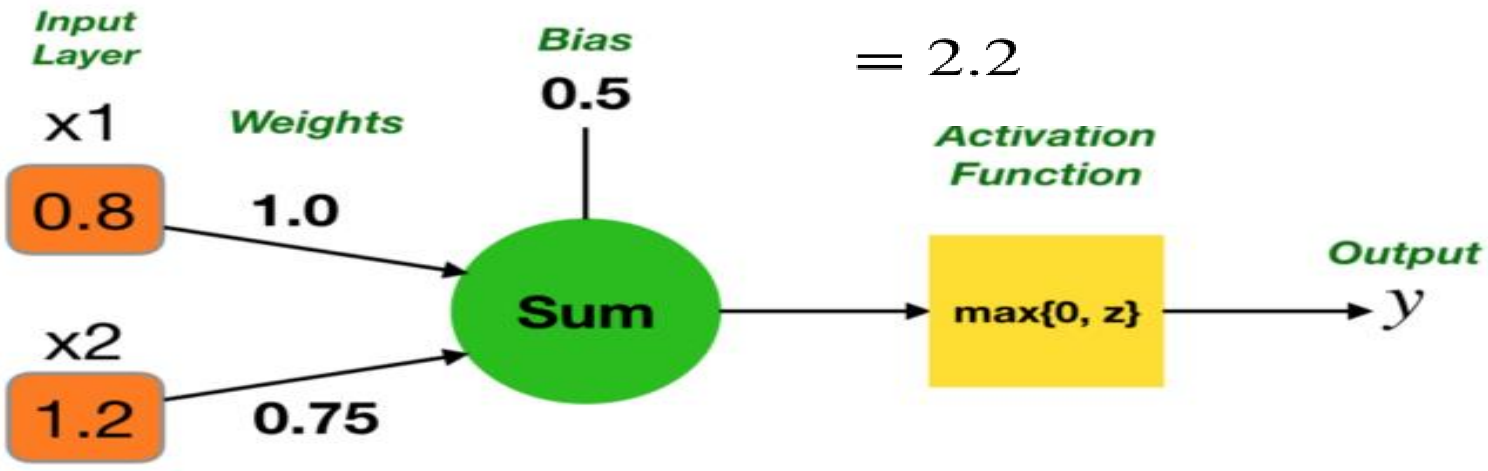
Combination value C, we got is 2.2 which is greater than 0 so the output value of our activation function will be 2.2.

$$C = w1 * x1 + w2 * x2 + b$$

$$= (1 * 0.8) + (0.75 * 1.2) + 0.5$$

$$= 0.8 + 0.9 + 0.5$$

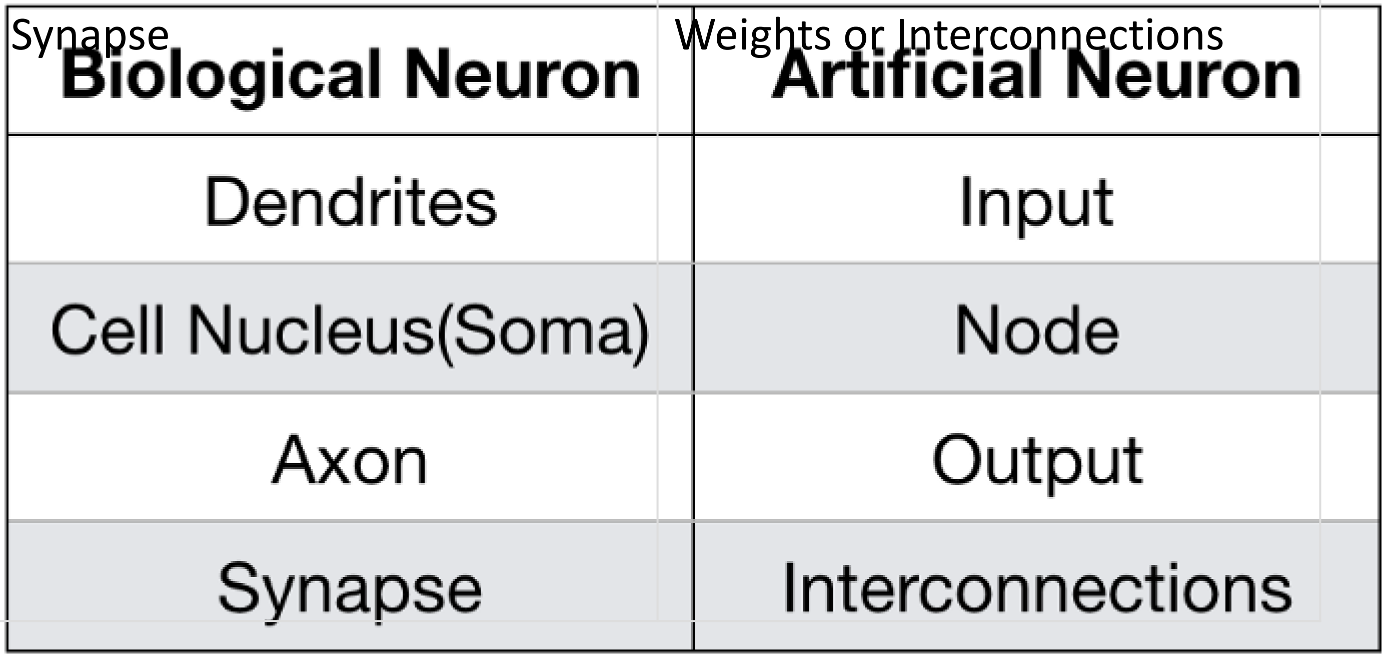
$$= 2.2$$



Biological Neuron vs. Artificial Neuron

- Since we have learnt a bit about both biological and artificial neurons we can now draw comparisons between both as follows:
-

Synapse	Weights or Interconnections



Biological Neuron	Artificial Neuron
Dendrites	Input
Cell Nucleus(Soma)	Node
Axon	Output
Synapse	Interconnections

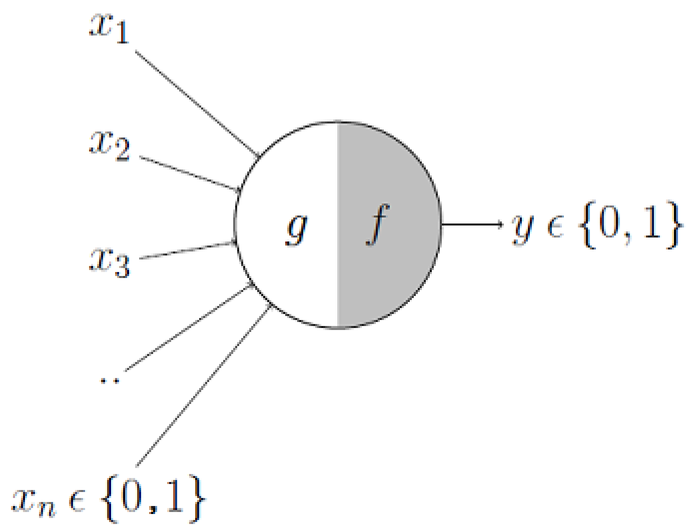
- For the above general model of artificial neural network, the net input can be calculated as follows −
- yin=x1.w1+x2.w2+x3.w3…xm.wm
- **yin=x1.w1+x2.w2+x3.w3…xm.wm**

