HINDUSTAN
INSTITUTE OF TECHNOLOGY & SCIENCE
(DEEMED TO BE UNIVERSITY)

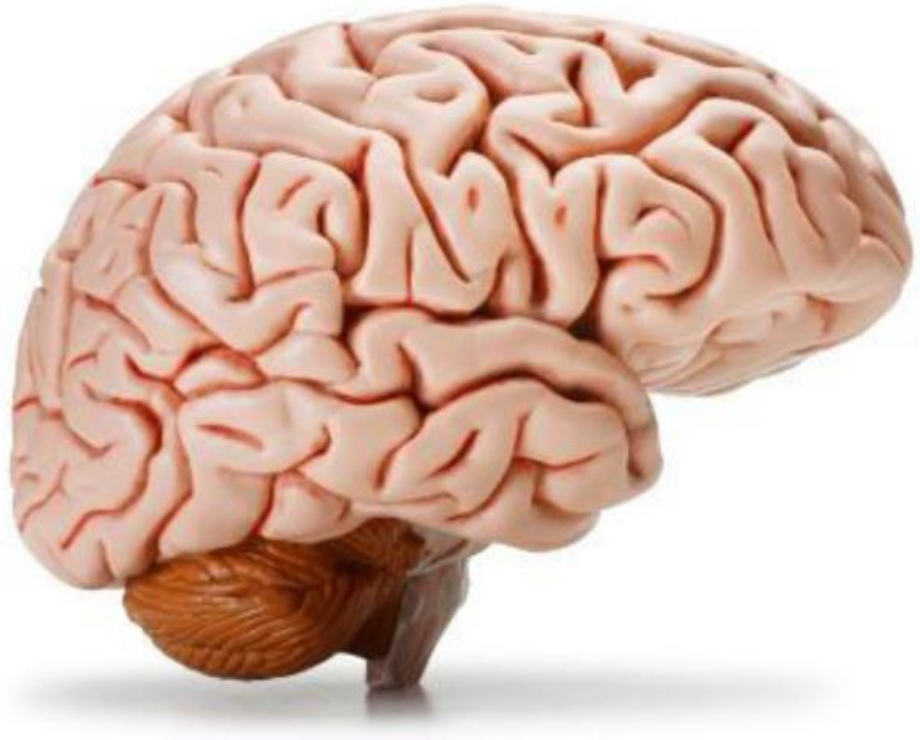
EAL51501 – ARTIFICIAL INTELLIGENCE
B.Tech[AIML] – III Semester

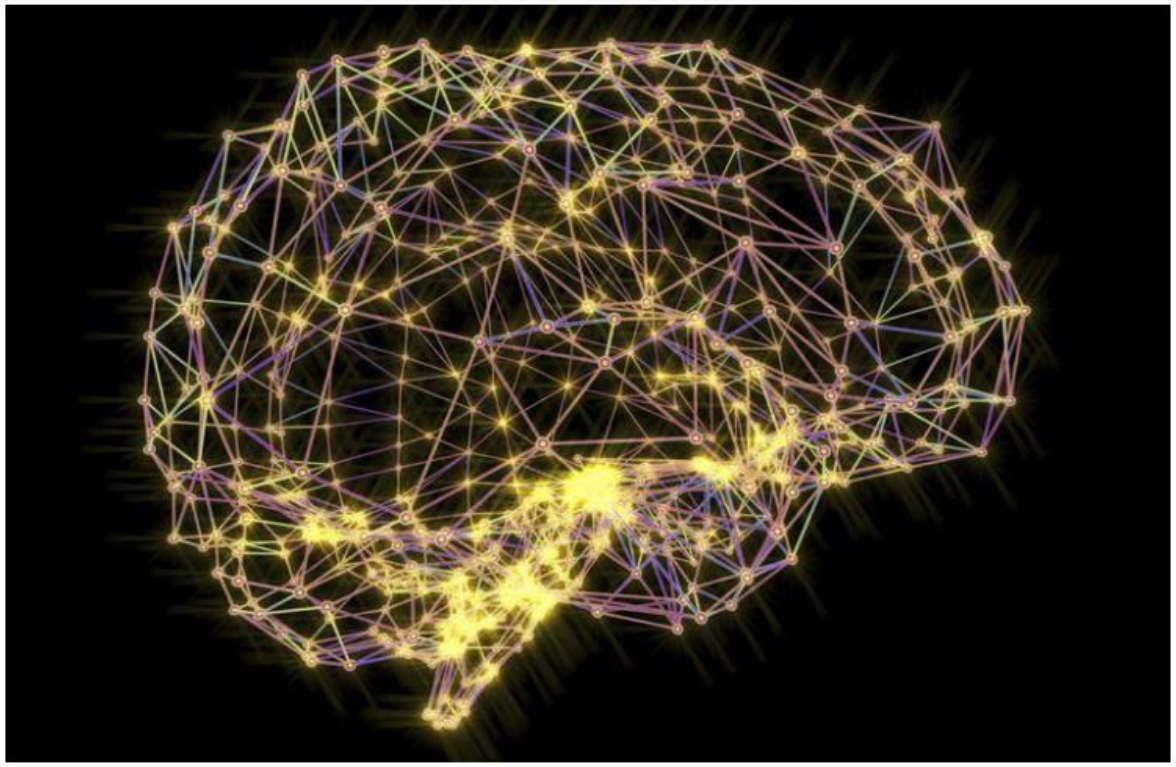
K.Kowsalya
Assistant Professor (SS)
School of Computing Sciences,
Department of Computer Science and Engineering

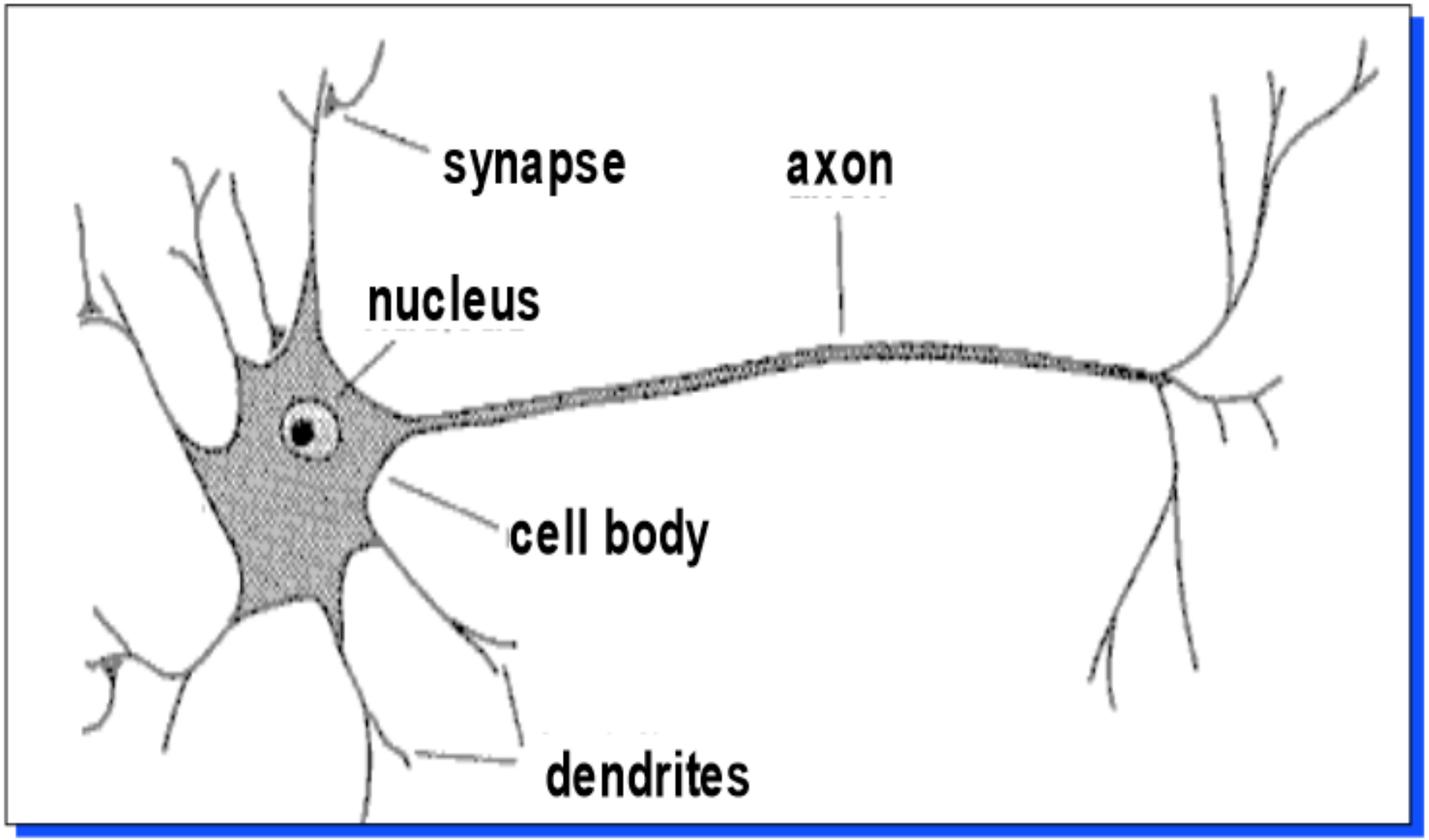
UNIT-IV

- **MODULE 4: INTRODUCTION TO NEURAL NETWORK & NATURAL LANGUAGE UNDERSTANDING**
- **Artificial Neural Network, Appropriate problems for neural network learning, Characteristics of the problems, Basic understanding of neural networks, A single neuron,** Activation Functions, Architectures of neural networks, Feedforward neural network, Single-Layer feedforward architecture, Multiple-Layer feedforward architecture, Types of feedforward networks, multi-layer perceptron, Training MLP: The back-propagation algorithm, Step 1: Forward propagation, Step 2: Back propagation and weight updation, Process of learning in neural network, Recurrent or feedback architecture, Mesh Architectures, GRADIENT-DESCENT (training examples, η), Stochastic GRADIENTDESCENT (training examples, η), Multilayer networks and Backpropagation algorithm, The Backpropagation algorithm, Natural language processing, Classical NLP, Feed-forward networks, Recurrent neural networks and recursive networks, Features for NLP problems, Framenet Vs. Wordnet, Features for text, Features for word relations, NGRAM features, Some terminologies.

- **Artificial Neural Networks**
- **Biological Motivation**
- The study of artificial neural networks (ANNs) has been inspired by the observation that biological learning systems are built of very complex webs of interconnected Neurons
- Human information processing system consists of brain neuron: basic building block cell that communicates information to and from various parts of body
- Simplest model of a neuron: considered as a threshold unit –a processing element (PE)
- Collects inputs & produces output if the sum of the input exceeds an internal threshold value






synapse
axon
nucleus
cell body
dendrites

Facts of Human Neurobiology

- Number of neurons ~ 1011
- Connection per neuron ~ 10 4 – 5
- Neuron switching time ~ 0.001 second or 10 -3
- Scene recognition time ~ 0.1 second
- 100 inference steps doesn’t seem like enough
- Highly parallel computation based on distributed representation

Properties of Neural Networks

- Many neuron-like threshold switching units
- Many weighted interconnections among units
- Highly parallel, distributed process
- Emphasis on tuning weights automatically
- Input is a high-dimensional discrete or real-valued (e.g, sensor input)

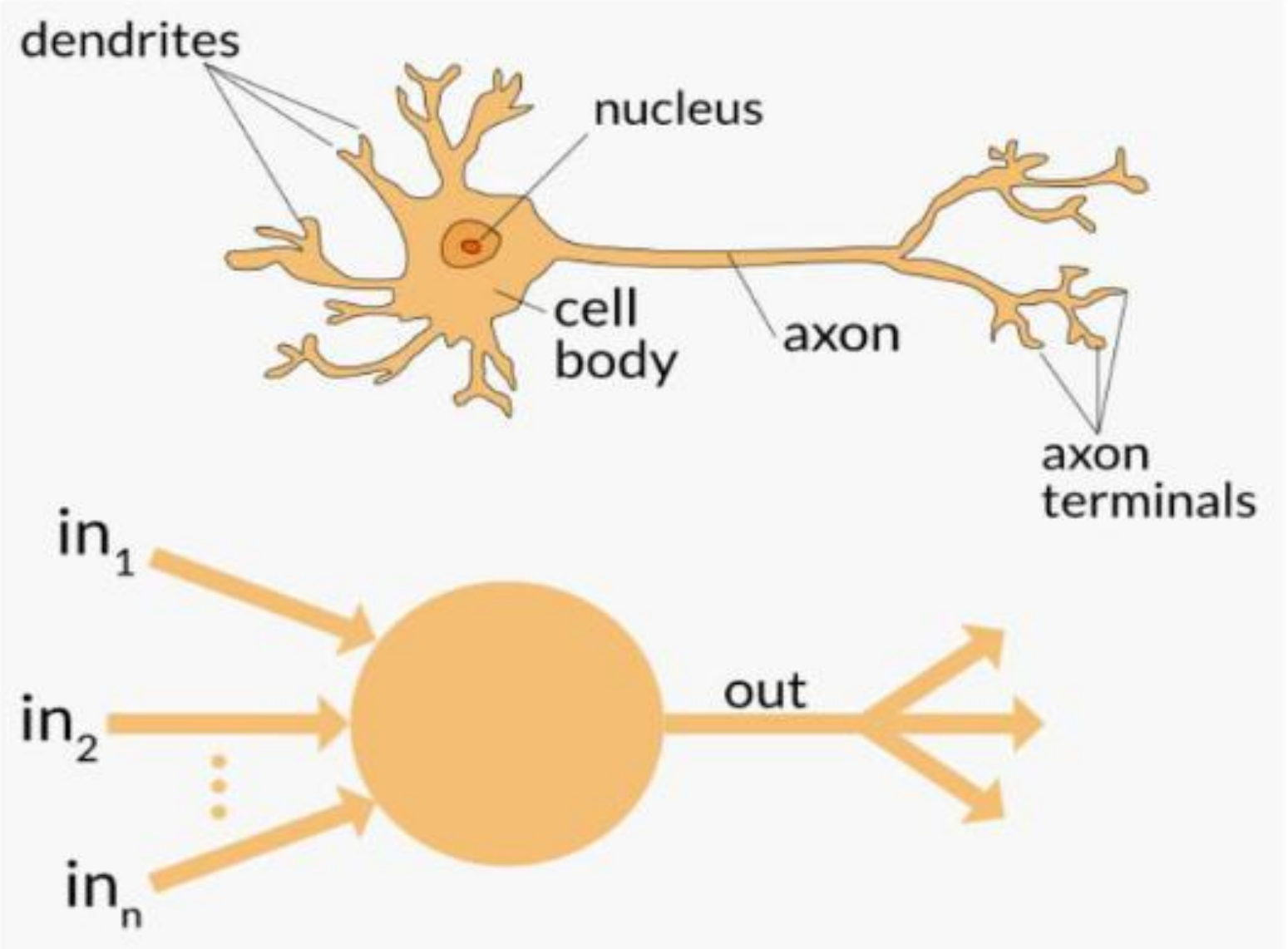
When to consider Neural Networks ?

- Input is a high-dimensional discrete or real-valued (e.g., sensor input)
- Output is discrete or real-valued
- Output is a vector of values
- Possibly noisy data
- Form of target function is unknown
- Human readability of result is unimportant

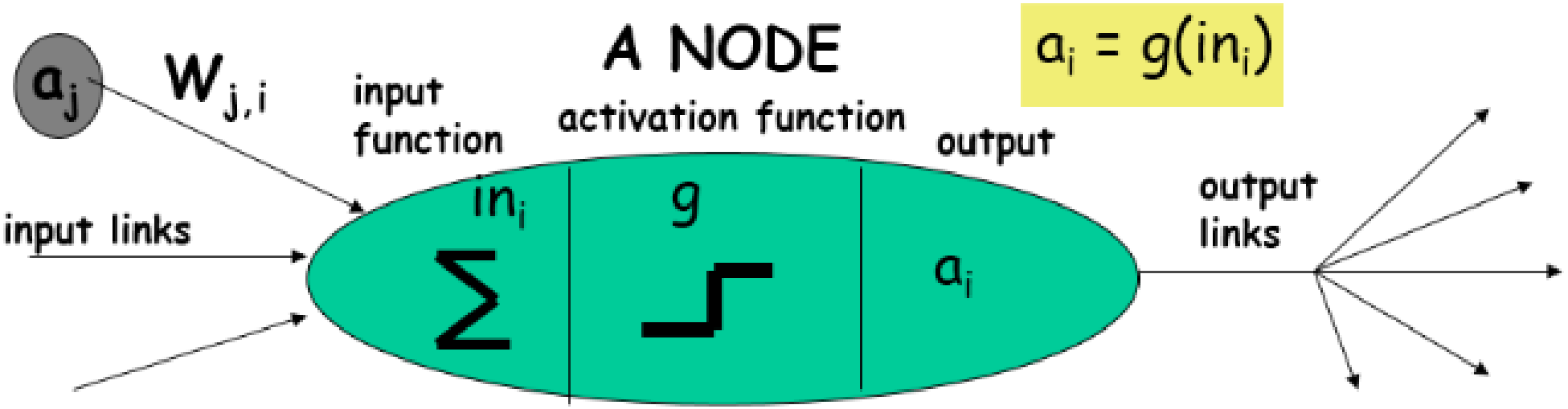
Examples:

