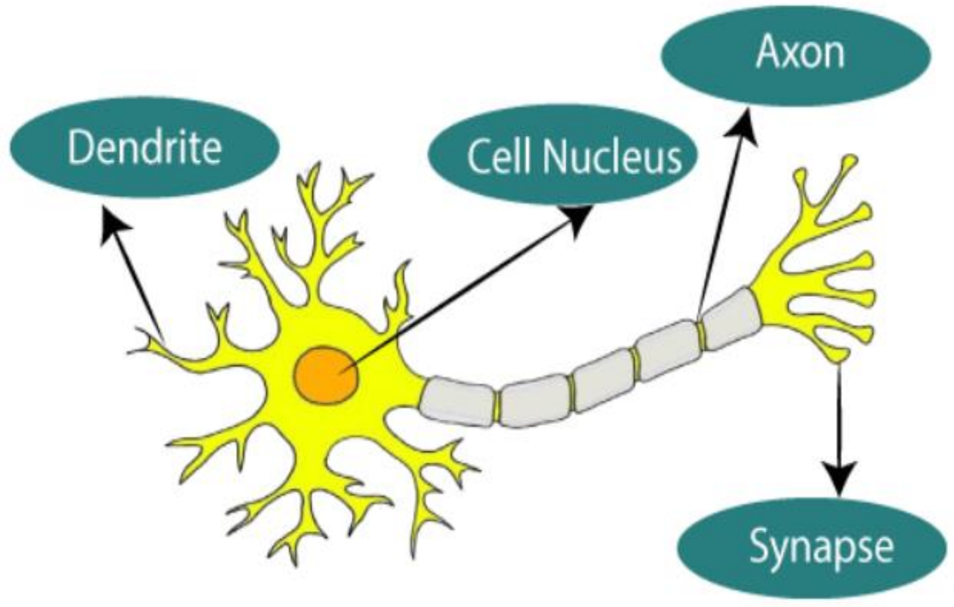
Artificial Neural Network

- The term "Artificial Neural Network" is derived from Biological neural networks that develop the structure of a human brain.

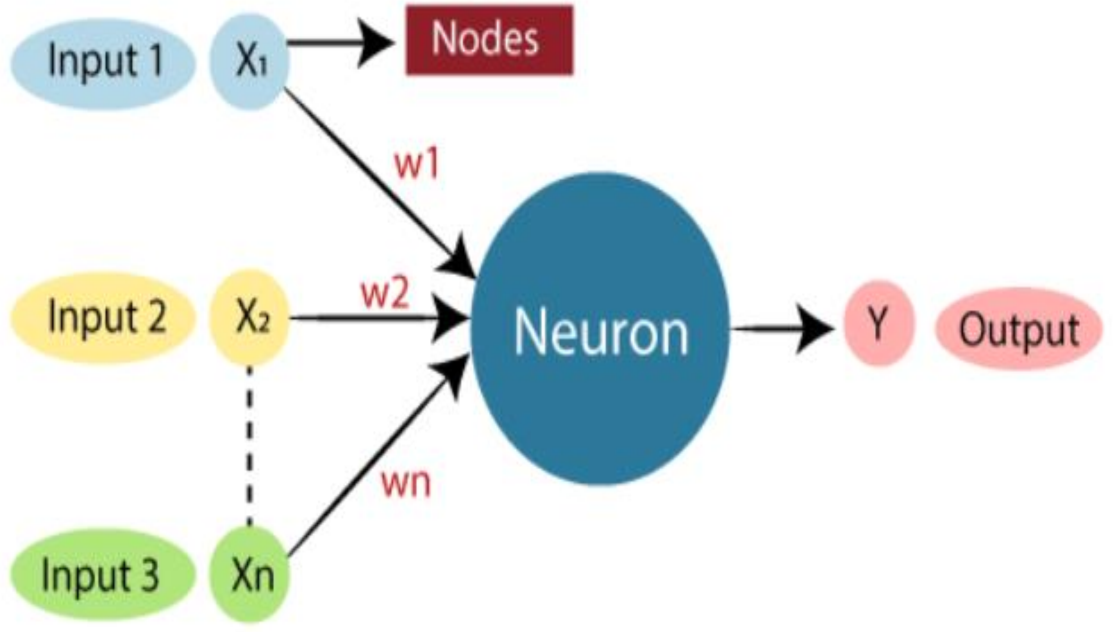
- Similar to the human brain that has neurons interconnected to one another, artificial neural networks also have neurons that are interconnected to one another in various layers of the networks.