- The output can be calculated by applying the activation function over the net input.
- **Y=F(yin)Y=F(yin)**
- Output = function net input calculated
-

Rererences

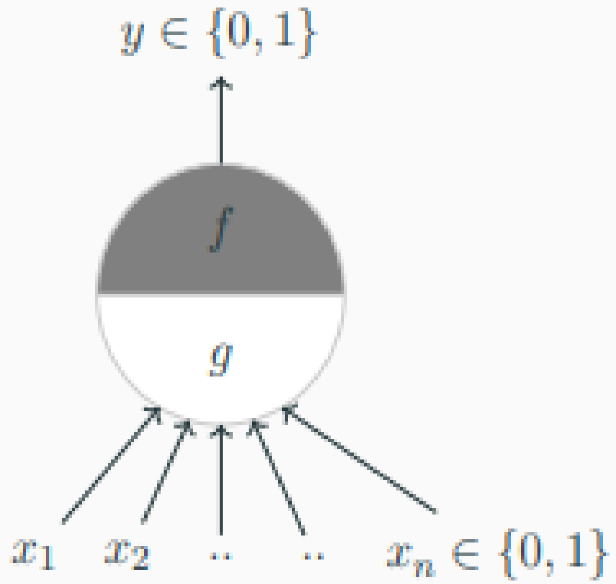
- References
- https://www.tutorialspoint.com/artificial_neural_network/artificial_neural_network_basic_concepts.htm

The McCulloch-Pitts Neuron



Ref.:http://www.cse.iitm.ac.in/~miteshk/CS7015/Slides/Teaching/pdf/Lecture2.pdf

The McCulloch-Pitts Neuron



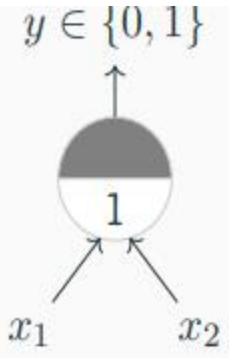
- McCulloch (neuroscientist) and Pitts (logician) proposed a highly simplified computational model of the neuron (1943)
- g aggregates the inputs and the function f takes a decision based on this aggregation
- The inputs can be excitatory or inhibitory
- $y = 0$ if any x_i is inhibitory, else

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

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

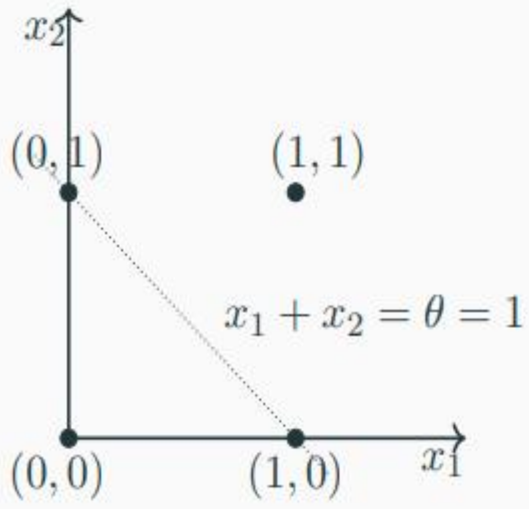
- θ is called the thresholding parameter
- This is called Thresholding Logic

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



OR function

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



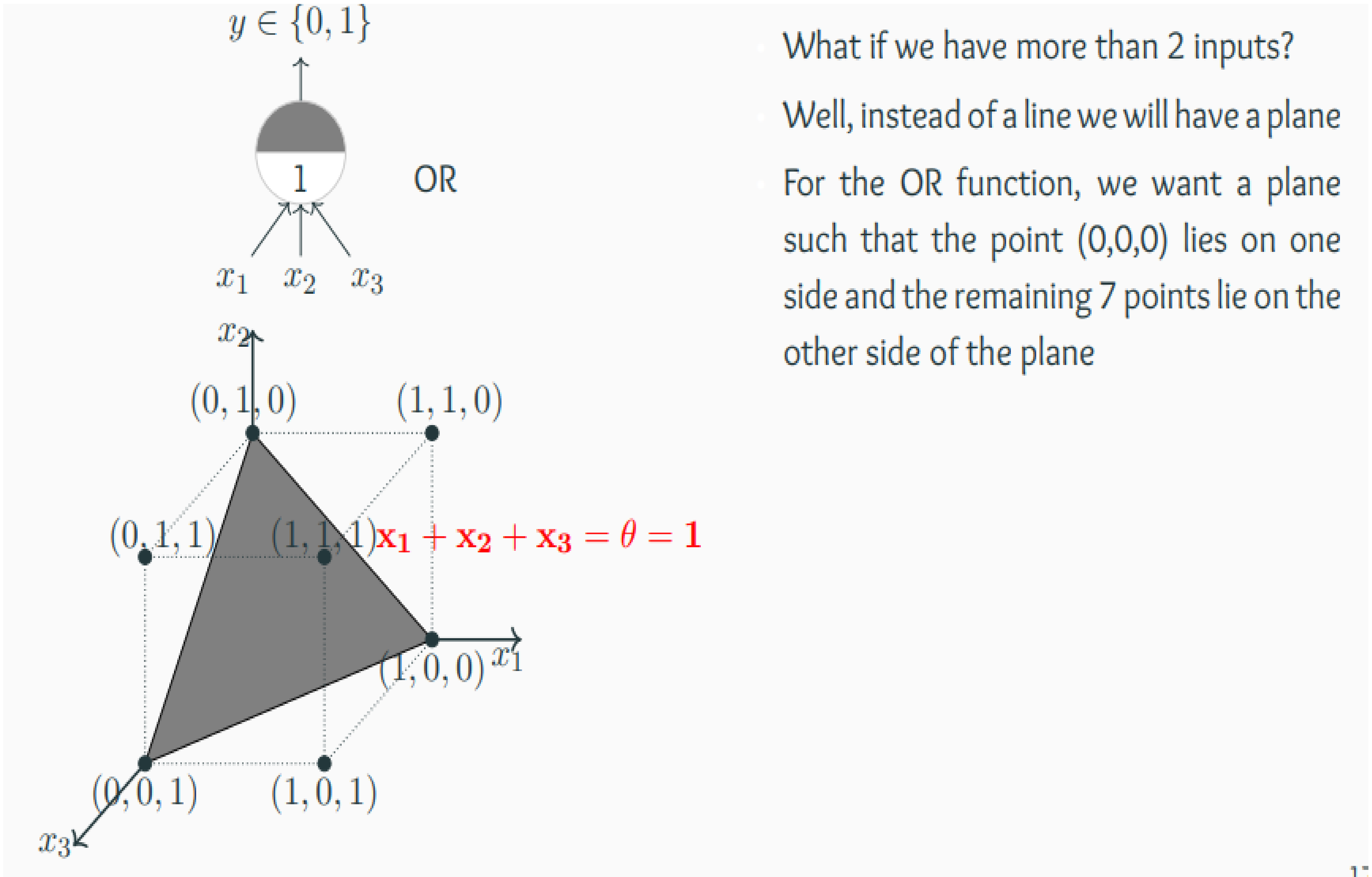
A single MP neuron splits the input points (4 points for 2 binary inputs) into two halves