1. Speech phoneme recognition
2. Image classification
3. Financial perdition

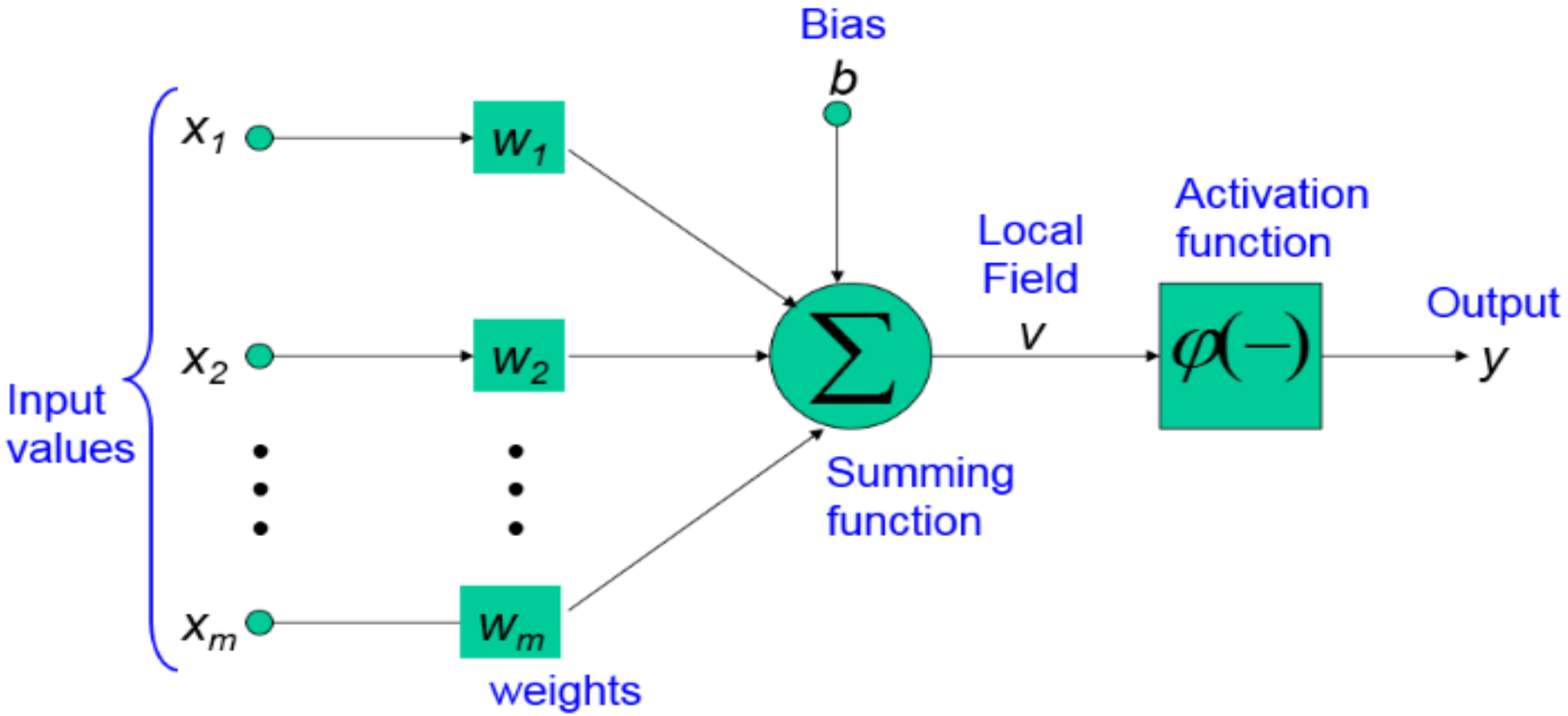
Neuron


dendrites
nucleus
cell body
axon
axon terminals
in1
in2
inn
out

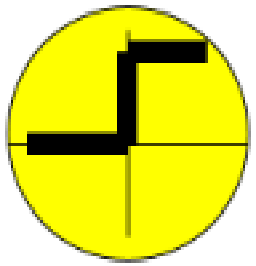
Neuron


a_j
$W_{j,i}$
input function
A NODE
activation function
output
$a_i = g(in_i)$
input links
in_i
g
a_i
output links

Neuron


Bias
b
x_1
x_2
x_m
w_1
w_2
w_m
Input values
weights
Summing function
Local Field
v
Activation function
$\varphi(-)$
Output
y

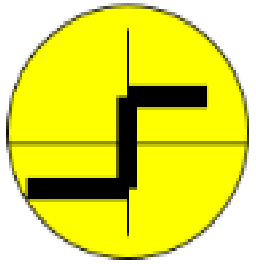
Neuron



Step function

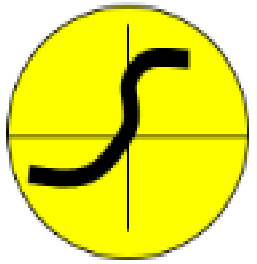
(Linear Threshold Unit)

$$step(x) = 1, \text{ if } x >= threshold$$
$$0, \text{ if } x < threshold$$



Sign function

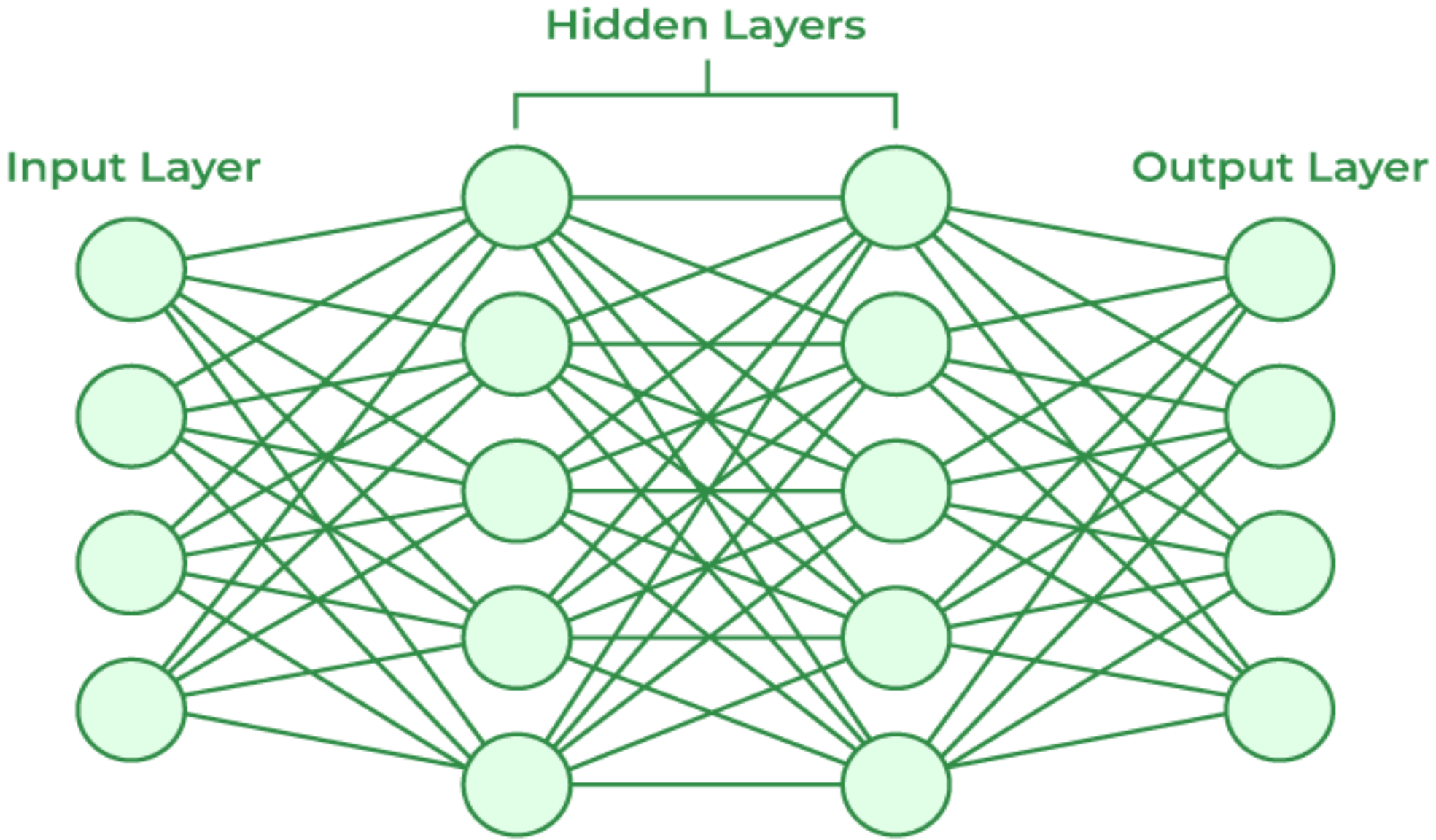
$$sign(x) = +1, \text{ if } x >= 0$$
$$-1, \text{ if } x < 0$$



Sigmoid function

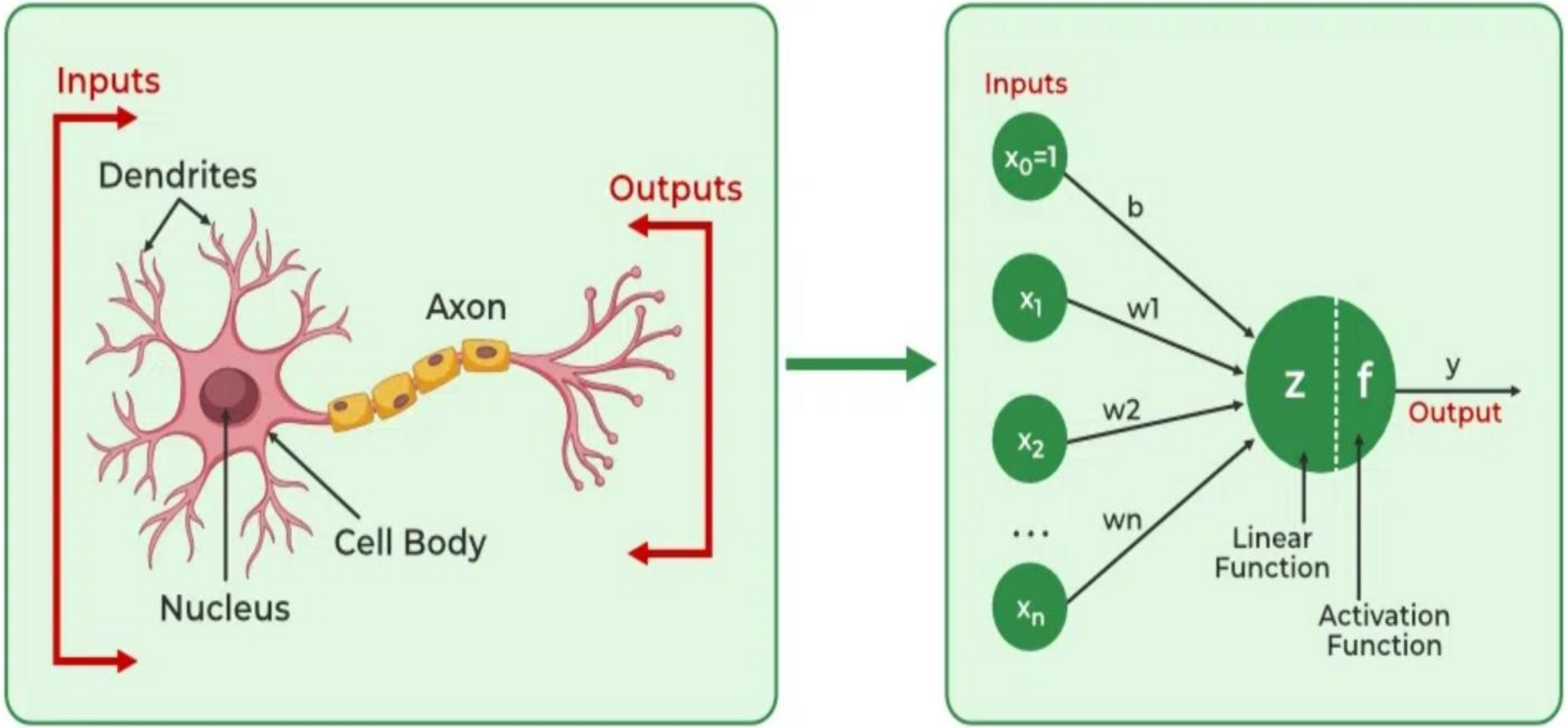
$$sigmoid(x) = 1/(1+e^{-x})$$

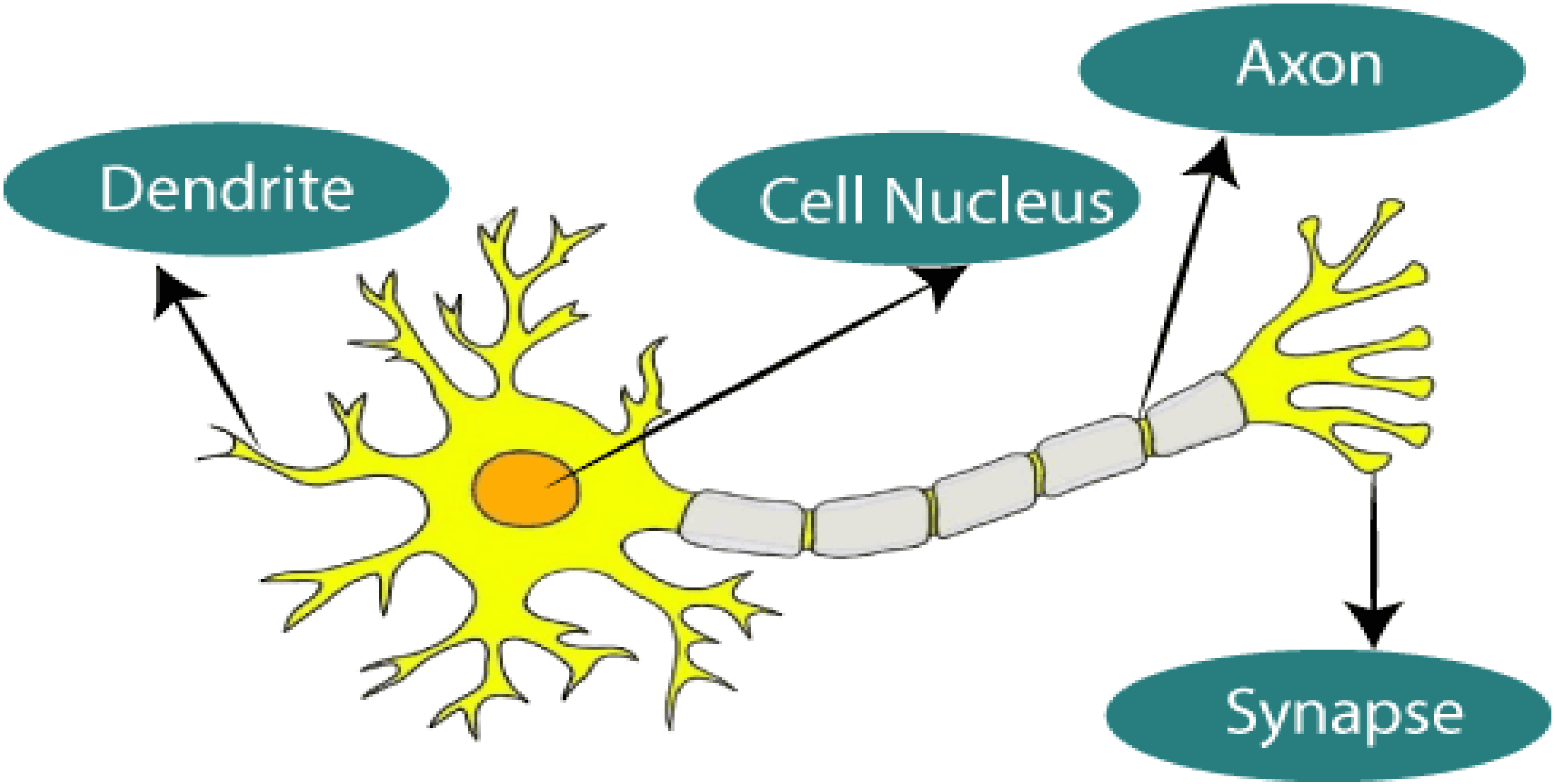
- Artificial Neural Networks contain artificial neurons which are called **units**.
- These units are arranged in a series of layers that together constitute the whole Artificial Neural Network in a system.
- A layer can have only a dozen units or millions of units as this depends on how the complex neural networks will be required to learn the hidden patterns in the dataset.
- Commonly, Artificial Neural Network has an input layer, an output layer as well as hidden layers.
- The **input layer receives data from the outside world which the neural network needs to analyze or learn about.**


Hidden Layers
Input Layer
Output Layer

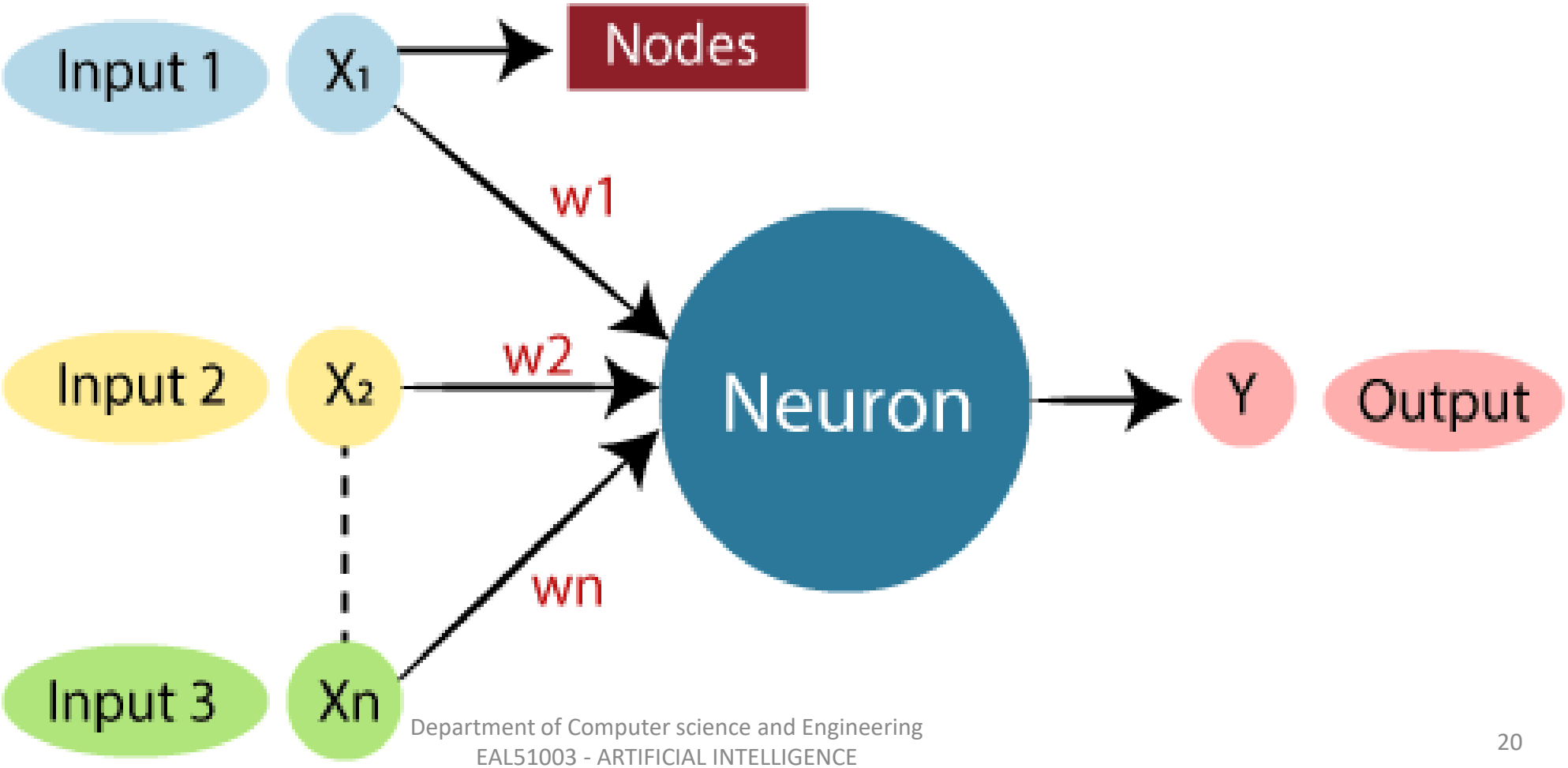
- **Neural Networks Architecture**
- The structures and operations of human neurons serve as the basis for artificial neural networks. It is also known as neural networks or neural nets.
- The input layer of an artificial neural network is the first layer, and it **receives input from external sources** and releases it to the hidden layer, which is the second layer.
- In the hidden layer, each neuron receives input from the **previous layer neurons, computes the weighted sum**, and sends it to the neurons in the next layer.
- These connections are weighted means effects of the inputs from the previous layer are optimized more or less by assigning different-different weights to each input and it is adjusted during the training process by optimizing these weights for improved model performance.