- These neurons are known as nodes.

Typical diagram of Biological Neural Network.



Typical Artificial Neural Network



Relationship between Biological neural network and artificial neural network:

Biological Neural Network	**Artificial Neural Network**
Dendrites	Inputs
Cell nucleus	Nodes
Synapse	Weights
Axon	Output

- An Artificial Neural Network in the field of Artificial intelligence where it attempts to mimic the network of neurons makes up a human brain so that computers will have an option to understand things and make decisions in a human-like manner.

- The artificial neural network is designed by programming computers to behave simply like interconnected brain cells.

- 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.

Basic Terminology in Neural Networks

1. Neuron — This is a basic building block of a NN. It takes weighted values, performs mathematical calculation and produce output. It is also called a unit, node or perceptron.

2.Input — This is the data/values passed to the neurons.

3. **Deep Neural Network (DNN)** — This is an ANN with many hidden layers (layers between the input (first) layer and the output (last) layer)

4. Weights — These values explain the strength (degree of importance) of the connection between any two neurons.

5. **Bias** — is a constant value added to the sum of the product between input values and respective weights. It is used to accelerate or delay the activation of a given node.

6. Activation function — is a function used to introduce the non-linearity phenomenon into the NN system. This property will allow the network to learn more complex patterns.

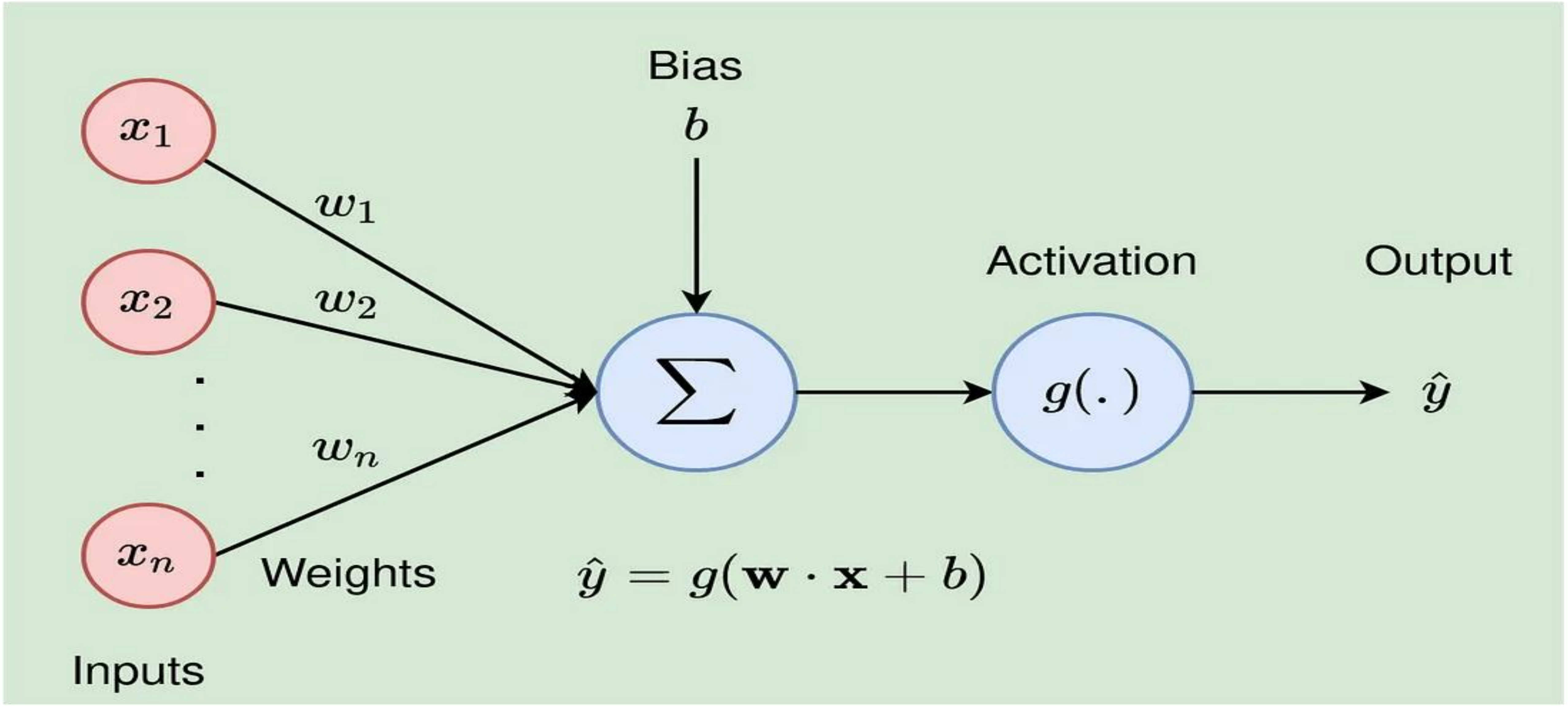
Note: weights and biases are the trainable parameters in NN, that is, the network learns patterns by tuning these parameters to get the best predictions.