Points lying on or above the line $\sum_{i=1}^{n} x_i - \theta = 0$ and points lying below this line

In other words, 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

Let us convince ourselves about this with a few more examples (if it is not already clear from the math)

What if we have more than 2 inputs?



What if we have more than 2 inputs?

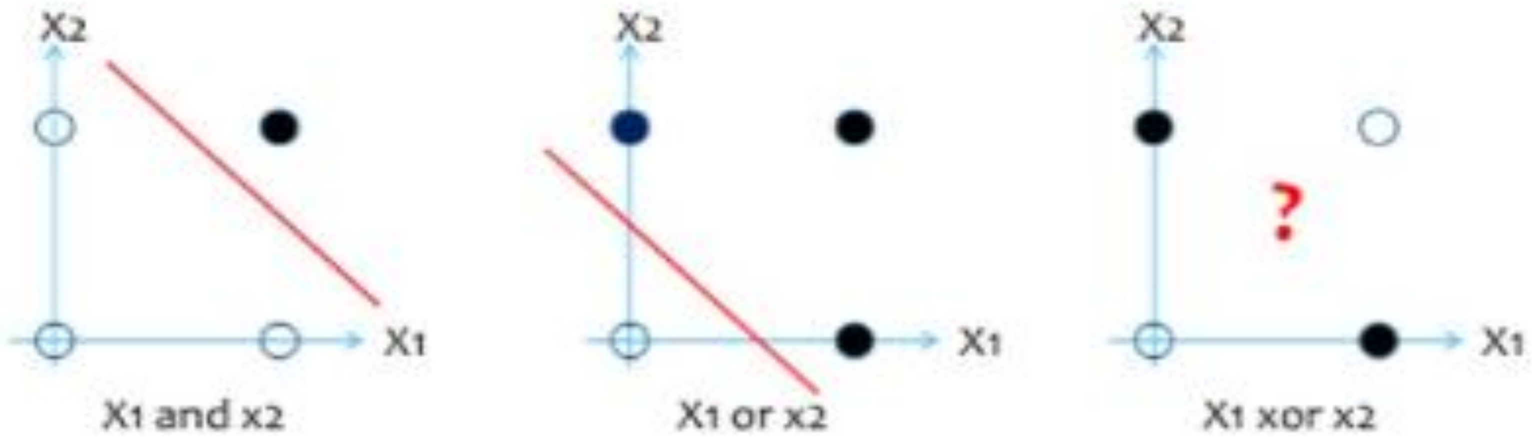
Well, instead of a line we will have a plane

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

The story so far ...

Linearly Separable patterns

- A single McCulloch Pitts Neuron can be used to represent **boolean functions which are linearly separable**

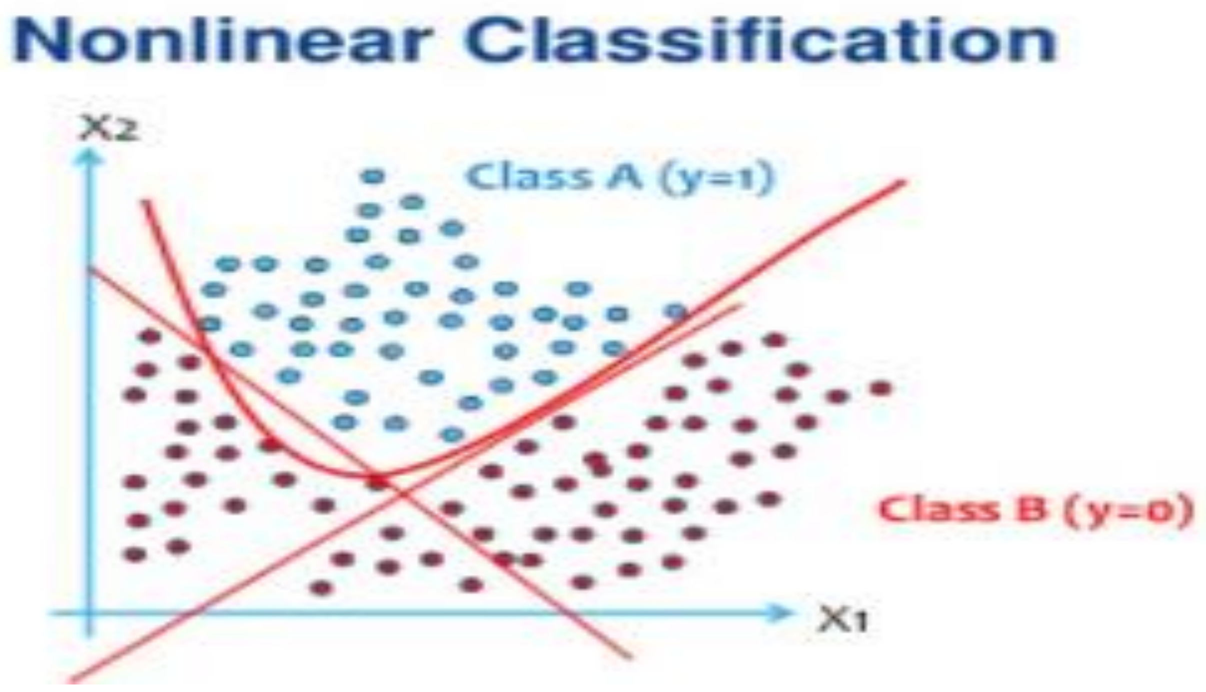


The story ahead...

- What about non-boolean (say, real) inputs ?
- Do we always need to hand code the threshold ?
- Are all inputs equal ?
- What if we want to assign more weight (importance) to some inputs ?
- What about functions which are not linearly separable ?