Biological Neuron	Artificial Neuron
Dendrite	Inputs
Cell nucleus or Soma	Nodes
Synapses	Weights
Axon	Output


Inputs
Dendrites
Outputs
Axon
Cell Body
Nucleus
Inputs
$x_0=1$
b
x_1
w1
x_2
w2
...
wn
x_n
z
f
y
Output
Linear Function
Activation Function


Axon
Dendrite
Cell Nucleus
Synapse

- **The given figure illustrates the typical diagram of Biological Neural Network.**
- **The typical Artificial Neural Network looks something like the given figure.**

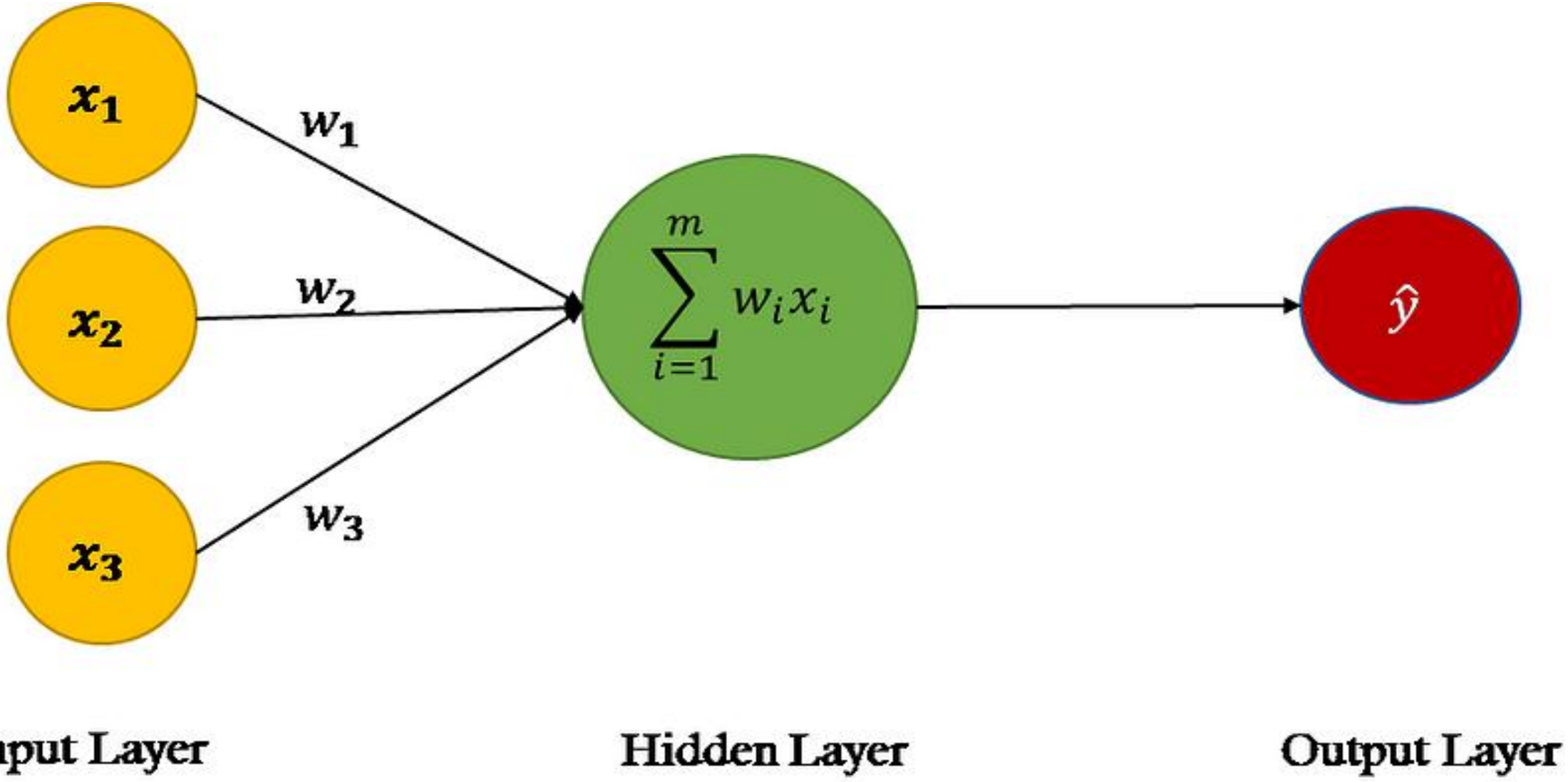


- There are around 1000 billion neurons in the human brain.
- Each neuron has an association point somewhere in the range of 1,000 and 100,000.
- In the human brain, data is stored in such a manner as to be distributed, and we can extract more than one piece of this data when necessary from our memory parallelly.
- We can say that the human brain is made up of incredibly amazing parallel processors.

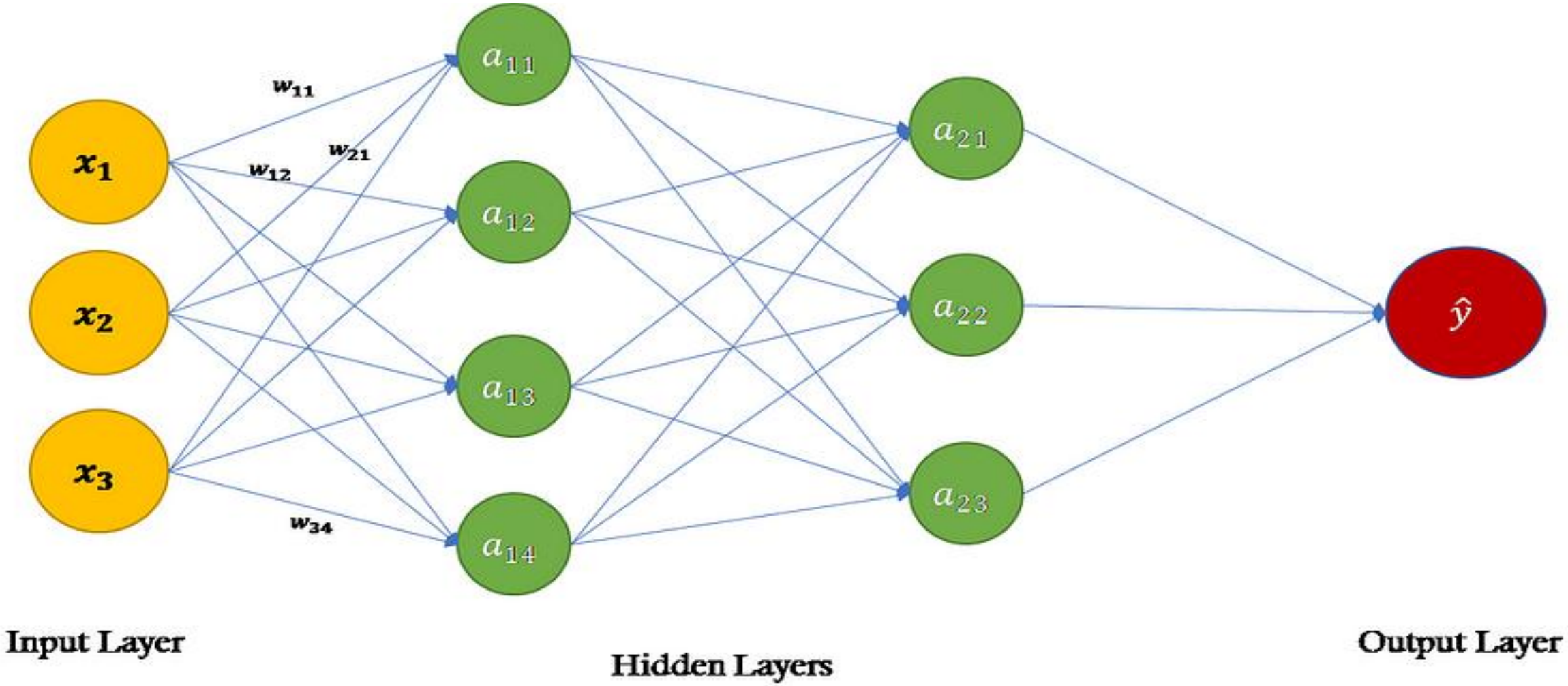
Characteristics of Artificial Neural Networks

- An Artificial Neural Network consists of large number of "neuron" like processing elements.
- All these processing elements have a large number of weighted connections between them.
- The connections between the elements provide a distributed representation of data.
- A Learning Process is implemented to acquire knowledge.

Power of a Single Neuron



- A Neural Network is combinations of **basic Neurons** — also called **perceptrons** (A basic Unit shown in the above diagram- *green circle in middle*) arranged in multiple layers as a network (*below diagram*).



Multilayer Perceptron

HINDUSTAN
INSTITUTE OF TECHNOLOGY & SCIENCE
(DEEMED TO BE UNIVERSITY)

EAL51501 – ARTIFICIAL INTELLIGENCE
B.Tech[AIML] – III Semester

K.Kowsalya
Assistant Professor (SS)
School of Computing Sciences,
Department of Computer Science and Engineering

- **What is an activation function and why use them?**
- The activation function decides whether a neuron should be activated or not by calculating the weighted sum and further adding bias to it. The purpose of the activation function is to introduce non-linearity into the output of a neuron.
- **Explanation:** We know, the neural network has neurons that work in correspondence with *weight, bias,* and their respective activation function.
- **In a neural network, we would update the weights and biases of the neurons on the basis of the error at the output. This process is known as *back-propagation*.**
- Activation functions make the back-propagation possible since the gradients are supplied along with the error to update the weights and biases.

Why we use Activation functions with Neural Networks?

It is used to determine the output of neural network like yes or no.

It maps the resulting values in between 0 to 1 or -1 to 1 etc.

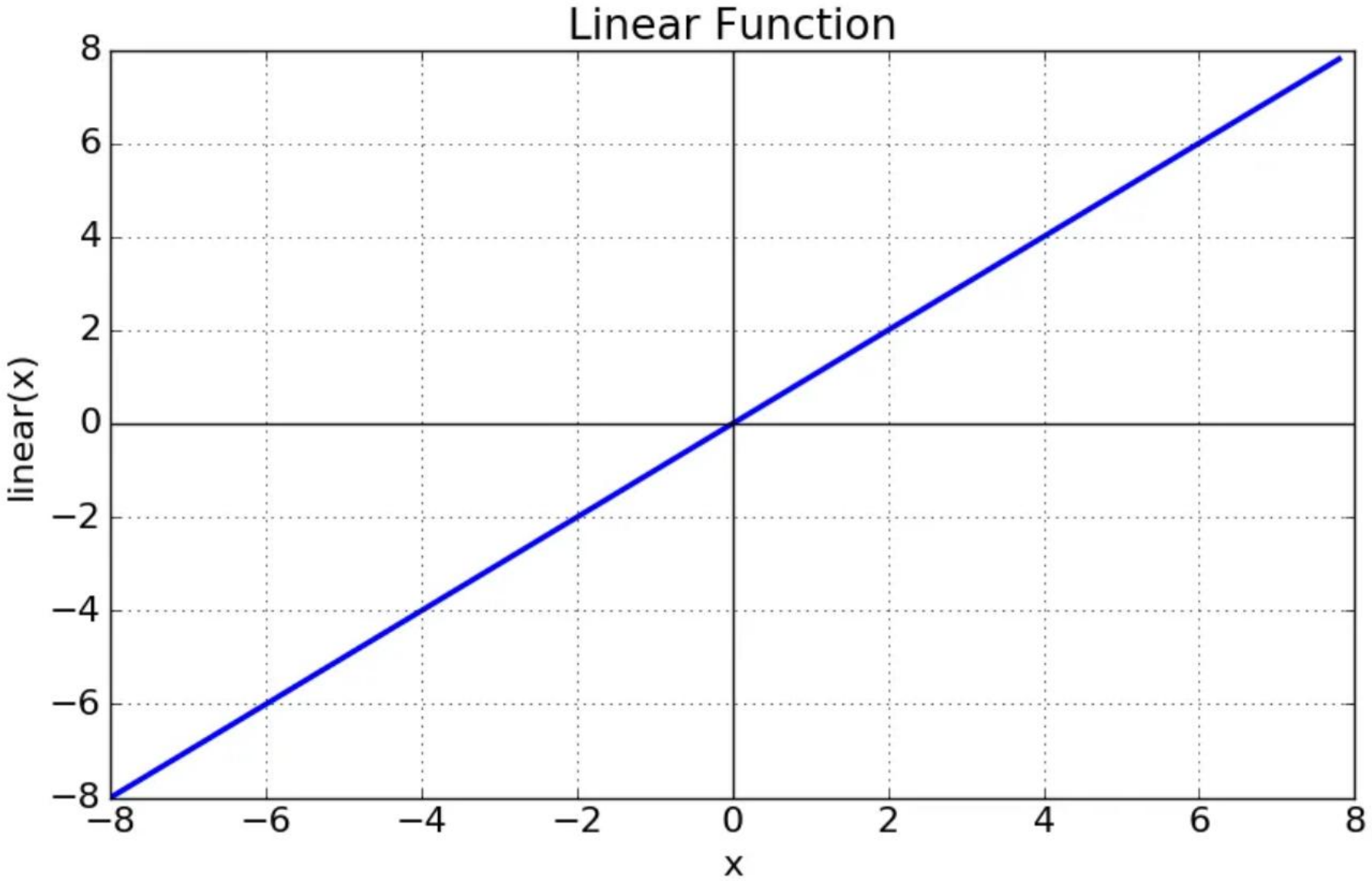
(depending upon the function).

The Activation Functions can be basically divided into 2 types-

1.Linear Activation Function

2.Non-linear Activation Functions

- **Linear or Identity Activation Function**
- As you can see the function is a line or linear. Therefore, the output of the functions will not be confined between any range.



- **Equation :** f(x) = x
- **Range :** (-infinity to infinity)
- It doesn’t help with the complexity or various parameters of usual data that is fed to the neural networks.

- **Non-linear Activation Function**
- The Nonlinear Activation Functions are the most used activation functions. Nonlinearity helps to makes the graph look something like this
- It makes it easy for the model to generalize or adapt with variety of data and to differentiate between the output.
- The main terminologies needed to understand for nonlinear functions are:
- **Derivative or Differential:** Change in y-axis w.r.t. change in x-axis.It is also known as slope.
- **Monotonic function:** A function which is either entirely non-increasing or non-decreasing.

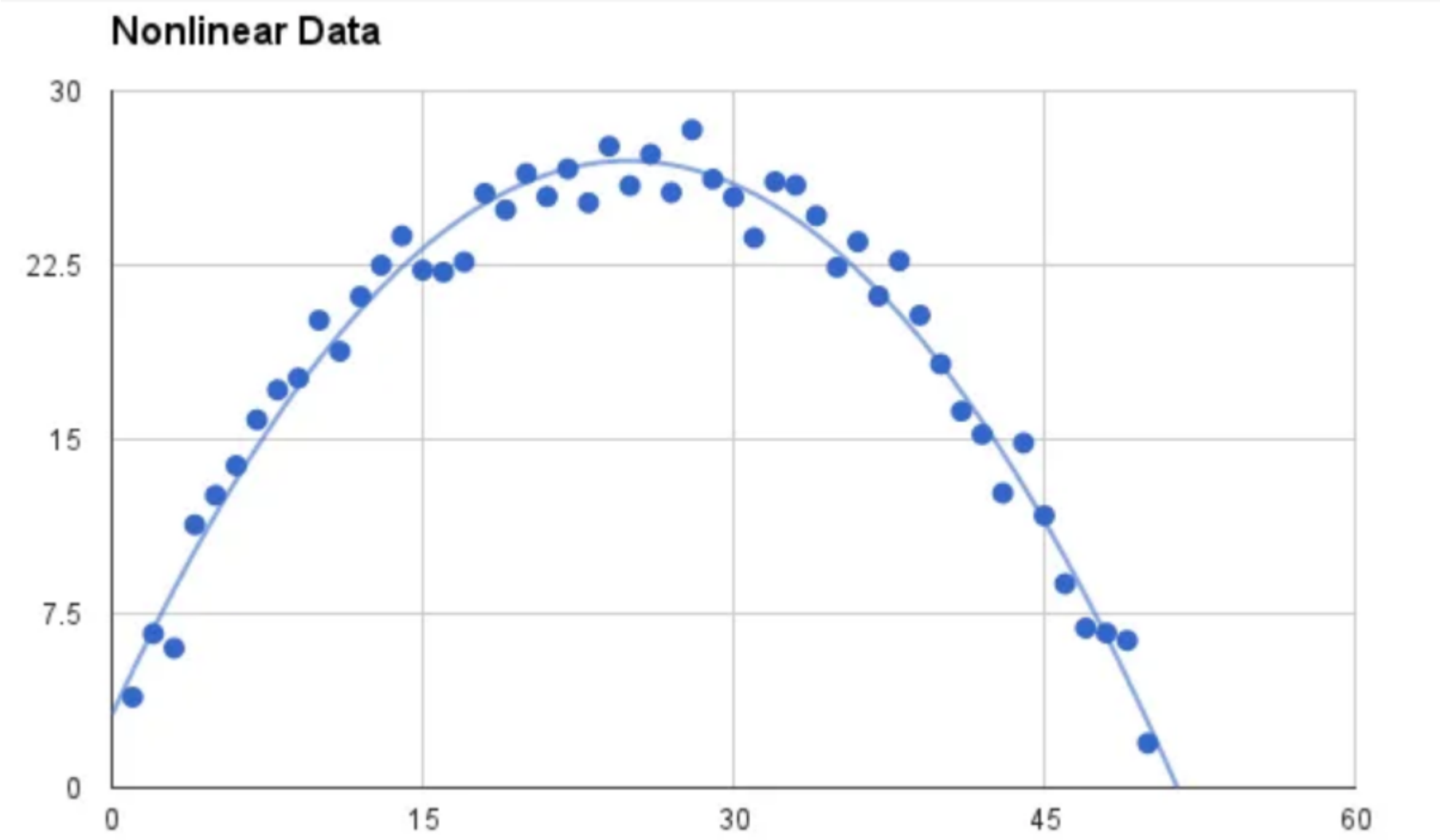


Fig: Non-linear Activation Function

- The Nonlinear Activation Functions are mainly divided on the basis of their **range or curves**-
- **1. Sigmoid or Logistic Activation Function**
- The Sigmoid Function curve looks like a S-shape.