Artificial Neuron — Mathematical Operation on one Neuron

An artificial neuron takes input (one or many) values with weights assigned to them.

Inside the node, the weighted inputs are summed up, and an activation function is applied to get the results.

The output of the node is passed on to the other nodes or, in the case of the last layer of the network, the output is the overall output of the network.



An artificial neuron with n input values

Four things are happening —

i. Input is multiplied with the respective weights and added,

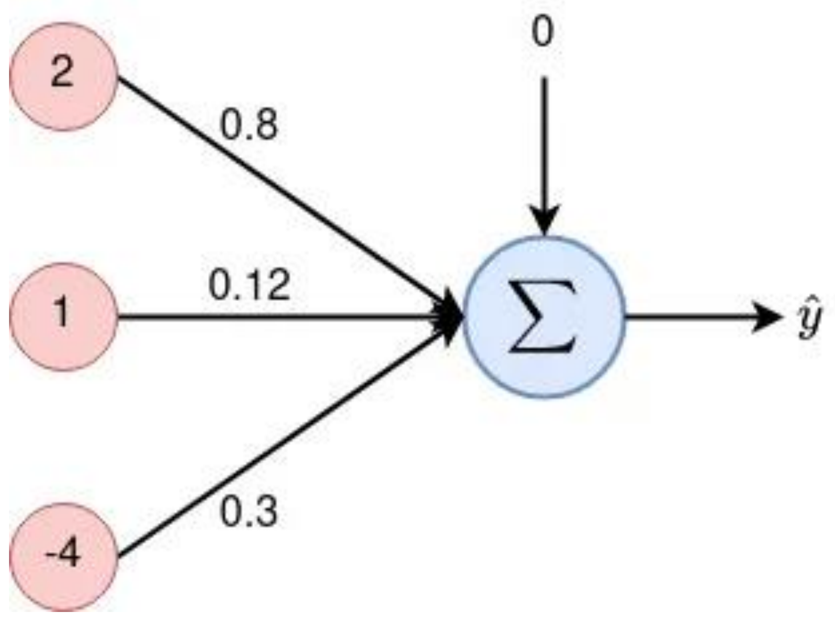
ii. Bias is added to the result,

iii. An activation function (g) is applied to the result,

iv. Output of the neuron is $g(w^T \cdot x+b)$.

A Simplified Example

Let’s take a simple example of how a single neuron work. In this example, we assume 3 input values and a bias of 0.



Step i)- the input values are weighted by multiplying the input values with corresponding weights.

$$x_1 \rightarrow x_1 w_1 = 2 \times 0.8 = 1.6$$
$$x_2 \rightarrow x_2 w_2 = 1 \times 0.12 = 0.12$$
$$x_3 \rightarrow x_3 w_3 = -4 \times 0.3 = -1.2$$

ii) Next, is to sum the weighted input then add the bias

$$x \mapsto \left(\sum_{i=1}^{3} x_i w_i\right) + b = (.6 + 0.12 - 1.2) + 0 = 0.52$$

iii)sigmoid activation function is applied on the above result

$$g(x) = \frac{1}{1 + e^{-x}} = \frac{1}{1 + e^{-0.52}} = 0.627$$

Finally, the output of the neuron is 0.627. If the given neuron is in the hidden layer, then this output becomes the input of the next neuron(s).