What about functions which are not linearly separable ?

- E.g. XOR



Perceptron model

- Frank Rosenblatt, an American psychologist, proposed the **classical perceptron model** (1958)
- A more general computational model than McCulloch–Pitts neurons

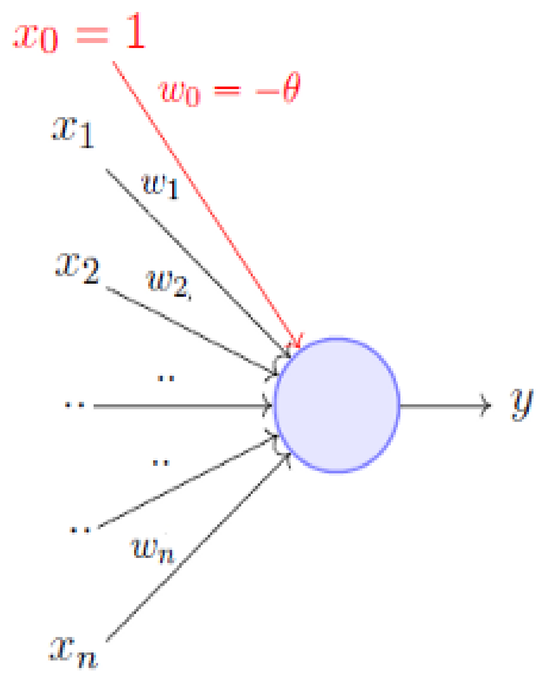
Main differences:

- Introduction of numerical weights for inputs and a mechanism for learning these weights
- **Inputs are no longer limited to boolean values.**
- Refined and carefully analyzed by Minsky and Papert (1969) - their model is referred to as **the perceptron model** here

Perceptron model

- the perceptron is a convenient model of a biological neuron,
- it was the early algorithm of binary classifiers in supervised machine learning.
- The purpose behind the designing of the perceptron model was to incorporate visual inputs, organizing subjects or captions into one of two classes and dividing classes through a line.

Perceptron model



A more accepted convention,

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

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

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

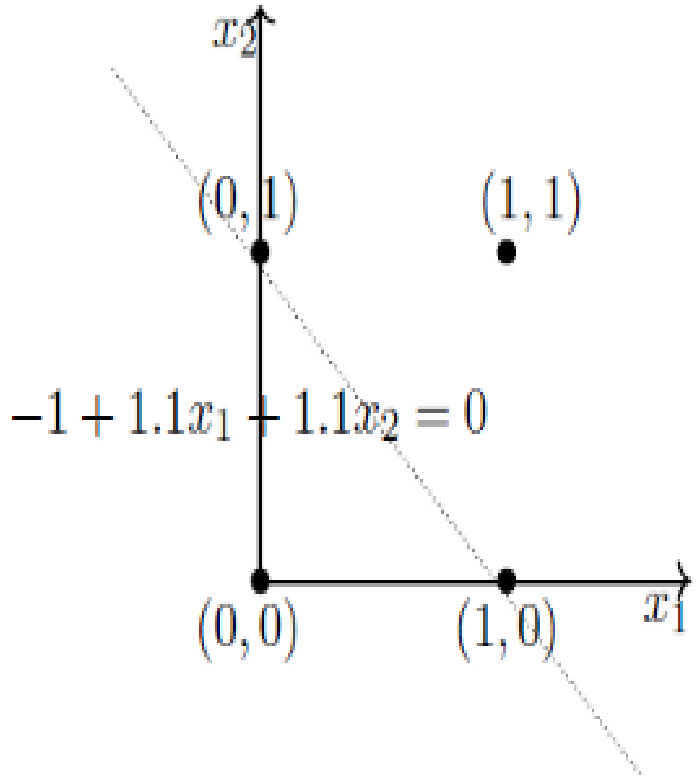
Function Using A Perceptron

x_1	x_2	OR	
0	0	0	$w_0 + \sum_{i=1}^{2} w_i x_i < 0$
1	0	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$
0	1	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$
1	1	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$

$$w_0 + w_1 \cdot 0 + w_2 \cdot 0 < 0 \implies w_0 < 0$$
$$w_0 + w_1 \cdot 0 + w_2 \cdot 1 \geq 0 \implies w_2 > -w_0$$
$$w_0 + w_1 \cdot 1 + w_2 \cdot 0 \geq 0 \implies w_1 > -w_0$$
$$w_0 + w_1 \cdot 1 + w_2 \cdot 1 \geq 0 \implies w_1 + w_2 > -w_0$$

One possible solution is

$w_0 = -1,\ w_1 = 1.1,\ w_2 = 1.1$



Perceptron Function

- Perceptron is a function that maps its input “x,” which is multiplied with the learned weight coefficient; an output value ”f(x)”is generated.
- In the equation given above:
- “w” = vector of real-valued weights
- “b” = bias (an element that adjusts the boundary away from origin without any dependence on the input value)
- “x” = vector of input x values
- “m” = number of inputs to the Perceptron
- The output can be represented as “1” or “0.” It can also be represented as “1” or “-1” depending on which activation function is used.

Different activation functions

1) **A Sigmoid Function** or **Binary sigmoidal function** is a mathematical function with a Sigmoid Curve ("S" Curve).

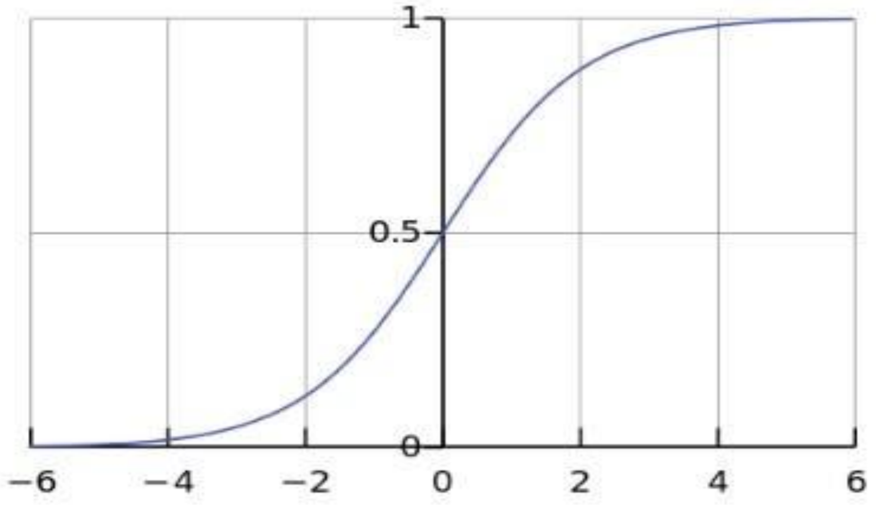
- It is a special case of the logistic function and is defined by the function given below:

$$\phi_{logistic}(z) = \frac{1}{1 + e^{-z}}$$

Ref:-https://www.simplilearn.com/tutorials/deep-learning-tutorial/perceptron

A Sigmoid Function

- The curve of the Sigmoid function called “S Curve” is shown here.
- This is called a logistic sigmoid and leads to a proba 0 and 1.