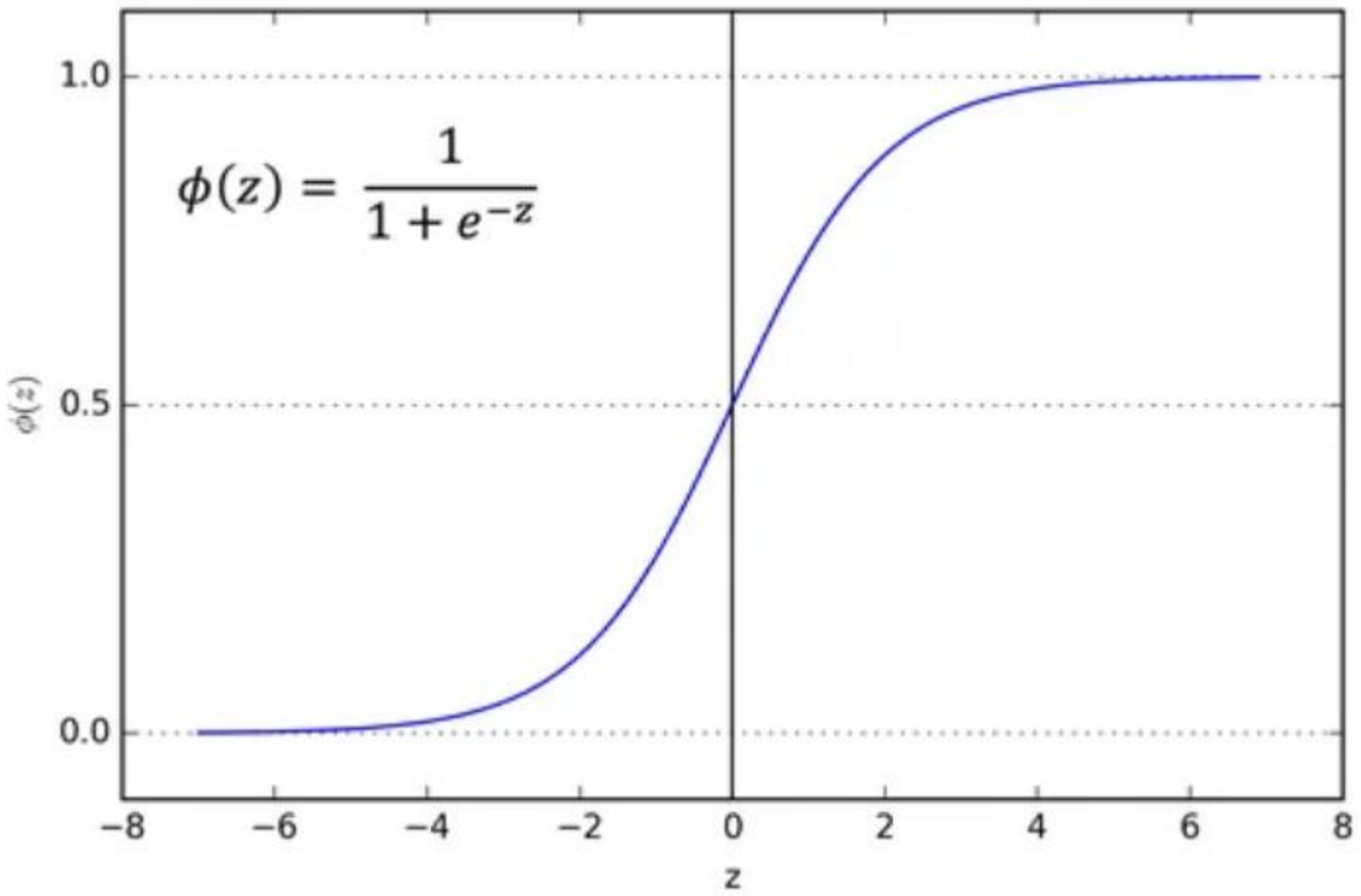


Fig: Sigmoid Function

- It is a function which is plotted as **'S'** shaped graph.
- **Equation :** $A = 1/(1 + e^{-x})$
- **Nature :** Non-linear. Notice that X values lies between -2 to 2, Y values are very steep. This means, small changes in x would also bring about large changes in the value of Y.
- **Value Range :** 0 to 1
- **Uses :** Usually used in output layer of a binary classification, where result is either 0 or 1, as value for sigmoid function lies between 0 and 1 only so, result can be predicted easily to be ***1*** if value is greater than **0.5** and ***0*** otherwise.

- **2. Tanh or hyperbolic tangent Activation Function**
- tanh is also like logistic sigmoid but better. The range of the tanh function is from (-1 to 1). tanh is also sigmoidal (s - shaped).

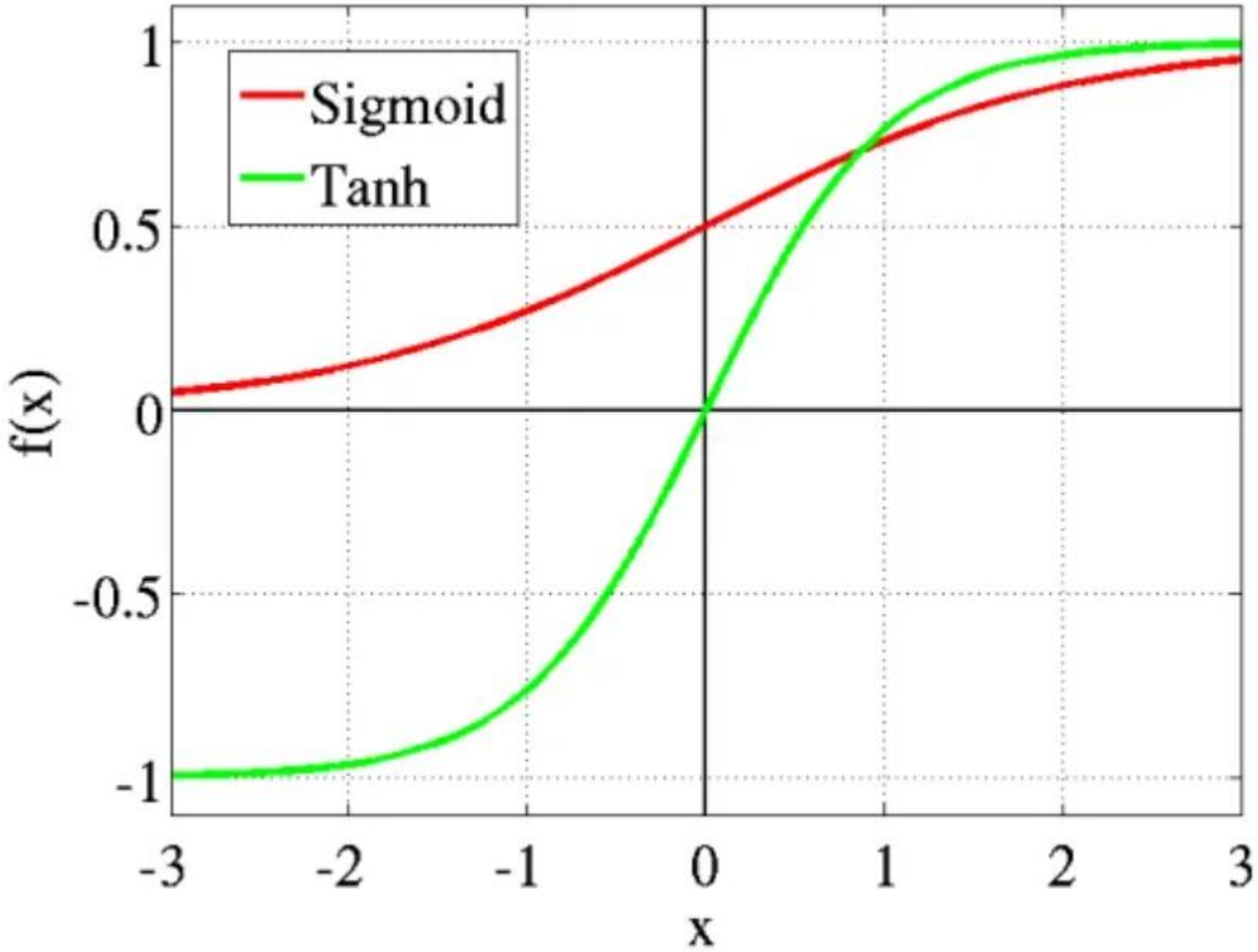
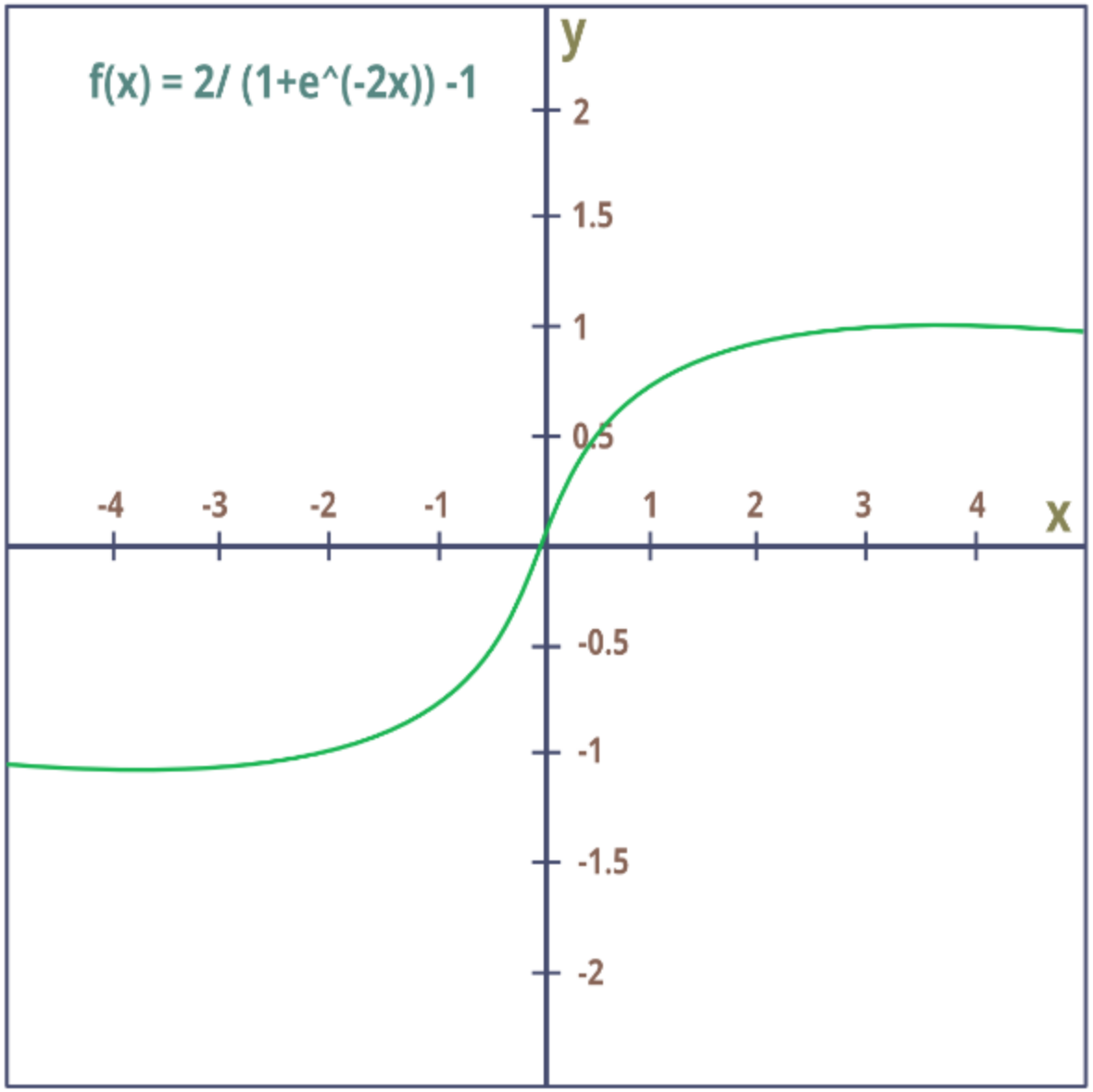


Fig: tanh v/s Logistic Sigmoid


f(x) = 2/ (1+e^(-2x)) -1
y
2
1.5
1
0.5
-4
-3
-2
-1
1
2
3
4
x
-0.5
-1
-1.5
-2

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

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

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

3. ReLU (Rectified Linear Unit) Activation Function

The ReLU is the most used activation function in the world right now.Since, it is used in almost all the convolutional neural networks or deep learning.

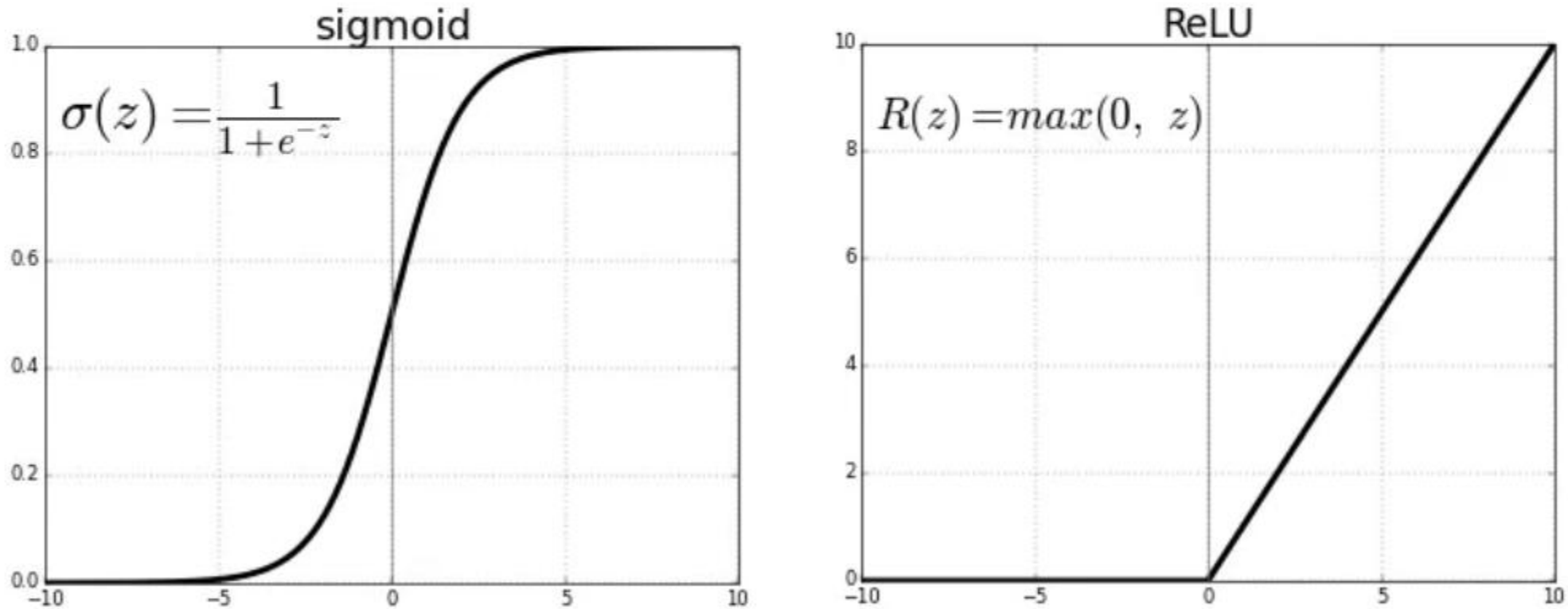
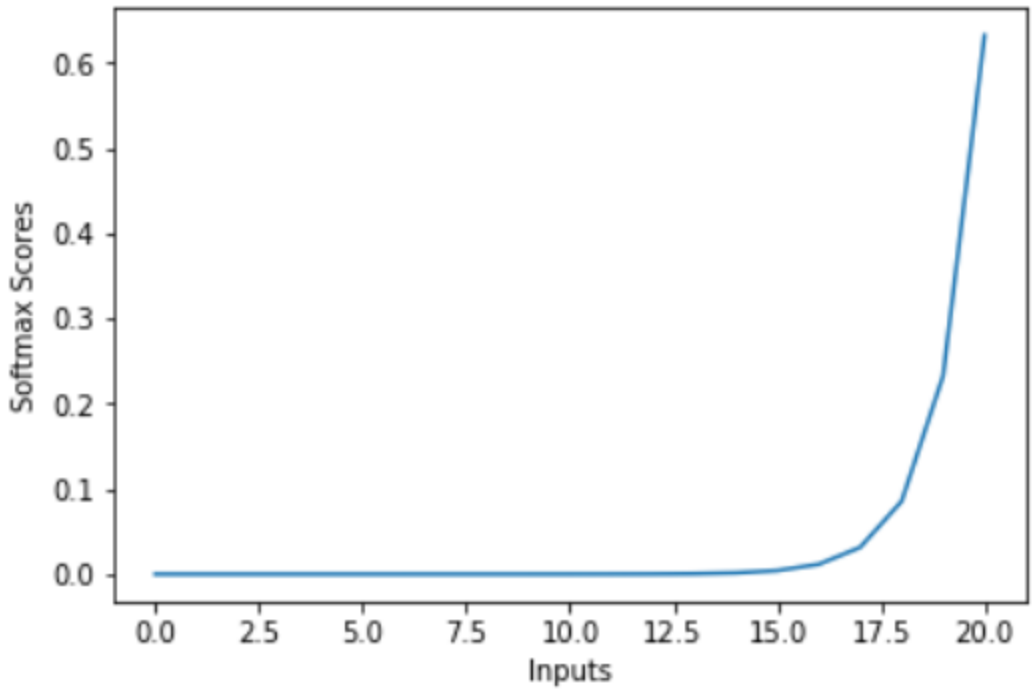


Fig: ReLU v/s Logistic Sigmoid

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

In simple words, RELU learns *much faster* than sigmoid and Tanh function.