On the other hand, if this value is the output of the last layer, then it can be interpreted as the final prediction of the model (it can be seen as the probability of a given class

A single neuron, like the one shown, performs the following mathematical operation,

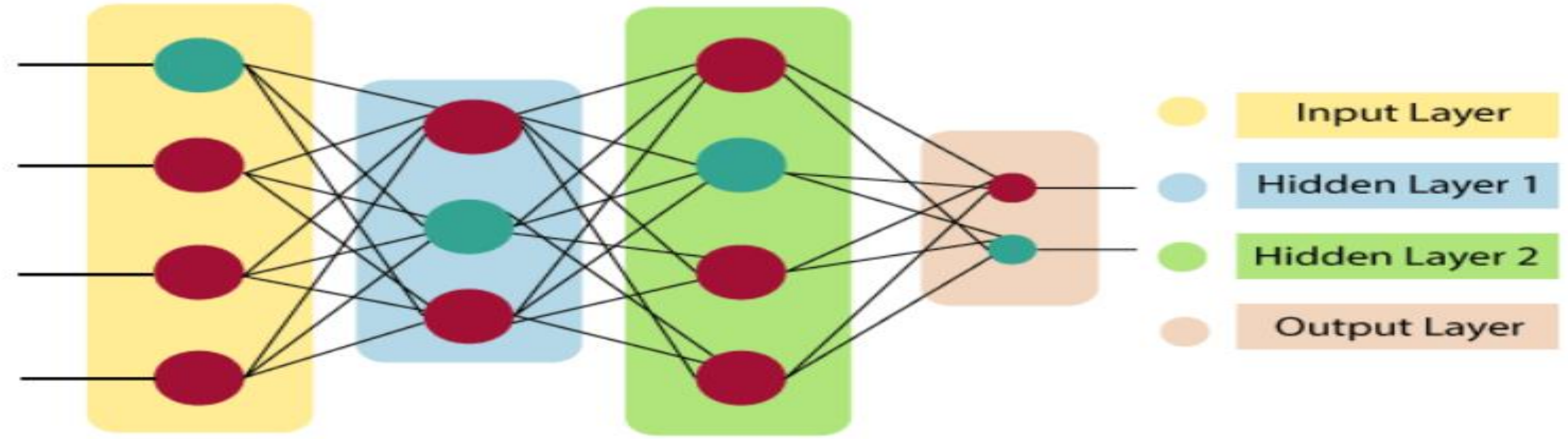
$$x \mapsto \begin{bmatrix} w_1 & w_2 & w_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = w_1x_1 + w_2x_2 + w_2x_3 + b = \mathbf{w}^{\mathsf{T}} \cdot \mathbf{x} + b$$

$$= (2 * 0.8) + (1 * 0.12) + (-4 * 0.3)$$

$$= 0.52$$

The architecture of an artificial neural network:

- To understand the concept of the architecture of an artificial neural network, we have to understand what a neural network consists of.
- In order to define a neural network that consists of a large number of artificial neurons, which are termed units arranged in a sequence of layers.
- Artificial Neural Network primarily consists of three layers:



Input Layer:

As the name suggests, it accepts inputs in several different formats provided by the programmer.

Hidden Layer:

The hidden layer presents in-between input and output layers. It performs all the calculations to find hidden features and patterns.

Output Layer:

The input goes through a series of transformations using the hidden layer, which finally results in output that is conveyed using this layer.

$$\sum_{i=1}^{n} Wi * Xi + b$$

The artificial neural network takes input and computes the weighted sum of the inputs and includes a bias. This computation is represented in the form of a transfer function.

It determines weighted total is passed as an input to an activation function to produce the output.

There are distinctive activation functions available that can be applied upon the sort of task we are performing.

How do artificial neural networks work?

- Artificial Neural Network can be best represented as a weighted directed graph, where the artificial neurons form the nodes.

- The association between the neurons outputs and neuron inputs can be viewed as the directed edges with weights.

- The Artificial Neural Network receives the input signal from the external source in the form of a pattern and image in the form of a vector.

- These inputs are then mathematically assigned by the notations x(n) for every n number of inputs.

- Afterward, each of the input is multiplied by its corresponding weights (these weights are the details utilized by the artificial neural networks to solve a specific problem).

- In general terms, these weights normally represent the strength of the interconnection between neurons inside the artificial neural network. All the weighted inputs are summarized inside the computing unit.

- If the weighted sum is equal to zero, then bias is added to make the output non-zero or something else to scale up to the system's response.