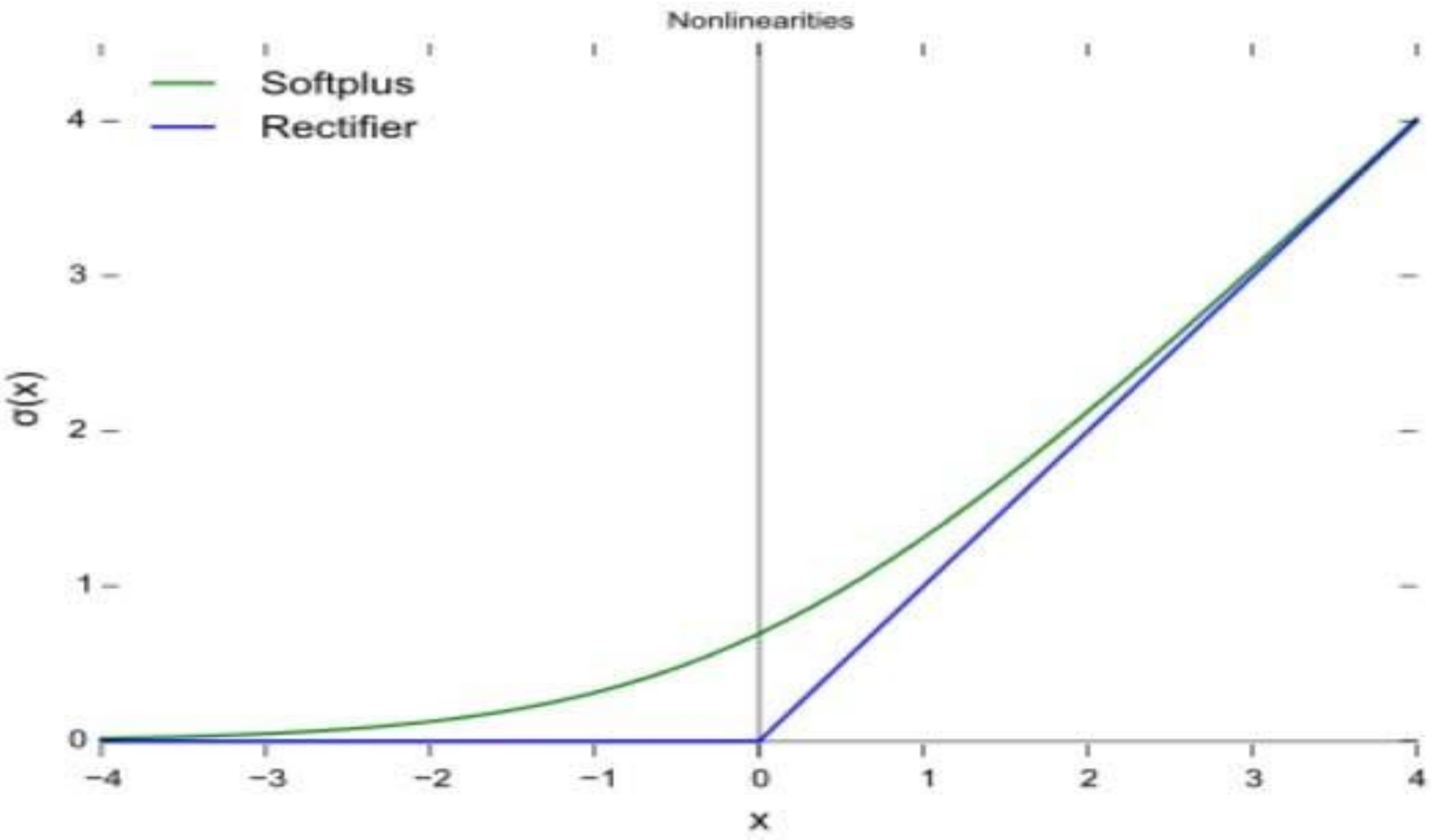
Activation functions: **A Sigmoid Function**

- **Use of sigmoidal activation function**
- when one is interested in probability mapping rather than precise values of input parameter t.
- The sigmoid output is close to zero for highly negative input.
- This can be a problem in neural network training and can lead to slow learning and the model getting trapped in local minima during training. Hence, **hyperbolic tangent is more preferable** as an activation function in hidden layers of a neural network.

Activation functions: **Softplus Functions**

- **Rectifier and Softplus Functions**
- Apart from Sigmoid and Sign activation functions seen earlier, other common activation functions are ReLU and Softplus. They eliminate negative units as an output of max function will output 0 for all units 0 or less.

Activation functions: **Softplus Functions**



ReLU

- A rectifier or ReLU (Rectified Linear Unit) is a commonly used activation function. This function allows one to eliminate negative units in an ANN. This is the most popular activation function used in deep neural networks.
- A smooth approximation to the rectifier is the Softplus function.
- The derivative of Softplus is the logistic or sigmoid function.
- In the next section, let us discuss the advantages of ReLu function.

Advantages of ReLu Functions

- **Advantages of ReLu Functions**
- Allows faster and more effective training of deep neural architectures on large and complex datasets
- Sparse activation of only about 50% of units in a neural network (as negative units are eliminated)
- More plausible or one-sided, compared to anti-symmetry of tanh
- Efficient gradient propagation, which means no vanishing or exploding gradient problems
- Efficient computation with the only comparison, addition, or multiplication
- Scales well

Limitations of ReLu Functions

- **Limitations of ReLu Functions**
- **Non-differentiable at zero** - Non-differentiable at zero means that values close to zero may give inconsistent or intractable results.
- **Non-zero centered** - Being non-zero centered creates asymmetry around data (only positive values handled), leading to the uneven handling of data.
- **Unbounded** - The output value has no limit and can lead to computational issues with large values being passed through.
- **Dying ReLU problem** - When the learning rate is too high, Relu neurons can become inactive and “die.”
- In the next section, let us focus on the Softmax function.