Softmax Function



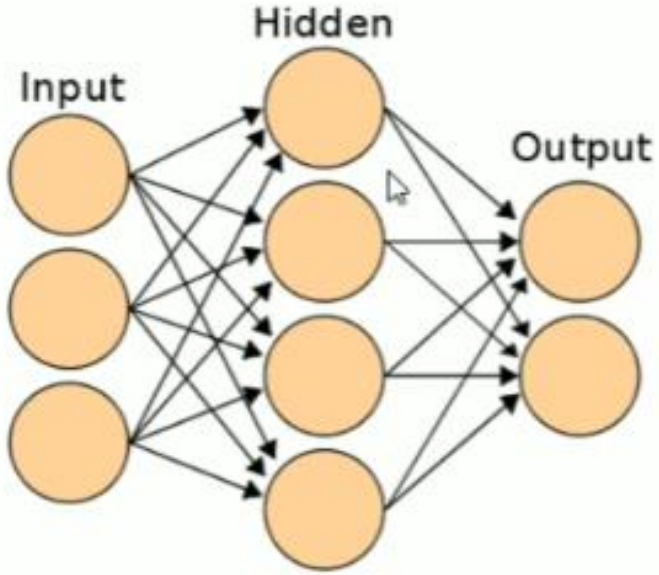
The softmax function is also a type of sigmoid function but is handy when we are trying to handle multi- class classification problems.

- **Nature :-** non-linear
- **Uses :-** Usually used when trying to handle multiple classes. the softmax function was commonly found in the output layer of image classification problems.The softmax function would squeeze the outputs for each class between 0 and 1 and would also divide by the sum of the outputs.
- **Output:-** The softmax function is ideally used in the output layer of the classifier where we are actually trying to attain the probabilities to define the class of each input.
- The basic rule of thumb is if you really don't know what activation function to use, then simply use *RELU* as it is a general activation function in hidden layers and is used in most cases these days.
- If your output is for binary classification then, *sigmoid function* is very natural choice for output layer.
- If your output is for multi-class classification then, Softmax is very useful to predict the probabilities of each classes.

ANN NETWORK ARCHITECTURE

Interconnections:

- Interconnection can be defined as the way processing elements (Neurons) in ANN are connected to each other.
- The arrangements of these processing elements and geometry of interconnections are very essential in ANN.
- Have Input layer and output layer where input layer buffers the input signal and output layer generates the output of the network.
- The third layer is the **Hidden layer**. The neurons are hidden from the people who are interfacing with the system and acts as a black box to them.
- On **increasing the hidden layers with neurons**, the system's **computational and processing power can be increased** but the **training phenomena of the system gets more complex at the same time**.



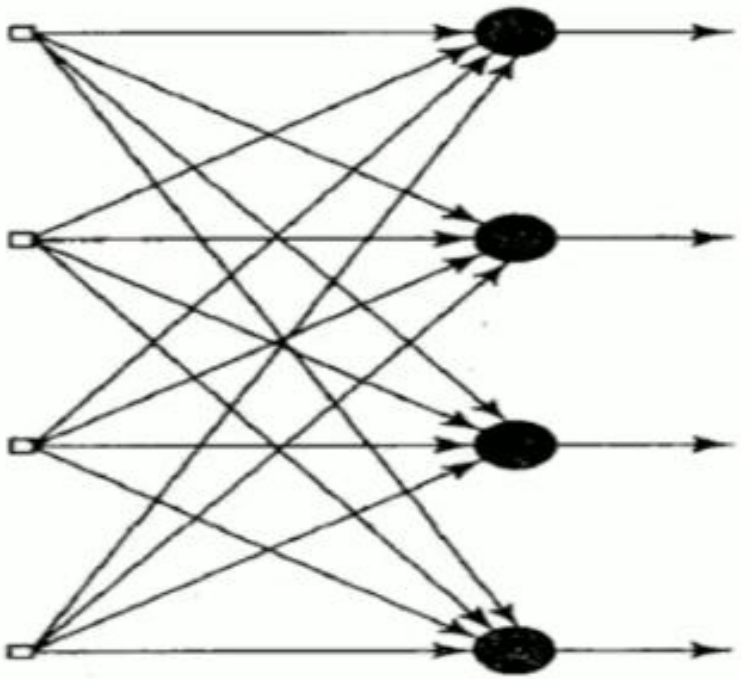
NETWORK ARCHITECTURE TYPES

There exist **Three basic types** of neuron connection architecture :

1. **Single-layer feed forward networks**
2. **Multilayer feed forward networks**
3. **Recurrent networks**

Single-layer feed forward networks:

- An input layer of **source node projects on to an output layer** of neurons (Computational nodes)-but not vice versa.
- Strictly speaking, it is **feed-forward or acyclic** type of network.

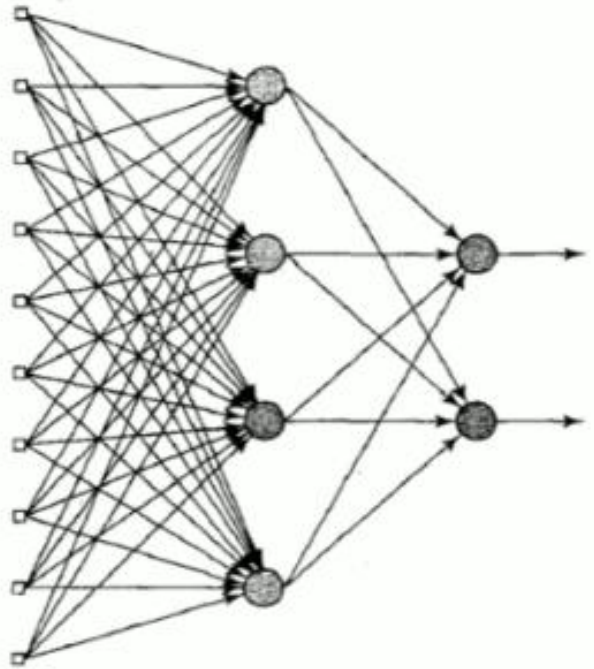


Feedforward or acyclic network with a single layer of neurons.

- This network is called **single-layer network**, where single-layer **refers to output layer of computation nodes**(neurons).
- We **do not count input layer** of source nodes, as computations are **not performed**.

Multi-layer feed forward networks:

- Presence of one or more hidden layers.
- Computational nodes are called hidden neurons or hidden units.
- Hidden neurons **intervene between external input and the network output** in some useful manner.
- Existence of one or more hidden layers enable the network to be **computationally stronger**.
- Referred as 10-4-2 network as it has 10 source nodes,4 hidden neurons, and 2 output neurons.
- In a **fully connected network**, every node in each layer of network is connected to every Other node in the adjacent forward layer.

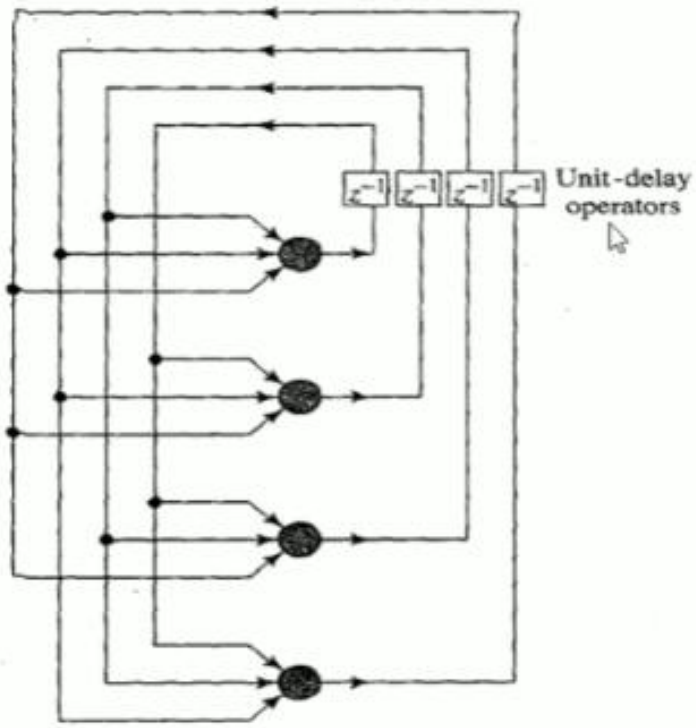


Fully connected feedforward or acyclic network with one hidden layer and one output layer.

a feedforward network with m source nodes, h_1 neurons in the first hidden layer, h_2 neurons in the second hidden layer, and q neurons in the output layer is referred to as an m-h_1-h_2-q network.

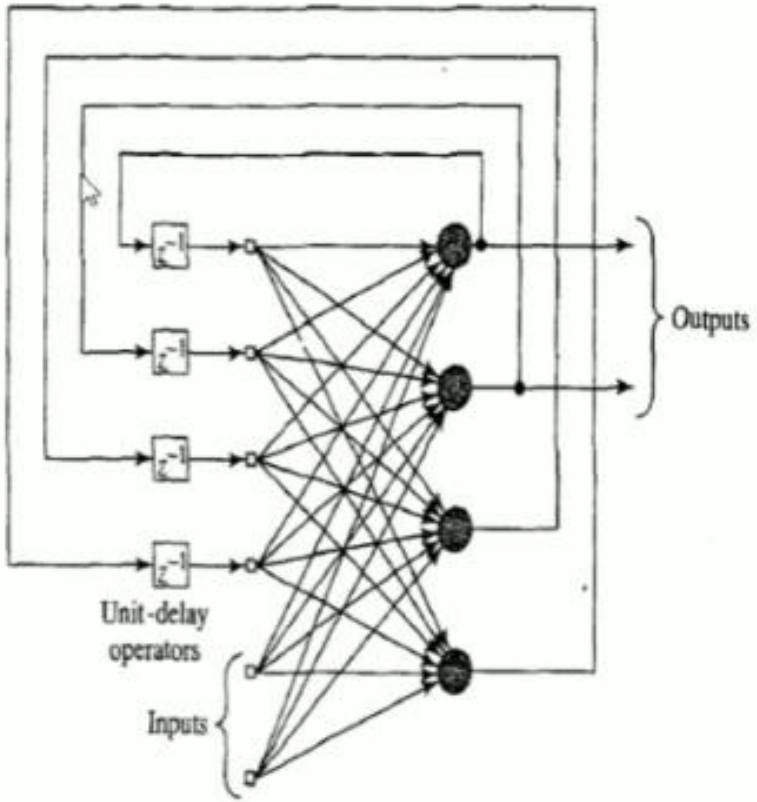
Recurrent networks

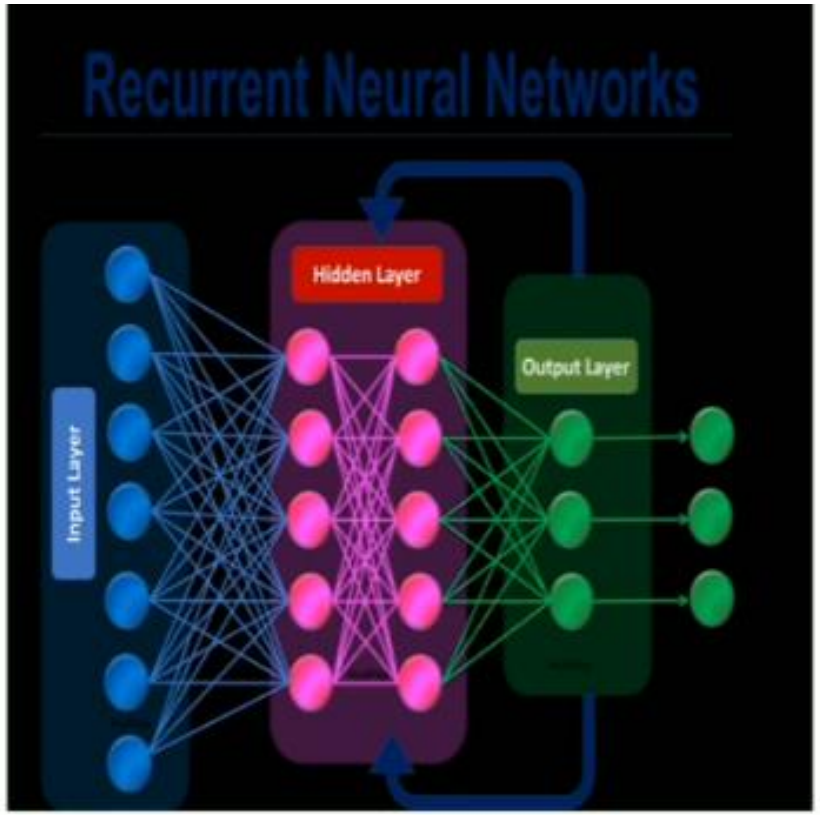
- A recurrent neural network is different from feedforward neural network.
- A RNN have atleast one feedback loop.



Recurrent network with no self-feedback loops and no hidden neurons.

Moreover, the feedback loops involve the use of particular branches composed of *unit-delay elements* (denoted by z^{-1}), which result in a nonlinear dynamical behavior, assuming that the neural network contains nonlinear units.



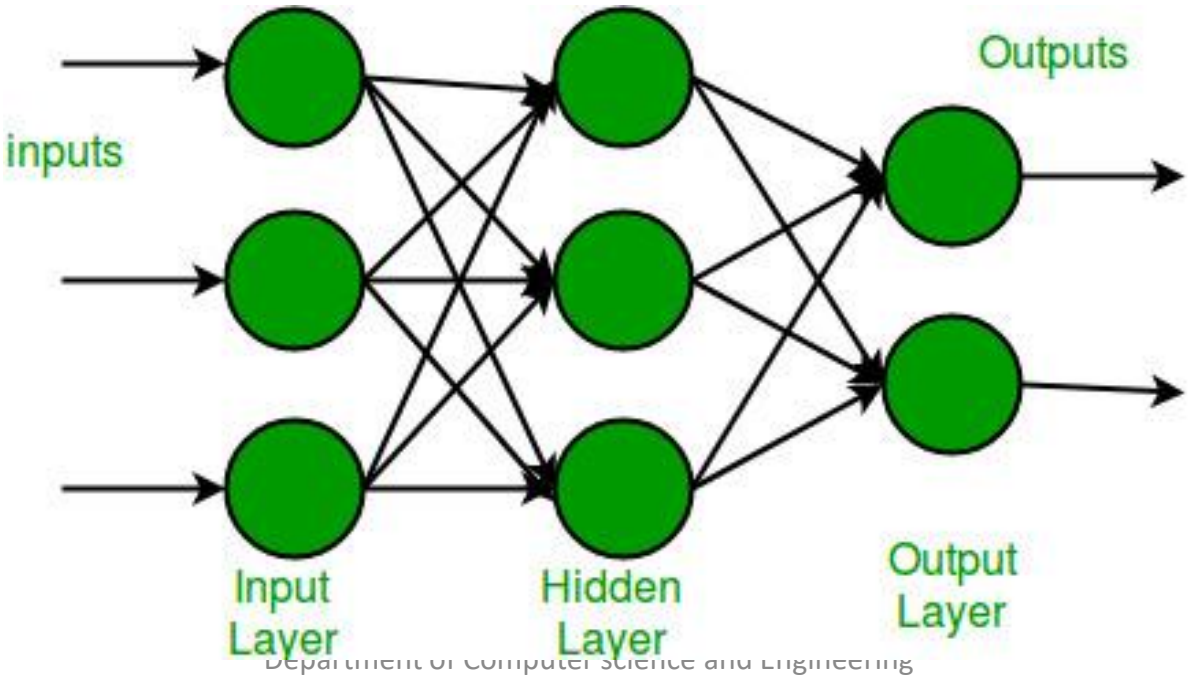


Recurrent network with hidden neurons.

Presence of feedback loops in recurrent structures has a profound impact on learning capability of the network and its performance.

- **Multi-layer Perceptron**
- Multi-layer perception is also known as MLP.
- It is fully connected dense layers, which transform any input dimension to the desired dimension
- A multi-layer perception is a neural network that has multiple layers.
- To create a neural network we combine neurons together so that the outputs of some neurons are inputs of other neurons.

- A multi-layer perceptron has one input layer and for each input, there is one neuron(or node), it has one output layer with a single node for each output and it can have any number of hidden layers and each hidden layer can have any number of nodes. A schematic diagram of a Multi-Layer Perceptron (MLP) is depicted below.
- It is a neural network where the mapping between inputs and output is non-linear.