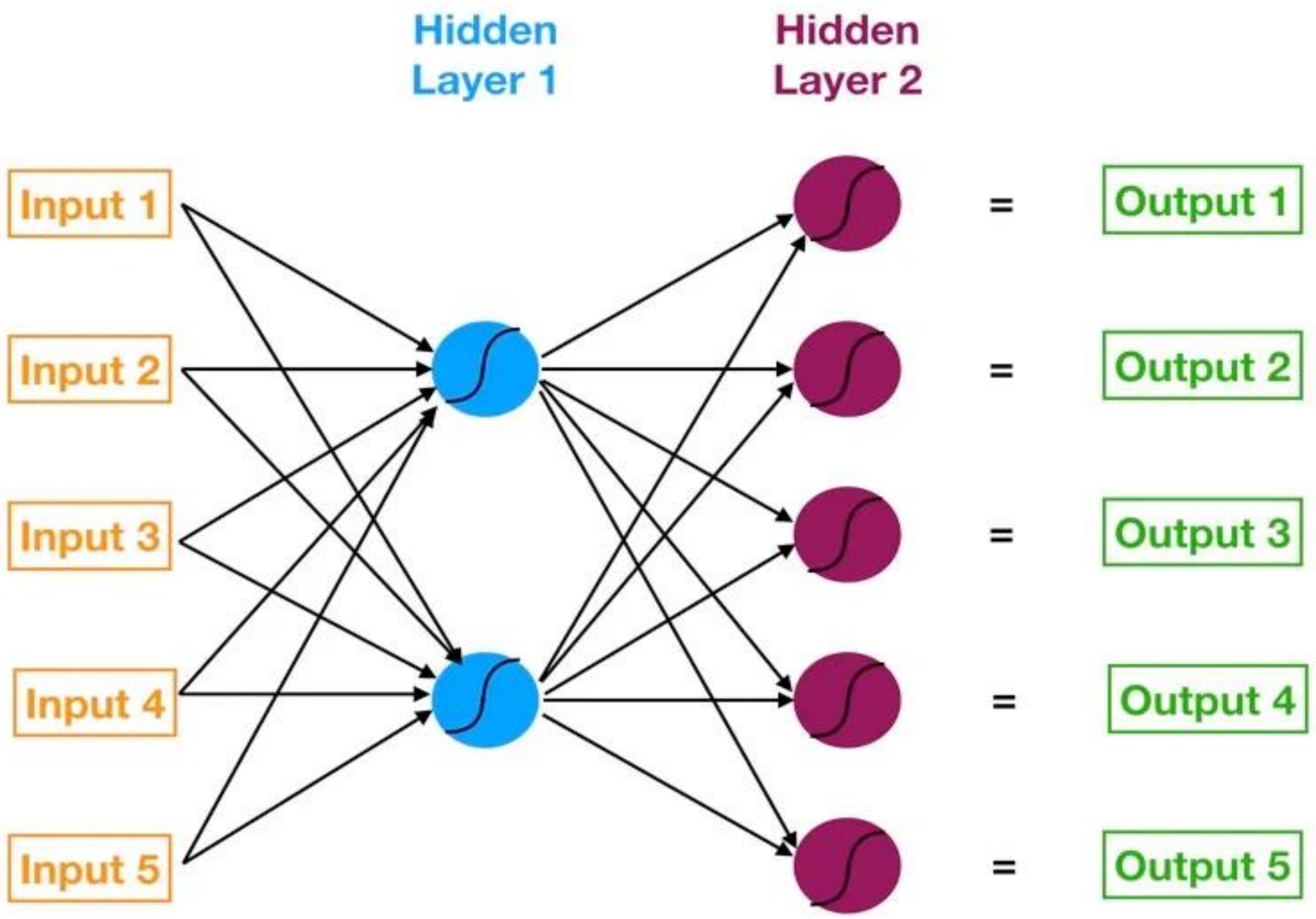
- Bias has the same input, and weight equals to 1. Here the total of weighted inputs can be in the range of 0 to positive infinity.
- Here, to keep the response in the limits of the desired value, a certain maximum value is benchmarked, and the total of weighted inputs is passed through the activation function.
- The activation function refers to the set of transfer functions used to achieve the desired output.

There is a different kind of the activation function, but primarily either linear or non-linear sets of functions. Some of the commonly used sets of activation functions are the Binary, linear, Tangent hyperbolic, sigmoidal, etc. activation functions.