Softmax Function

- **Softmax Function**
- Another very popular activation function is the Softmax function.
- The Softmax outputs probability of the result belonging to a certain set of classes.
- It is akin to a categorization logic at the end of a neural network. For example, it may be used at the end of a neural network that is trying to determine if the image of a moving object contains an animal, a car, or an airplane.
- In Mathematics, the Softmax or normalized exponential function is a generalization of the logistic function that squashes a K-dimensional vector of arbitrary real values to a K-dimensional vector of real values in the range (0, 1) that add up to 1

- n probability theory, the output of the Softmax function represents a probability distribution over K different outcomes.
- In Softmax, the probability of a particular sample with net input z belonging to the ith class can be computed with a normalization term in the denominator, that is, the sum of all M linear functions:

$$p(y=i|z)=\phi(z)=\frac{e^{z_i}}{\sum_{i=1}^{M} e^{z_j}}$$

- The Softmax function is used in ANNs and Naïve Bayes classifiers.
- **For example,** if we take an input of [1,2,3,4,1,2,3], the Softmax of that is [0.024, 0.064, 0.175, 0.475, 0.024, 0.064, 0.175]. The output has most of its weight if the original input is '4' This function is normally used for:
- Highlighting the largest values
- Suppressing values that are significantly below the maximum value.

Hyperbolic Tangent 0r Bipolar sigmoidal function

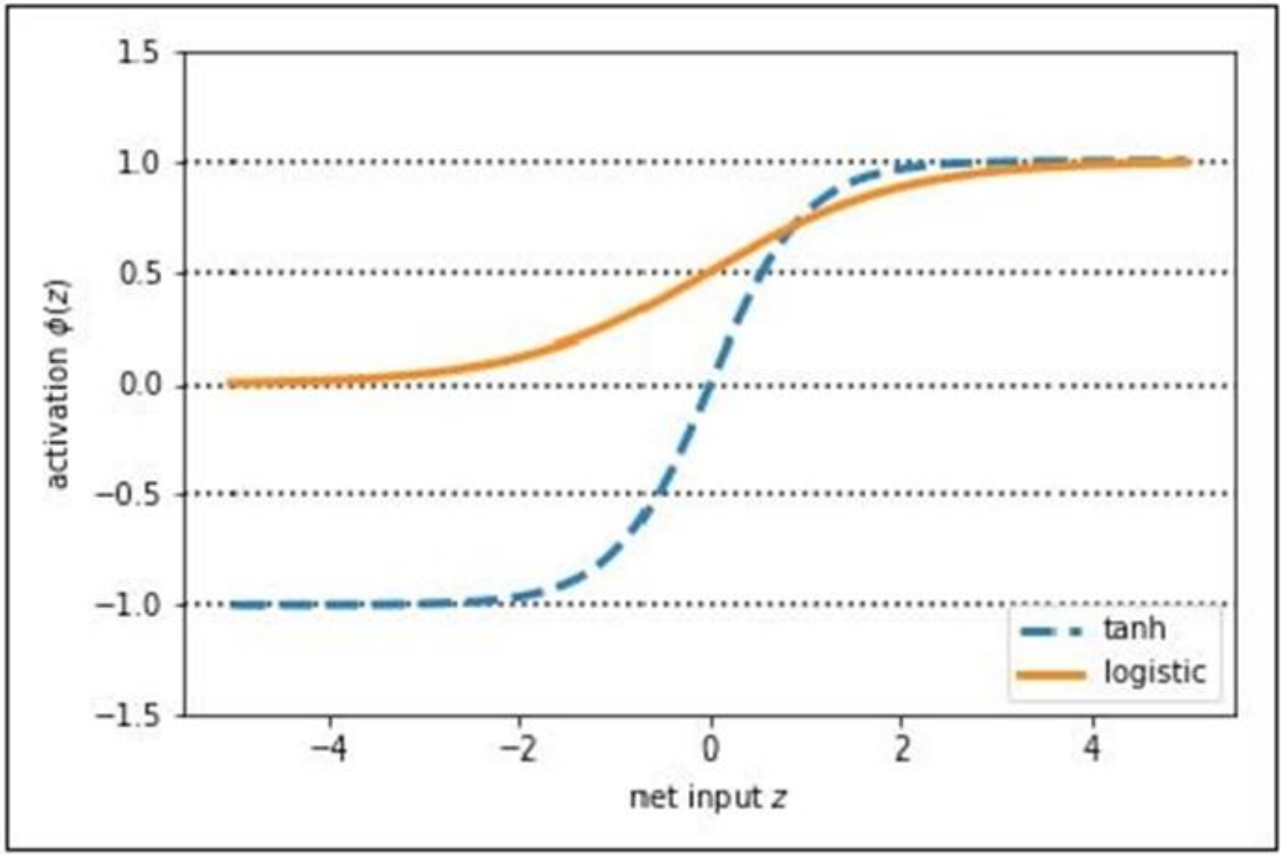
The tanh function has two times larger output space than the logistic function.

With larger output space and symmetry around zero, the tanh function leads to the more even handling of data, and it is easier to arrive at the global maxima in the loss function.

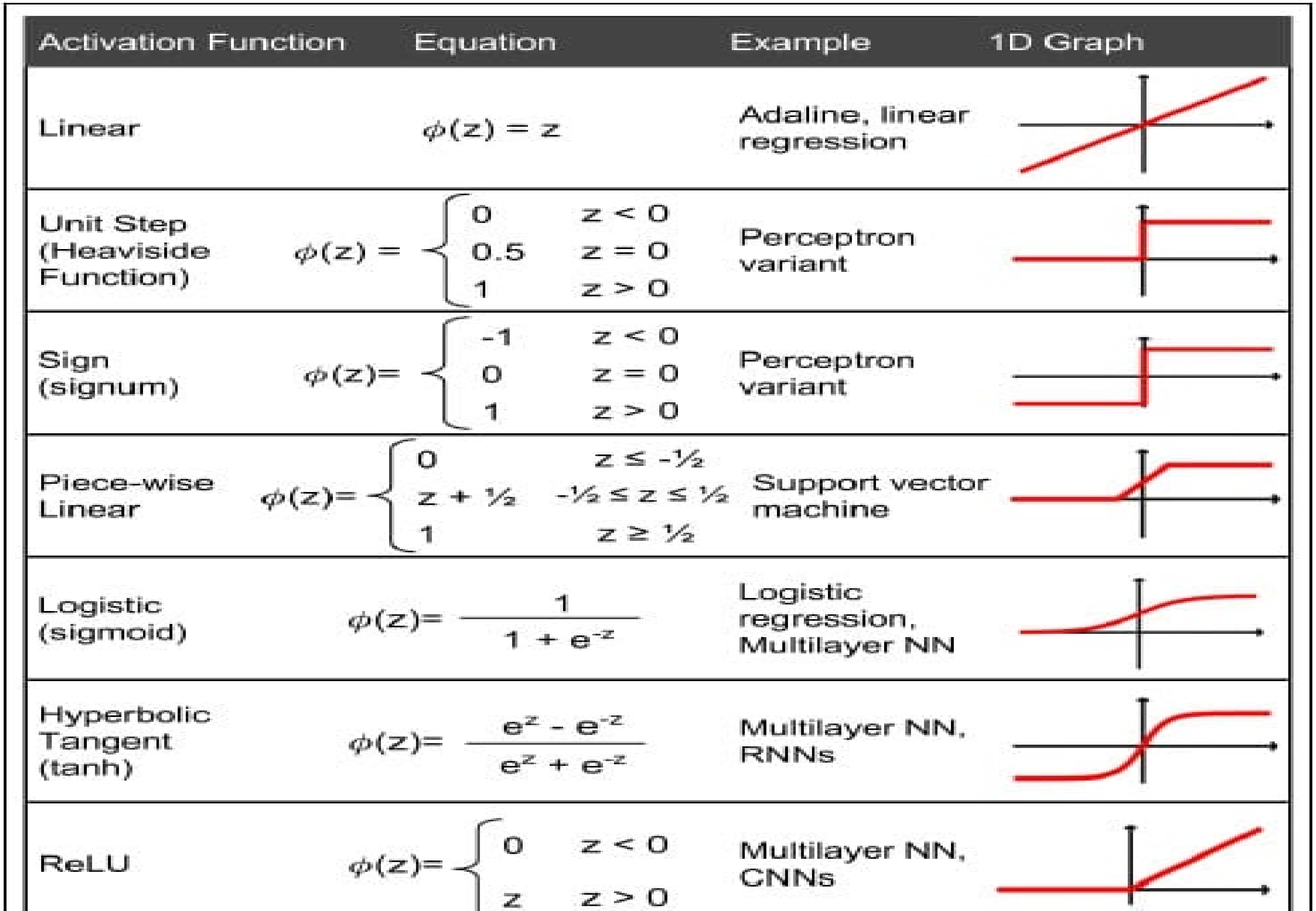
- Hyperbolic tangent:

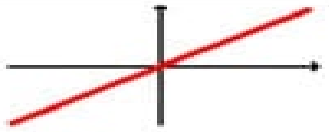
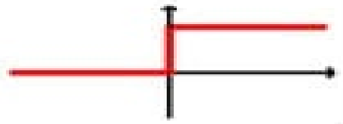
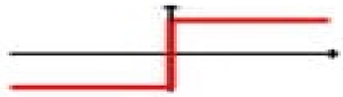
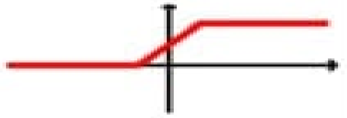
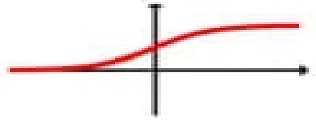
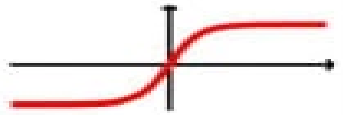
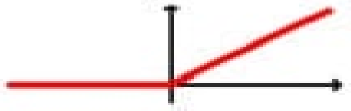
$$\tanh x = \frac{\sinh x}{\cosh x} = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} =$$

$$= \frac{e^{2x} - 1}{e^{2x} + 1} = \frac{1 - e^{-2x}}{1 + e^{-2x}}.$$


1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
activation $\phi(z)$
−4
−2
0
2
4
net input z
tanh
logistic

Summary of activation functions



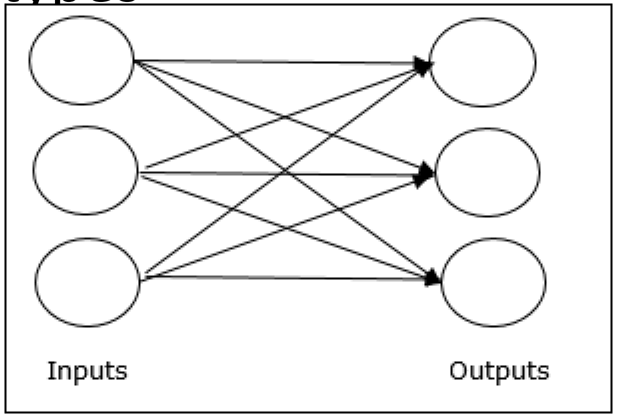
Activation Function	Equation	Example	1D Graph
Linear	$\phi(z) = z$	Adaline, linear regression	
Unit Step (Heaviside Function)	$\phi(z) = \begin{cases} 0 & z < 0 \\ 0.5 & z = 0 \\ 1 & z > 0 \end{cases}$	Perceptron variant	
Sign (signum)	$\phi(z)= \begin{cases} -1 & z < 0 \\ 0 & z = 0 \\ 1 & z > 0 \end{cases}$	Perceptron variant	
Piece-wise Linear	$\phi(z)= \begin{cases} 0 & z \leq -½ \\ z + ½ & -½ \leq z \leq ½ \\ 1 & z \geq ½ \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z)= \frac{1}{1 + e^{-z}}$	Logistic regression, Multilayer NN	
Hyperbolic Tangent (tanh)	$\phi(z)= \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$	Multilayer NN, RNNs	
ReLU	$\phi(z)= \begin{cases} 0 & z < 0 \\ z & z > 0 \end{cases}$	Multilayer NN, CNNs	

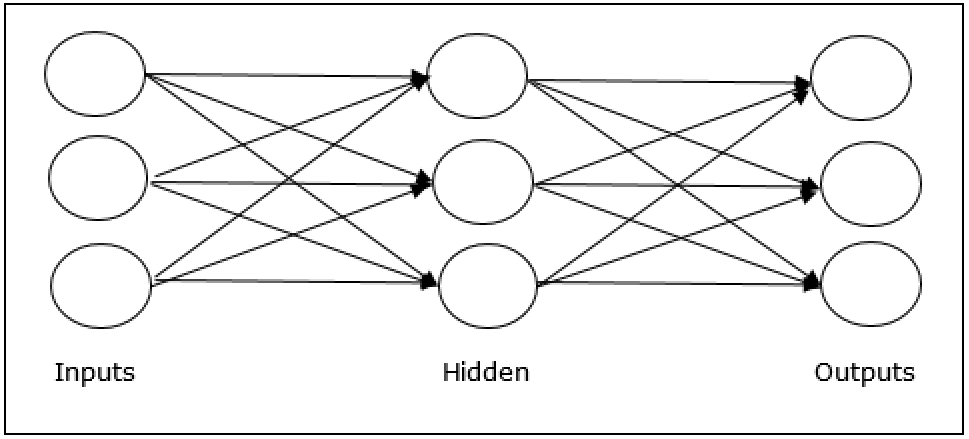
Network Topology/architectures

- A network topology is the arrangement of a network along with its nodes and connecting lines. According to the topology, ANN can be classified as the following kinds −
- Feedforward Network