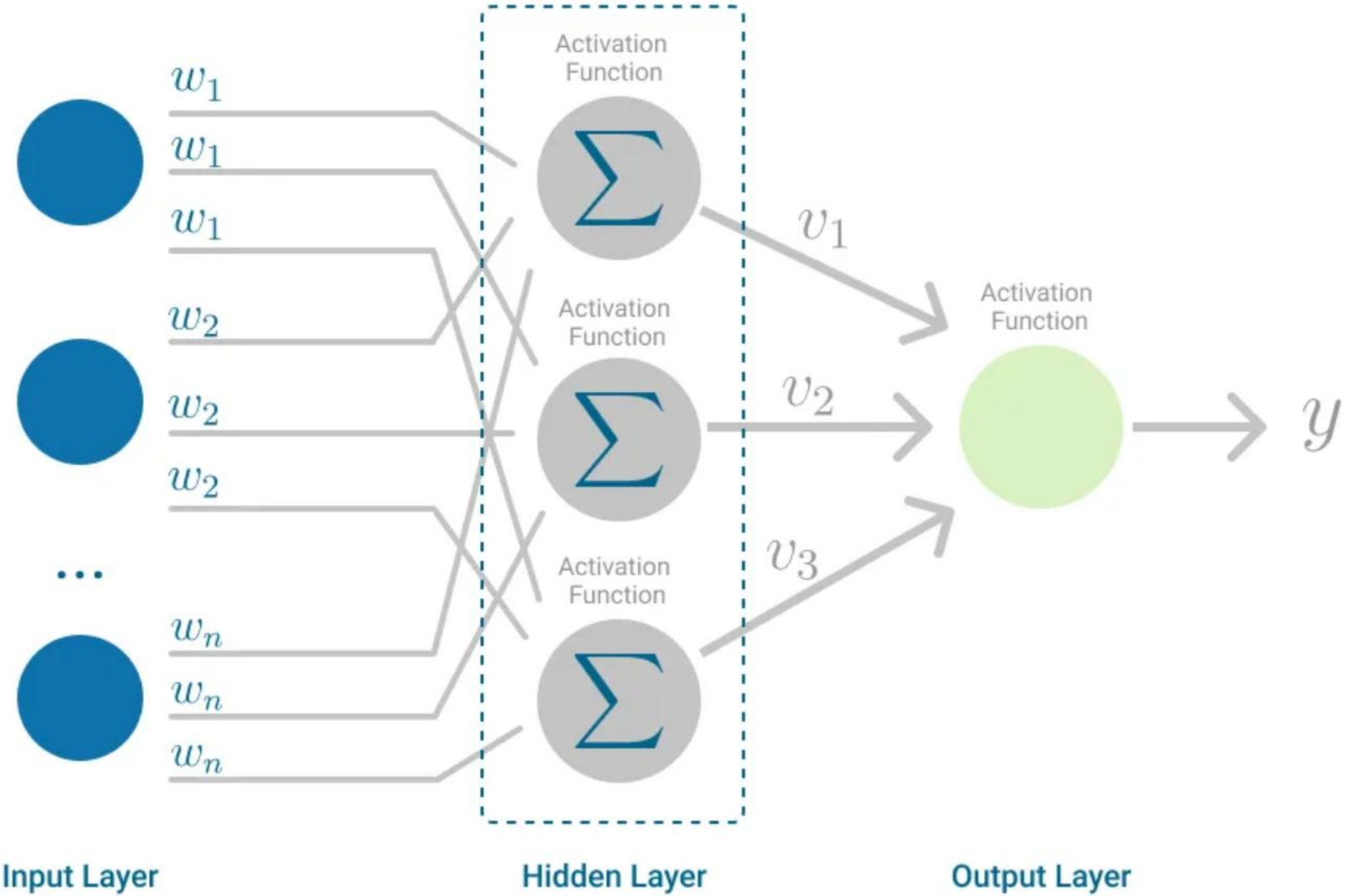


- In the multi-layer perceptron diagram above, we can see that there are three inputs and thus three input nodes and the hidden layer has three nodes. The output layer gives two outputs, therefore there are two output nodes.
- The nodes in the input layer take input and forward it for further process, in the diagram above the nodes in the input layer forwards their output to each of the three nodes in the hidden layer, and in the same way, the hidden layer processes the information and passes it to the output layer.

- Every node in the multi-layer perception uses a sigmoid activation function. The sigmoid activation function takes real values as input and converts them to numbers between 0 and 1 using the sigmoid formula.

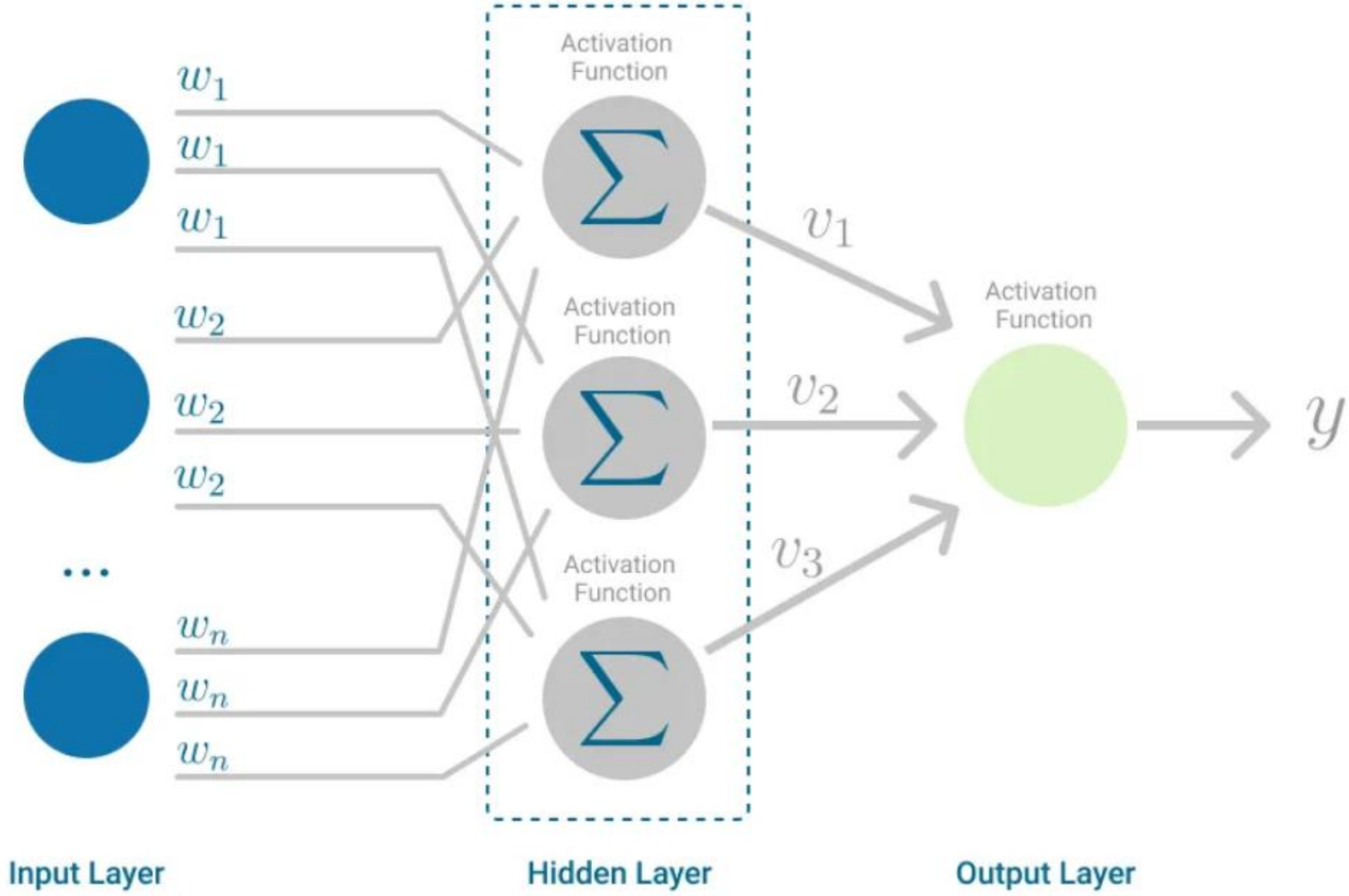
$$(x) = 1/(1 + exp(-x))$$

- A Multilayer Perceptron has input and output layers, and one or more **hidden layers** with many neurons stacked together.
- And while in the Perceptron the neuron must have an activation function that imposes a threshold, like ReLU or sigmoid, neurons in a Multilayer Perceptron can use any arbitrary activation function.


Activation Function
Activation Function
Activation Function
Activation Function
w_1
w_1
w_1
w_2
w_2
w_2
w_n
w_n
w_n
v_1
v_2
v_3
y
Input Layer
Hidden Layer
Output Layer

- Multilayer Perceptron falls under the category of feedforward algorithms, because inputs are combined with the initial weights in a weighted sum and subjected to the activation function, just like in the Perceptron. But the difference is that each linear combination is propagated to the next layer.
- Each layer is *feeding* the next one with the result of their computation, their internal representation of the data. This goes all the way through the hidden layers to the output layer.
- But it has more to it.
- If the algorithm only computed the weighted sums in each neuron, propagated results to the output layer, and stopped there, it wouldn't be able to *learn* the weights that minimize the cost function. If the algorithm only computed one iteration, there would be no actual learning.

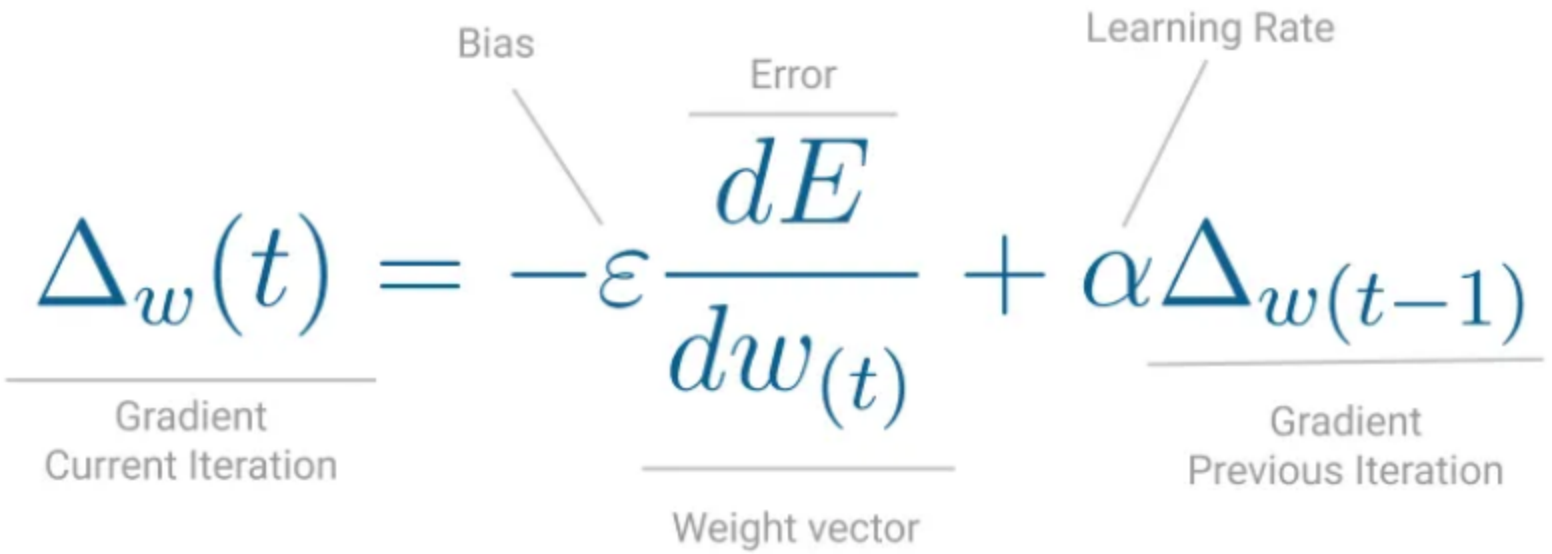
- **Backpropagation**
- Backpropagation is the learning mechanism that allows the Multilayer Perceptron to iteratively adjust the weights in the network, with the goal of minimizing the cost function.
- There is one hard requirement for backpropagation to work properly. The function that combines inputs and weights in a neuron, for instance the weighted sum, and the threshold function, for instance ReLU, must be differentiable. These functions must have a **bounded derivative**, because Gradient Descent is typically the optimization function used in MultiLayer Perceptron.



1. Feedforward | Mean Squared Error (MSE) computed

2. Backpropagation | Gradient is computed

In each iteration, after the weighted sums are forwarded through all layers, the gradient of the **Mean Squared Error** is computed across all input and output pairs. Then, to propagate it back, the weights of the first hidden layer are updated with the value of the gradient. That's how the weights are propagated back to the starting point of the neural network!



One iteration of Gradient Descent. (Image by author)

- This process keeps going until gradient for each input-output pair has converged, meaning the newly computed gradient hasn't changed more than a specified *convergence threshold*, compared to the previous iteration.

FEEDFORWARD NEURAL NETWORK

Neural networks feedforward, also known as **multi-layered networks of neurons**, are called "feedforward," where information flows in one direction from the input to the output layer without looping back. It is composed of three types of layers:

- **Input Layer:**
  The input layer accepts the input data and passes it to the next layer.

- **Hidden Layers:**

One or more hidden layers that process and transform the input data. Each hidden layer has a set of neurons connected to the neurons of the previous and next layers. These layers use activation functions, such as ReLU or sigmoid, to introduce non-linearity into the network, allowing it to learn and model more complex relationships between the inputs and outputs.

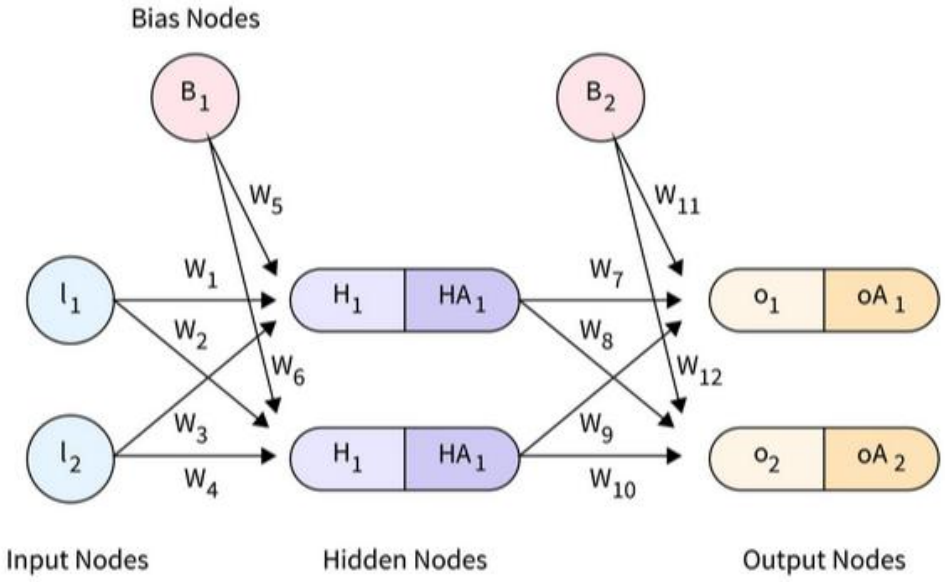
Output Layer:

The output layer generates the final output. Depending on the type of problem, the number of neurons in the output layer may vary. For example, in a binary classification problem, it would only have one neuron. In contrast, a multi-class classification problem would have as many neurons as the number of classes.

The purpose of Neural networks feedforward is to **approximate** certain functions. The input to the network is a vector of values, **x**, which is passed through the network, layer by layer, and transformed into an **output**, y. The network's final output predicts the target function for the given input. The network makes this prediction using a set of parameters, **θ (theta)**, adjusted during training to minimize the **error** between the network's predictions and the target function.

The training involves adjusting the θθ (theta) values to minimize errors. This is done by presenting the network with a set of **input-output pairs (also called training data)** and computing the error between the network's prediction and the true output for each pair. This error is then used to compute the **gradient** of the error concerning the parameters, which tells us how to adjust the parameters to reduce the error. This is done using optimization techniques like gradient descent. Once the training process is completed, the network has " learned " the function and can be used to predict new input.

Finally, the network stores this **optimal value** of θθ (theta) in its memory, so it can use it to predict new inputs.



- **I:**

Input node (the starting point for data entering the neural network)

- **W:**

Connection weight (used to determine the strength of the connection between nodes)

- **H:**

Hidden node (a layer within the network that processes input)

- **HA:**

Activated hidden node (the value of the hidden node after passing through a predefined function)

- **O:**

Output node (the final output of the network, calculated as a weighted sum of the last hidden layer)

- **OA:**

Activated output node (the final output of the network after passing through a predefined function)

- **B:**

Bias node (a constant value, typically set to 1.0, used to adjust the output of the network)

Layers in Neural Network Feed-forward

Input Layer

In a Neural network feedforward, the input layer is the **first layer** of the network, and it is responsible for accepting the input data and passing it to the next layer. The input layer does not perform any computations or transformations on the data. It only acts as a placeholder for the input data.

The input layer has several neurons corresponding to the number of features in the input data. For example, if we use an image as input, the number of neurons in the input layer would be the number of **pixels** in the image. Each neuron in the input layer is connected to all the neurons in the next layer.

We can also use the input layer to add information, such as a **bias** term, to the input data. This is done by adding a bias neuron to the input layer, which always outputs 1.

The input layer of a neural network feedforward is simple, and it only has a function to accept the input data and feed it to the next layers. It has no learnable parameters, so it is unnecessary to update those. It only acts as a **starting point** for the neural network to work upon, and the computation starts at the next layers.

Hidden Layer

In a neural network feedforward, a hidden layer refers to one of the layers between the input and output layers. It's called hidden because it doesn't directly interact with the external environment. Instead, it only **receives input** from the input layer or previously hidden layers, then performs internal computations before passing the output to the next layer.

The main function of a hidden layer is to extract **features** and **abstract representations** of the input data. By having multiple hidden layers, a neural network can learn increasingly complex and abstract features of the input data. Each neuron in a hidden layer receives input from the neurons in the previous layer, processes it, and passes it on to the next layer. This way, the hidden layers transform the input data and extract useful features, allowing the network to learn more complex and abstract relationships between the inputs and outputs.

Activation functions are used in hidden layers to introduce **non-linearity** into the network. Common examples of activation functions include ReLU, sigmoid, and tanh. The choice of activation function depends on the specific problem, but ReLU is commonly used in many cases because it tends to work well and improves the training speed.

The number of neurons and layers in the hidden layers is one of the hyperparameters that can be adjusted during the design and training of the network. Generally speaking, the more neurons and layers there are, the more complex and abstract features the network can learn. However, this also increases the risk of **overfitting** and requires more computational power to train the network.

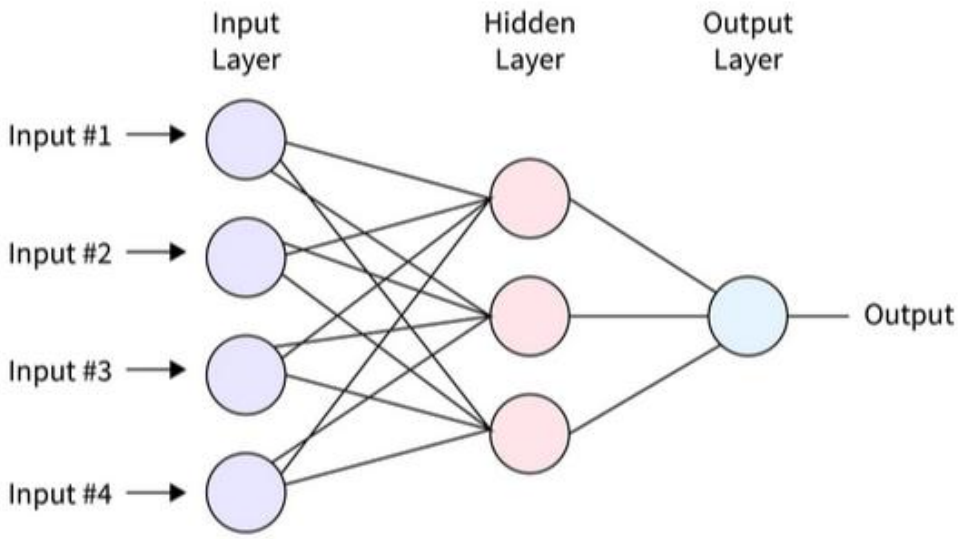
Output Layer

The **output layer** in a neural network feedforward is the **final layer** in the network architecture. Its main function is to generate the network's final output based on the processed input data. The output layer takes the output of the last hidden layer as its input and generates the final output of the network by applying a final set of transformations to this data.