Network Topology/architectures

- Feedforward Network
- It is a non-recurrent network having processing units/nodes in layers and all the nodes in a layer are connected with the nodes of the previous layers.
- The connection has different weights upon them. There is no feedback loop means the signal can only flow in one direction, from input to output. It may be divided into the following two types −





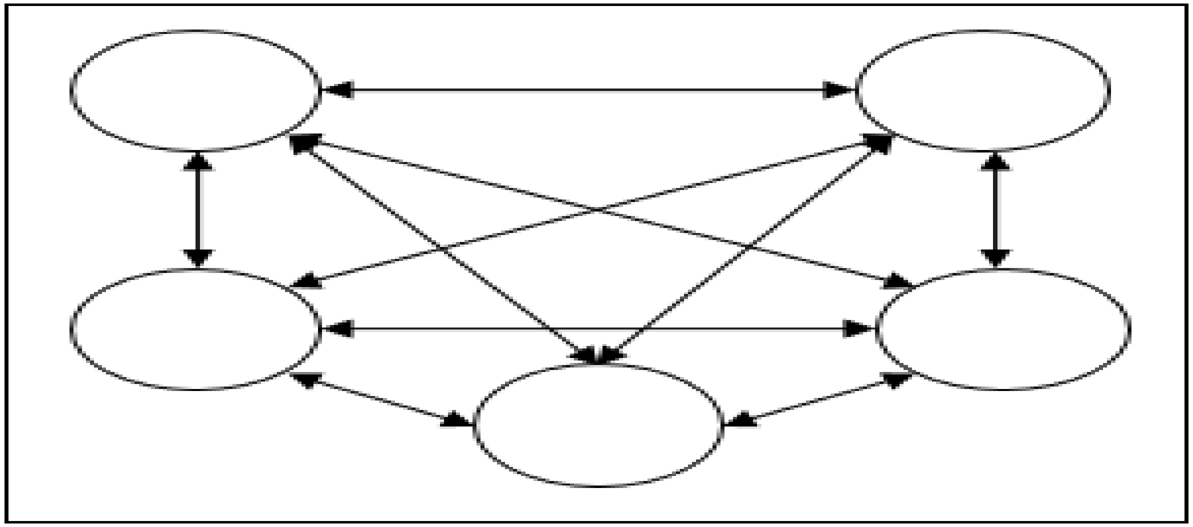
Network Topology/architectures

Feedforward Network

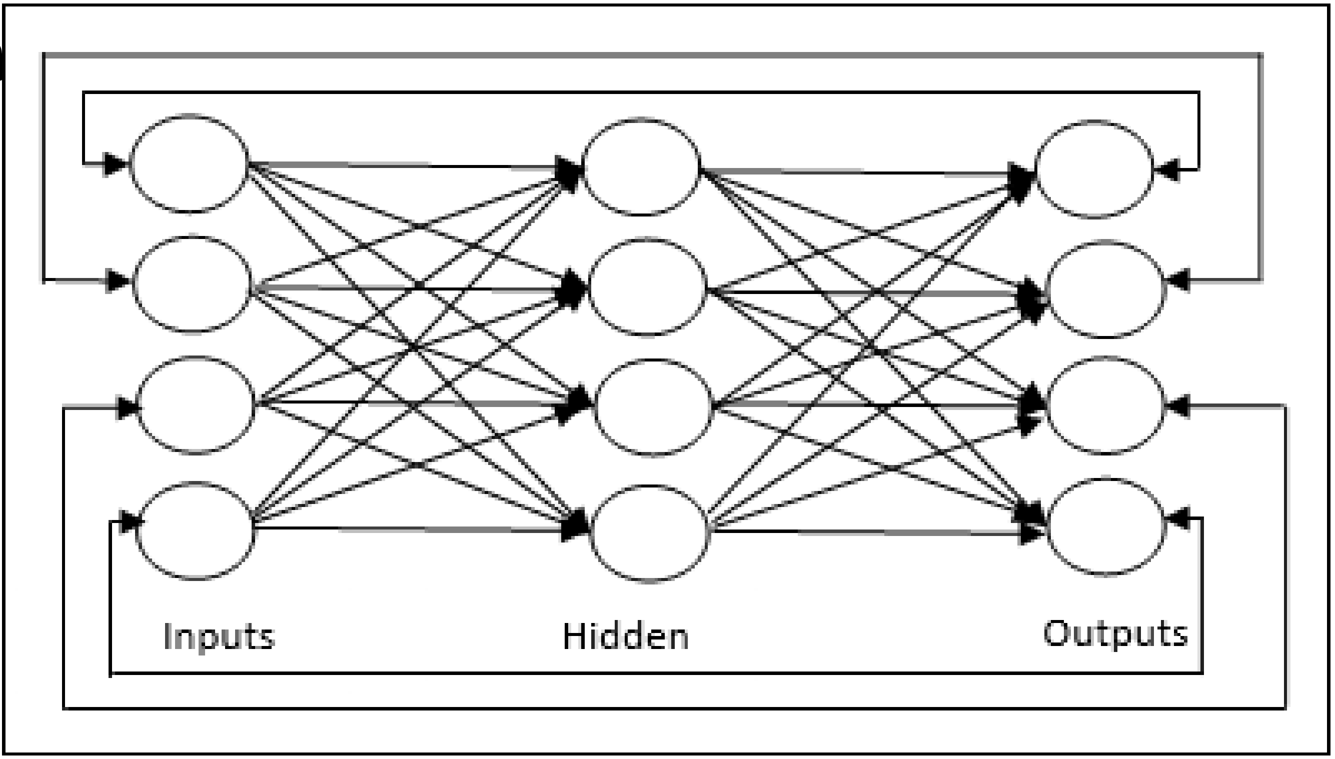
- **Single layer feedforward network** − The concept is of feedforward ANN having only one weighted layer. In other words, we can say the input layer is fully connected to the output layer.
- **Multilayer feedforward network** − The concept is of feedforward ANN having more than one weighted layer. As this network has one or more layers between the input and the output layer, it is called hidden layers.
-

- Feedback Network
- As the name suggests, a feedback network has feedback paths, which means the signal can flow in both directions using loops. This makes it a non-linear dynamic system, which changes continuously until it reaches a state of equilibrium.

- It may be divided into the following types –
- **Recurrent networks** – They are feedback networks with closed loops. Following are the two types of recurrent networks.
- **Fully recurrent network** – It is the simplest neural network architecture because all nodes are connected to all other nodes and each node works as both input and output.



- **Jordan network** – It is a closed loop network in which the output will go to the input again as feedback as shown in the following diagram
-


Inputs
Hidden
Outputs