The number of neurons in the output layer depends on the specific problem the network is designed to solve. For example, in a **binary classification problem**, the output layer would typically have a single neuron that generates a probability value between 0 and 1, indicating the probability of the input data belonging to the positive class. Similarly, in a multi-class classification problem, the output layer would have as many neurons as the number of classes. Each neuron would generate a probability value indicating the probability of the input data belonging to each class.

The output layer also has a set of learnable parameters, such as **weights and biases**, that are updated during training to minimize a chosen loss function.

Activation functions are also applied in the output layer, as it depends on the problem. Some common activation functions for the output layer are **sigmoid** for binary classification and **softmax** for multiclass classification.



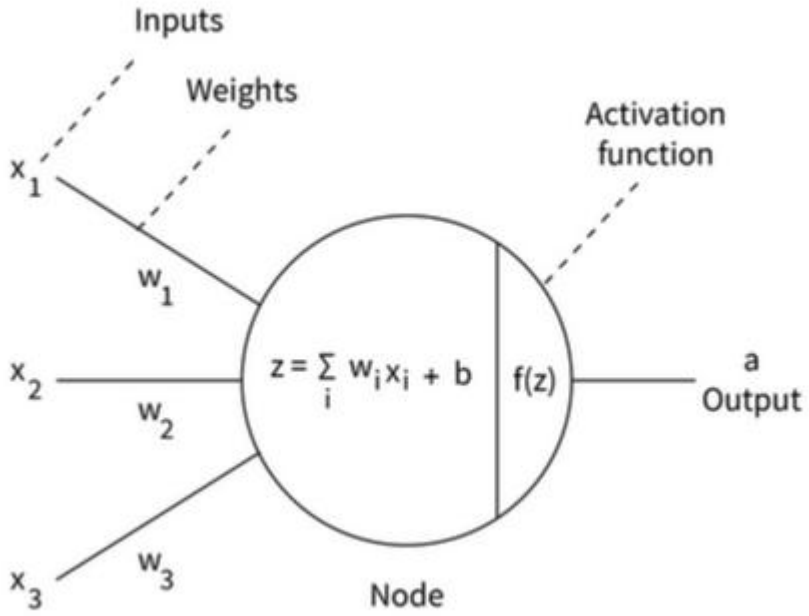
Weights and Biases

In a neural network's feedforward, the weights and biases are the **learnable parameters** updated during training to minimize a chosen loss function. These parameters are specific to each neuron in the network and play a crucial role in determining the network's **final output**.

Weights are the parameters that control the strength of the connection between neurons in different layers. They are used to **scale** the input signal before it is passed through the activation function of a neuron. In other words, the weights determine how much influence a particular input has on the output of a neuron. In a neural network feedforward, weights are typically represented as **matrices**, with one matrix for each layer.

Biases are the parameters that control the **offset**, or the baseline activation level, of a neuron. They shift the input signal along the y-axis before passing it through the activation function. They help prevent all outputs from **zero** when the input is zero. Like the weights, biases are also represented as **matrices**, with one matrix for each layer.

The weights and biases are updated iteratively during training to minimize the loss function. This is typically done using optimization algorithms such as **stochastic gradient descent** or its variants. The process of updating the weights and biases is known as **Backpropagation**, and it is an essential step in training a neural network's feedforward.



Module : 4

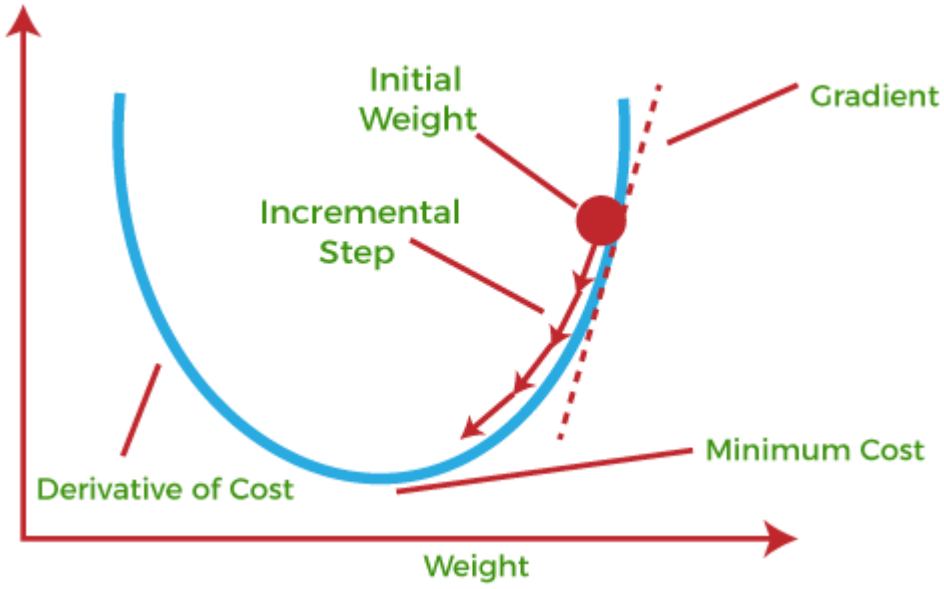
GRADIENT DESCENT ALGORITHM

Most commonly used optimization algorithms to train machine learning models by means of minimizing errors between actual and expected results. Further, gradient descent is also used to train Neural Networks.

Gradient Descent is defined as one of the most commonly used iterative optimization algorithms of machine learning to train the machine learning and deep learning models. It helps in finding the local minimum of a function.

The best way to define the local minimum or local maximum of a function using gradient descent is as follows:

- If we move towards a negative gradient or away from the gradient of the function at the current point, it will give the **local minimum** of that function.
- Whenever we move towards a positive gradient or towards the gradient of the function at the current point, we will get the **local maximum** of that function.



This entire procedure is known as Gradient Ascent, which is also known as steepest descent. ***The main objective of using a gradient descent algorithm is to minimize the cost function using iteration.*** To achieve this goal, it performs two steps iteratively:

- Calculates the first-order derivative of the function to compute the gradient or slope of that function.
- Move away from the direction of the gradient, which means slope increased from the current point by alpha times, where Alpha is defined as Learning Rate.
- It is a tuning parameter in the optimization process which helps to decide the length of the steps.

What is Cost-function?

The cost function is defined as the measurement of difference or error between actual values and expected values at the current position and present in the form of a single real number

It helps to increase and improve machine learning efficiency by providing feedback to this model so that it can minimize error and find the local or global minimum.

Further, it continuously iterates along the direction of the negative gradient until the cost function approaches zero.

At this steepest descent point, the model will stop learning further.

Although cost function and loss function are considered synonymous, also there is a minor difference between them.

The slight difference between the loss function and the cost function is about the error within the training of machine learning models, as loss function refers to the error of one training example, while a cost function calculates the average error across an entire training set.

The cost function is calculated after making a hypothesis with initial parameters and modifying these parameters using gradient descent algorithms over known data to reduce the cost function.

Hypothesis:

Parameters:

Cost function:

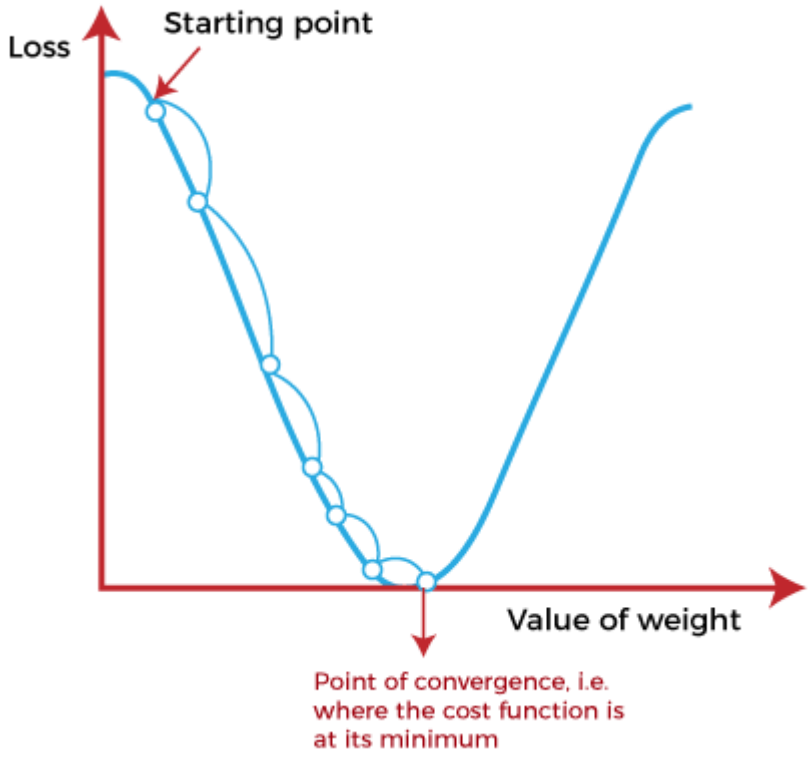
Goal:

How does Gradient Descent work?

The equation for simple linear regression is given as:

$$Y=mX+c$$

Where 'm' represents the slope of the line, and 'c' represents the intercepts on the y-axis.



The starting point(shown in above fig.) is used to evaluate the performance as it is considered just as an arbitrary point.

At this starting point, we will derive the first derivative or slope and then use a tangent line to calculate the steepness of this slope.

Further, this slope will inform the updates to the parameters (weights and bias).

The slope becomes steeper at the starting point or arbitrary point, but whenever new parameters are generated, then steepness gradually reduces, and at the lowest point, it approaches the lowest point, which is called a point of convergence.

The main objective of gradient descent is to minimize the cost function or the error between expected and actual.

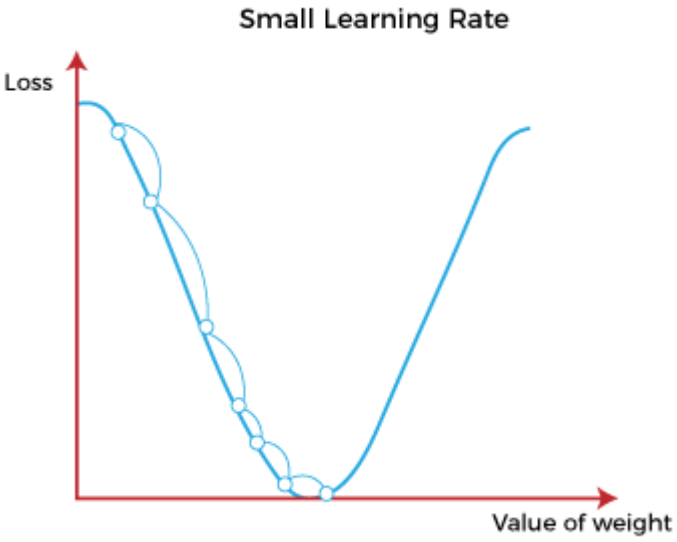
To minimize the cost function, **two data points are required:**

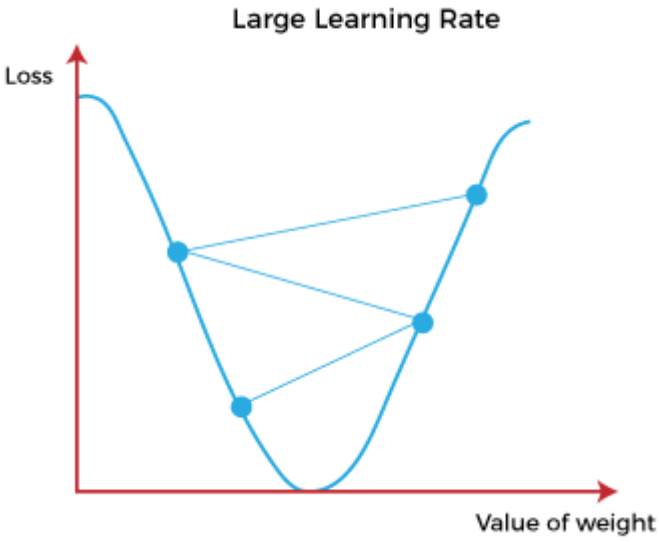
- **Direction & Learning Rate**

These two factors are used to determine the partial derivative calculation of future iteration and allow it to the point of convergence or local minimum or global minimum.

- **Learning Rate:**

It is defined as the step size taken to reach the minimum or lowest point. This is typically a small value that is evaluated and updated based on the behavior of the cost function. If the learning rate is high, it results in larger steps but also leads to risks of overshooting the minimum. At the same time, a low learning rate shows the small step sizes, which compromises overall efficiency but gives the advantage of more precision.





BACK PROPAGATION ALGORITHM

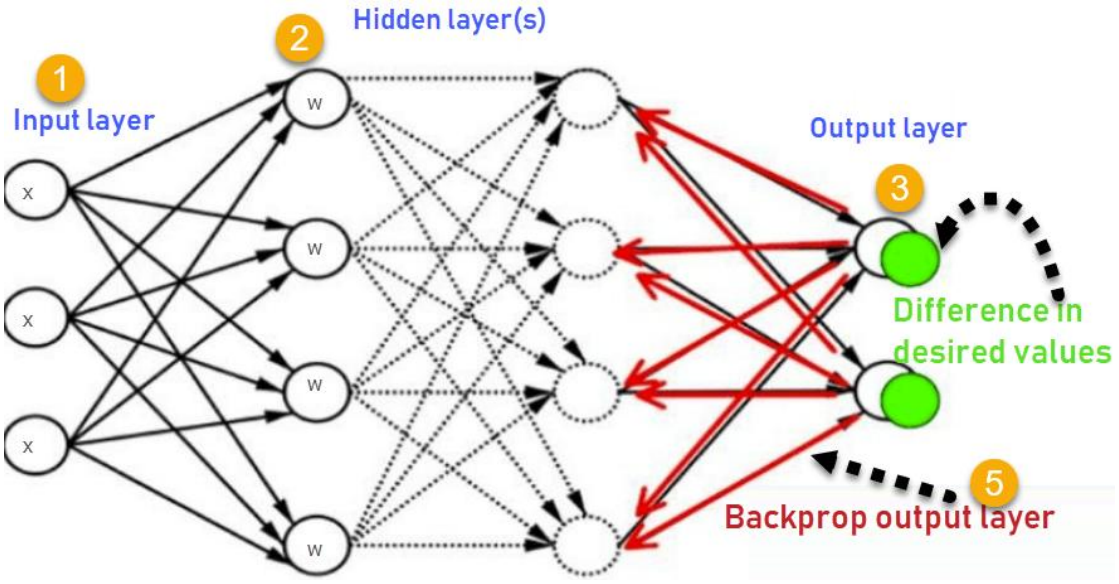
Backpropagation is the essence of neural network training. It is the method of fine-tuning the weights of a neural network based on the error rate obtained in the previous epoch (i.e., iteration). Proper tuning of the weights allows you to reduce error rates and make the model reliable by increasing its generalization.

Backpropagation in neural network is a short form for "backward propagation of errors." It is a standard method of training artificial neural networks. This method helps calculate the gradient of a loss function with respect to all the weights in the network.

How Backpropagation Algorithm Works

The Back propagation algorithm in neural network computes the gradient of the loss function for a single weight by the chain rule. It efficiently computes one layer at a time, unlike a native direct computation. It computes the gradient, but it does not define how the gradient is used. It generalizes the computation in the delta rule.

Consider the following Back propagation neural network example diagram to understand:



How Backpropagation Algorithm Works

1. Inputs X, arrive through the preconnected path
2. Input is modeled using real weights W. The weights are usually randomly selected.
3. Calculate the output for every neuron from the input layer, to the hidden layers, to the output layer.
4. Calculate the error in the outputs

$$\text{Error}_B = \text{Actual Output} - \text{Desired Output}$$

5. Travel back from the output layer to the hidden layer to adjust the weights such that the error is decreased.

Keep repeating the process until the desired output is achieved

Why We Need Backpropagation?

Most prominent advantages of Backpropagation are:

- Backpropagation is fast, simple and easy to program
- It has no parameters to tune apart from the numbers of input
- It is a flexible method as it does not require prior knowledge about the network
- It is a standard method that generally works well
- It does not need any special mention of the features of the function to be learned.

BACKPROPAGATION (training_example, η, n_{in}, n_{out}, n_{hidden})

- Each training example is a pair of the form (x, t), where (x) is the vector of network input values, and (t) is the vector of target network output values.
- η is the learning rate (e.g., 0.05).
- n_i, is the number of network inputs,
- n_{hidden} the number of units in the hidden layer, and
- n_{out} the number of output units.
- The input from unit i into unit j is denoted x_{ji}, and the weight from unit i to unit j is denoted w_{ji}

- Create a feed-forward network with n_i inputs, n_{hidden} hidden units, and n_{out} output units.
- Initialize all network weights to small random numbers
- Until the termination condition is met, Do
  - For each (x, t), in training examples, Do
    - Propagate the input forward through the network:
      1. Input the instance x, to the network and compute the output o_u of every unit u in the network.
    - Propagate the errors backward through the network
      2. For each network unit k, calculate its error term δ_k

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$

      3. For each network unit h, calculate its error term δ_h

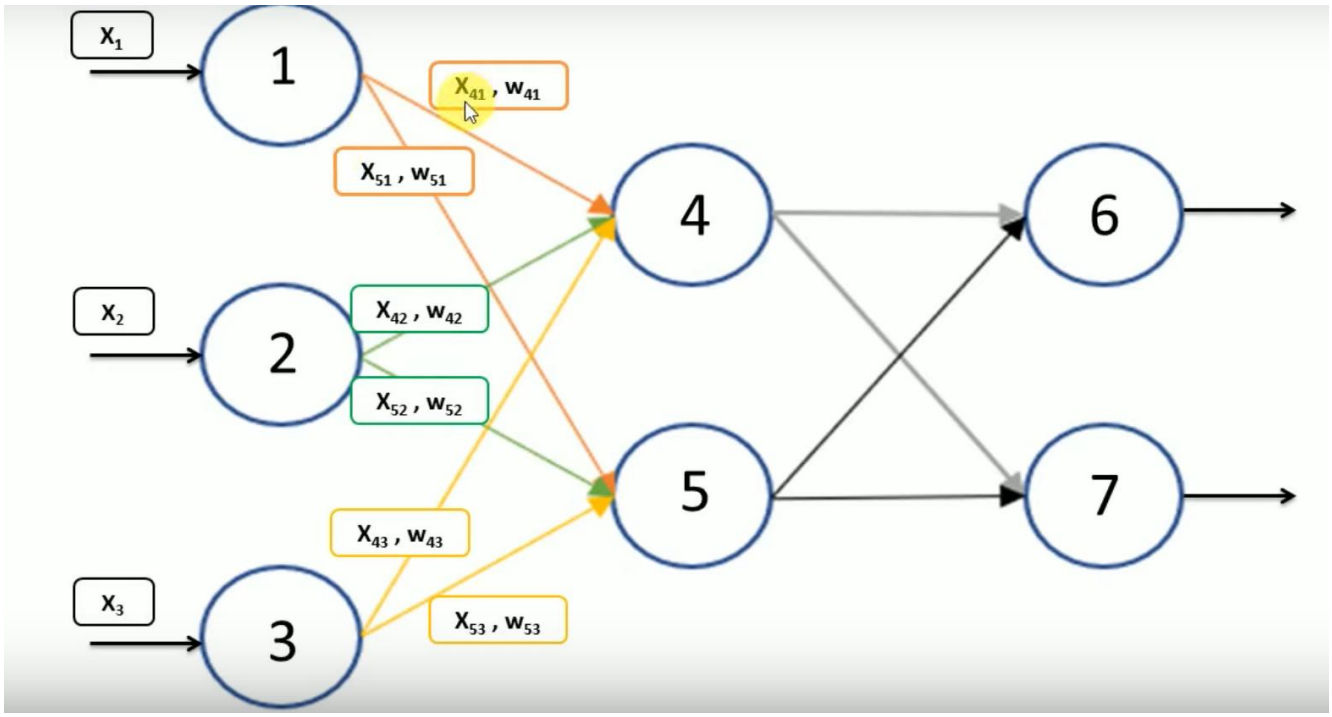
$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{h,k}\delta_k$$

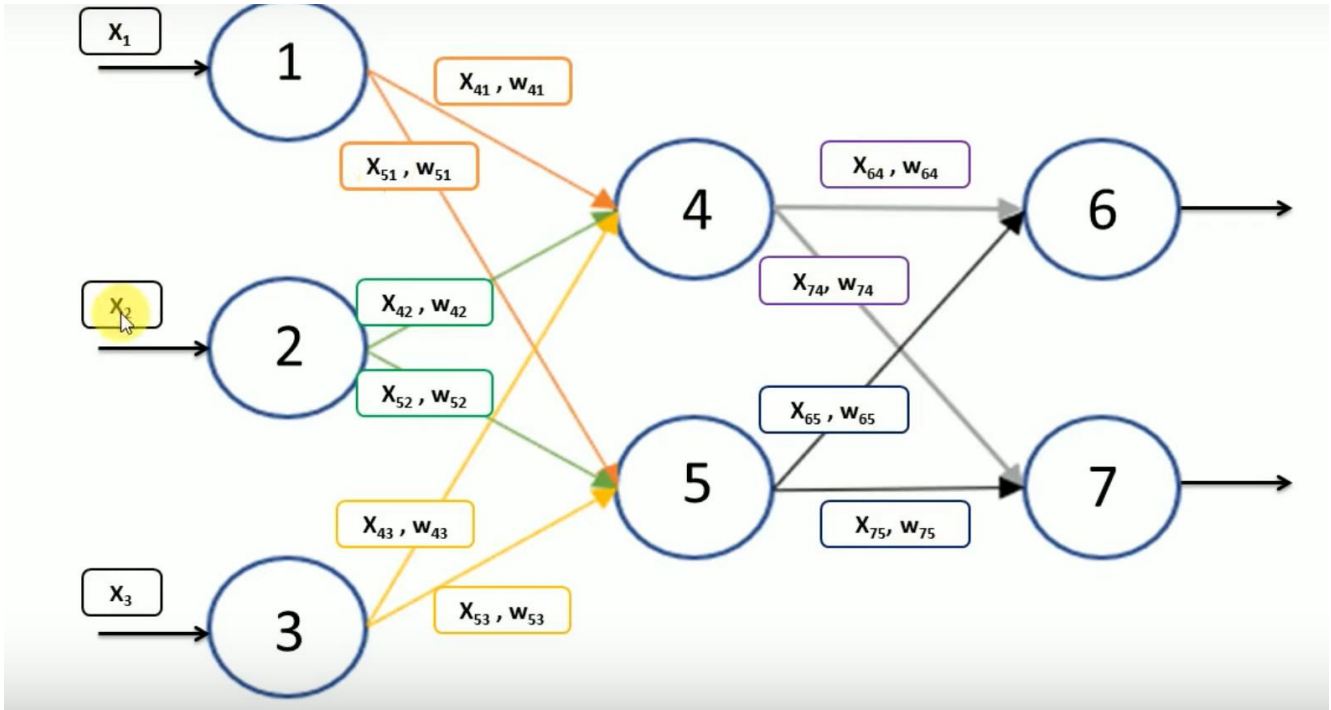
4. Update each network weight w_{ji}

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

Where

$$\Delta w_{ji} = \eta \delta_j x_{ji}$$


X1
1
X41 , w41
X51 , w51
4
6
X2
2
X42 , w42
X52 , w52
5
7
X43 , w43
X3
3
X53 , w53


X1
1
X41 , w41
X51 , w51
4
X64 , w64
6
X74, w74
X2
2
X42 , w42
X52 , w52
X65 , w65
5
7
X43 , w43
X75, w75
X3
3
X53 , w53

- Create a feed-forward network with n_i inputs, n_{hidden} hidden units, and n_{out} output units.
- Initialize all network weights to small random numbers
- Until the termination condition is met, Do
  - For each (x, t), in training examples, Do
    - Propagate the input forward through the network:
      1. Input the instance x, to the network and compute the output o_u of every unit u in the network.
    - Propagate the errors backward through the network
      2. For each network unit k, calculate its error term δ_k

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$

      3. For each network unit h, calculate its error term δ_h

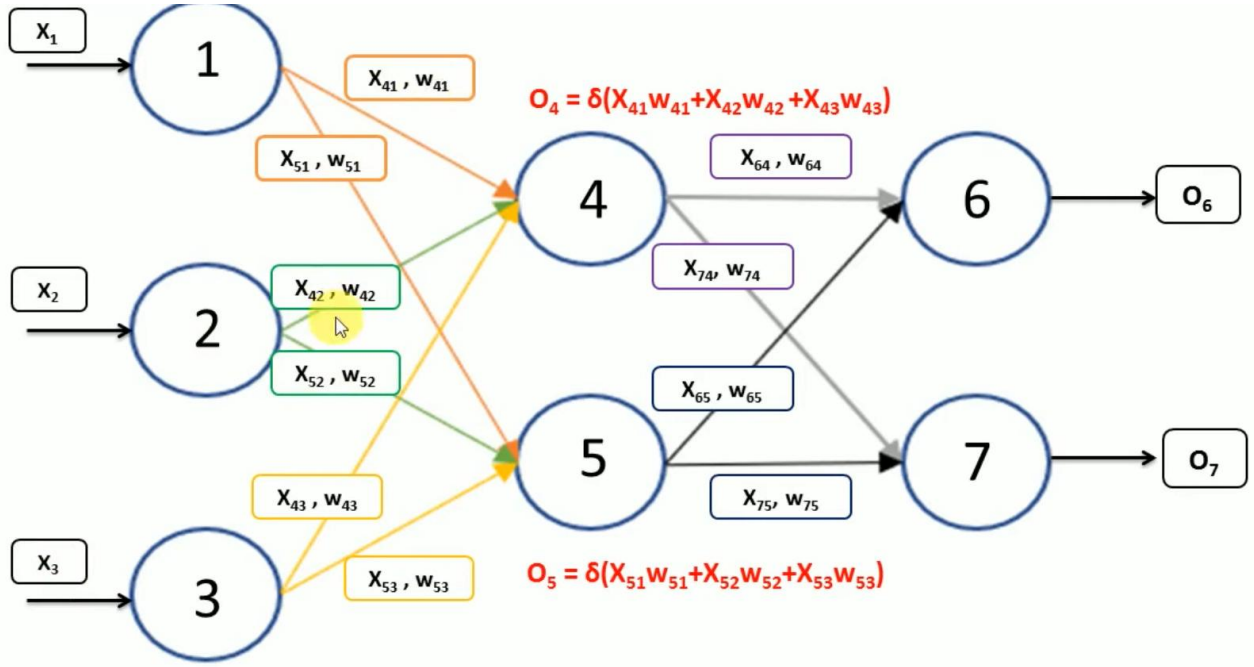
$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{h,k}\delta_k$$

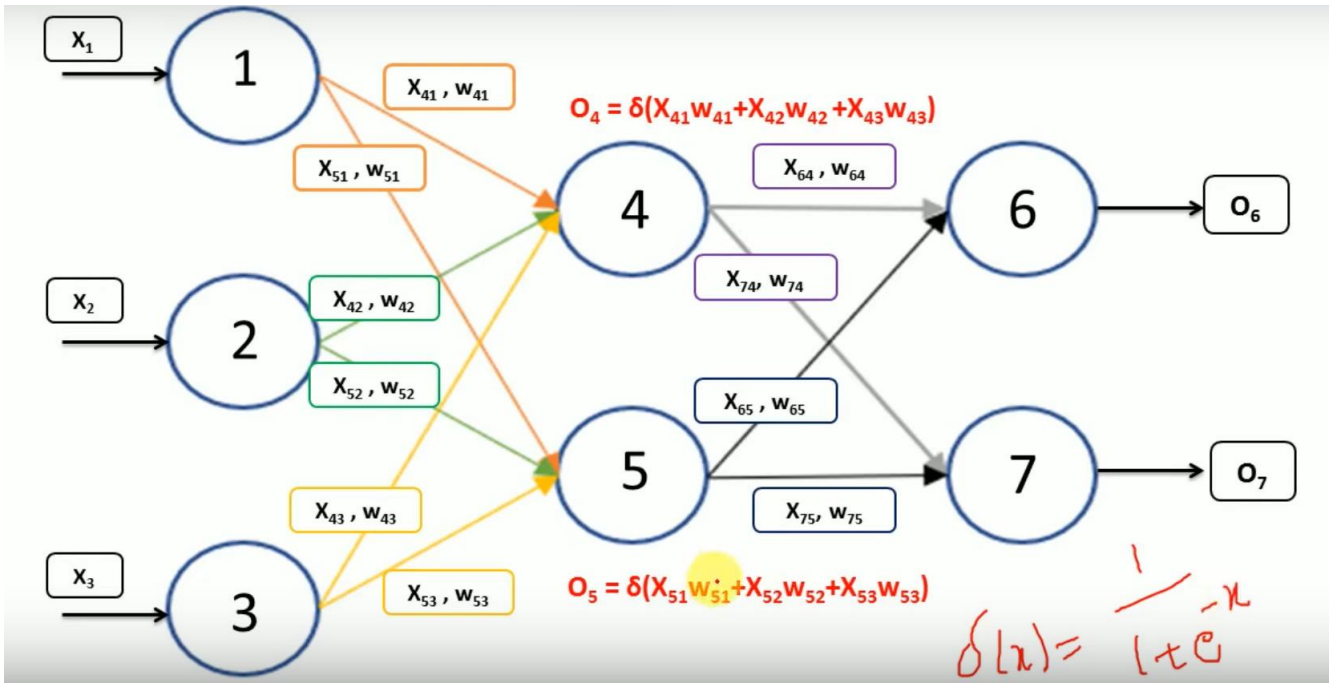
4. Update each network weight w_{ji}

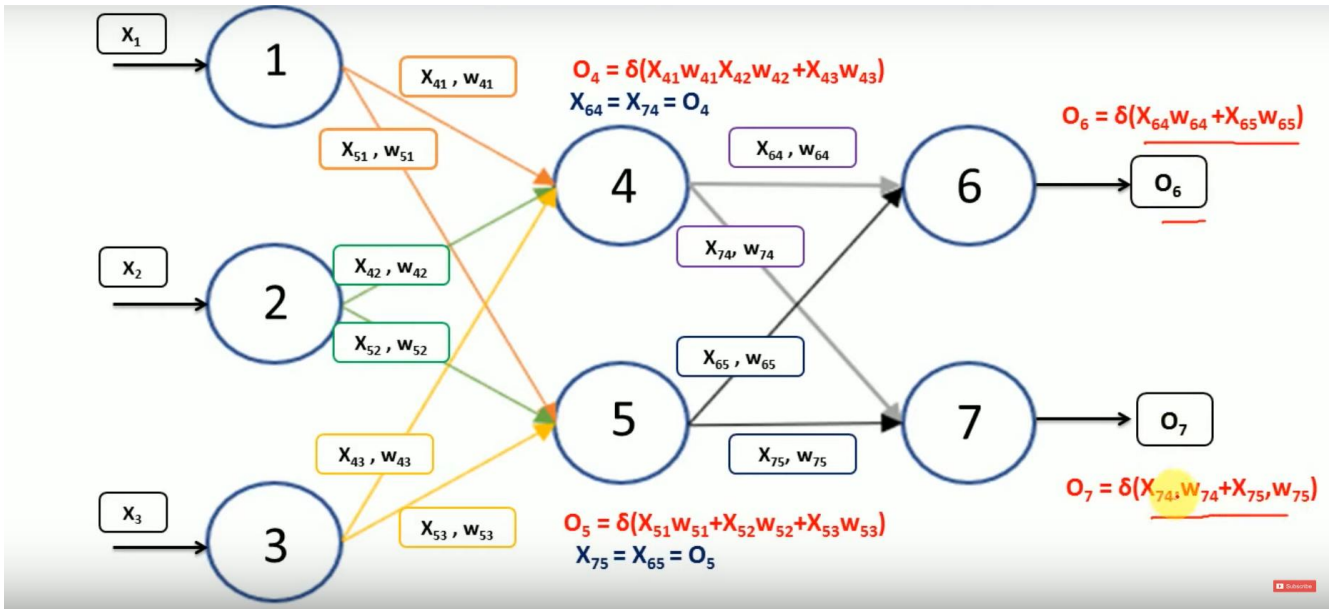
$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

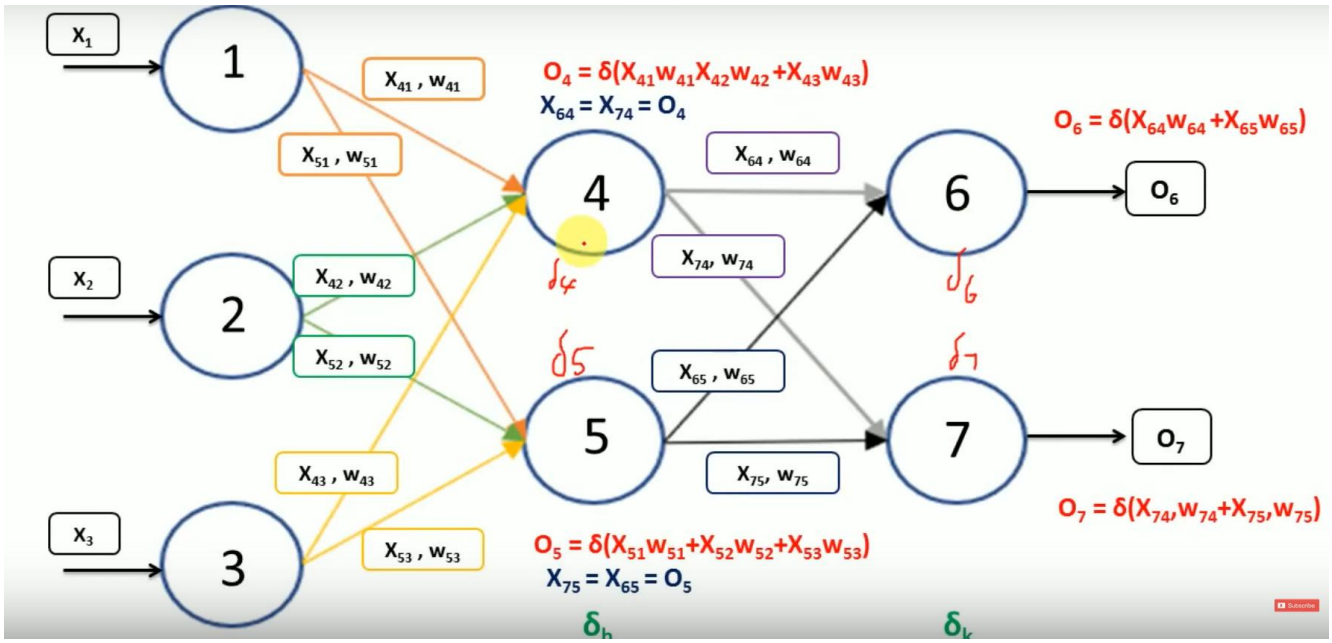
Where

$$\Delta w_{ji} = \eta \delta_j x_{ji}$$




X1
X2
X3
1
2
3
4
5
6
7
O6
O7
X41 , w41
X51 , w51
X42 , w42
X52 , w52
X43 , w43
X53 , w53
X64 , w64
X74, w74
X65 , w65
X75, w75
O4 = δ(X41w41+X42w42 +X43w43)
O5 = δ(X51w51+X52w52+X53w53)
δ(x)= 1 / 1+e^-x


X1
X2
X3
1
2
3
4
5
6
7
O6
O7
X41 , w41
X51 , w51
X42 , w42
X52 , w52
X43 , w43
X53 , w53
X64 , w64
X74, w74
X65 , w65
X75, w75
O4 = δ(X41w41X42w42 +X43w43)
X64 = X74 = O4
O5 = δ(X51w51+X52w52+X53w53)
X75 = X65 = O5
O6 = δ(X64w64 +X65w65)
O7 = δ(X74w74+X75w75)



- Create a feed-forward network with n_i inputs, n_{hidden} hidden units, and n_{out} output units.
- Initialize all network weights to small random numbers
- Until the termination condition is met, Do
  - For each (x, t), in training examples, Do
    - Propagate the input forward through the network:
      1. Input the instance x, to the network and compute the output o_u of every unit u in the network.
    - Propagate the errors backward through the network
      2. For each network unit k, calculate its error term δ_k

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$

      3. For each network unit h, calculate its error term δ_h

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{h,k}\delta_k$$

4. Update each network weight w_{ji}

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

Where

$$\Delta w_{ji} = \eta \delta_j x_{ji}$$