Neural networks are multi-layer networks of neurons that we use to classify things, make predictions.

The learning process of a neural network is performed with the layers.

Hidden layers reside in-between input and output layers and this is the primary reason why they are referred to as *hidden*.



Neural network with two hidden layers.

1. The input layer of our model in orange.

2. Our first hidden layer of neurons in blue.

3. Our second hidden layer of neurons in magenta.

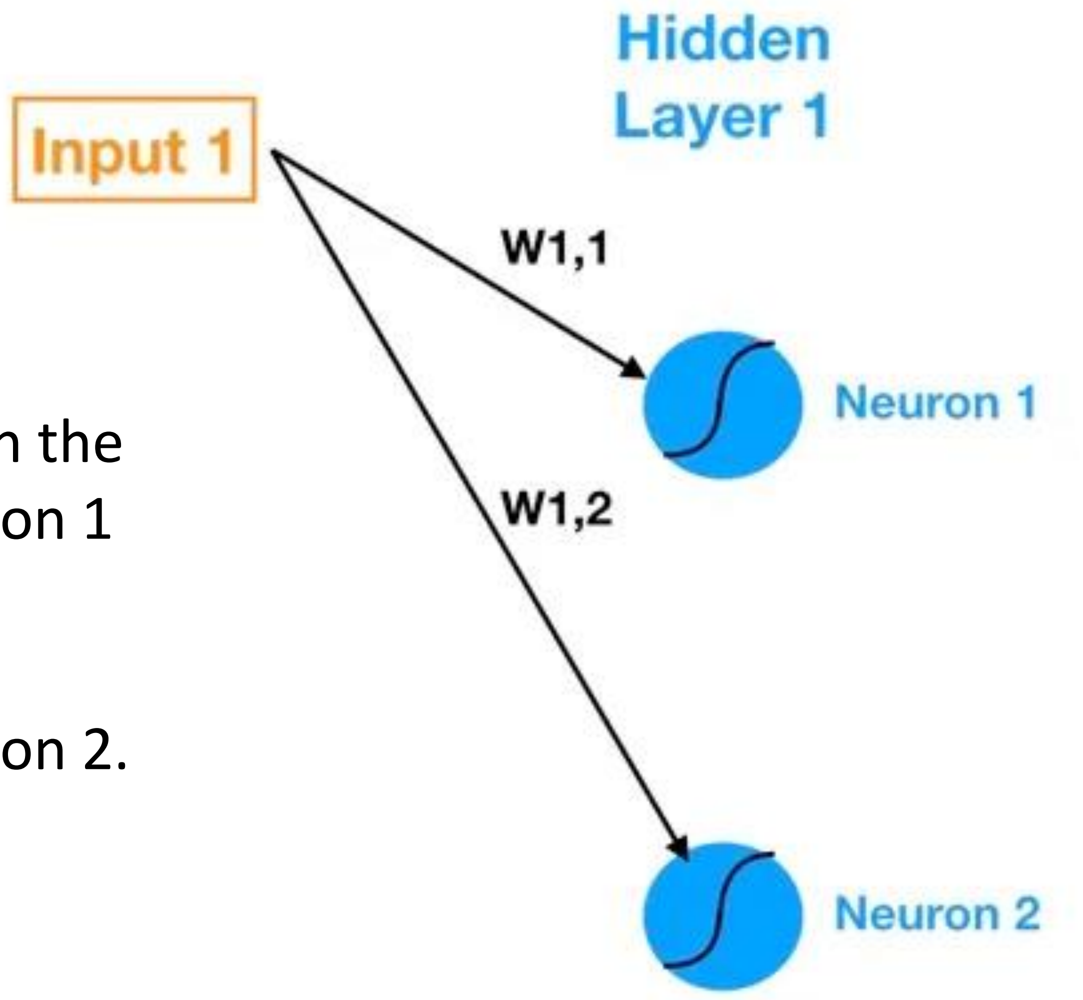
4. The output layer (a.k.a. the prediction) of our model in green.

The first hidden layer consists of two neurons. So to connect all five inputs to the neurons in Hidden Layer 1, we need ten connections.

The connections between Input 1 and Hidden Layer 1

— W1,1 denotes the weight that lies in the connection between Input 1 and Neuron 1

and W1,2 denotes the weight in the connection between Input 1 and Neuron 2.

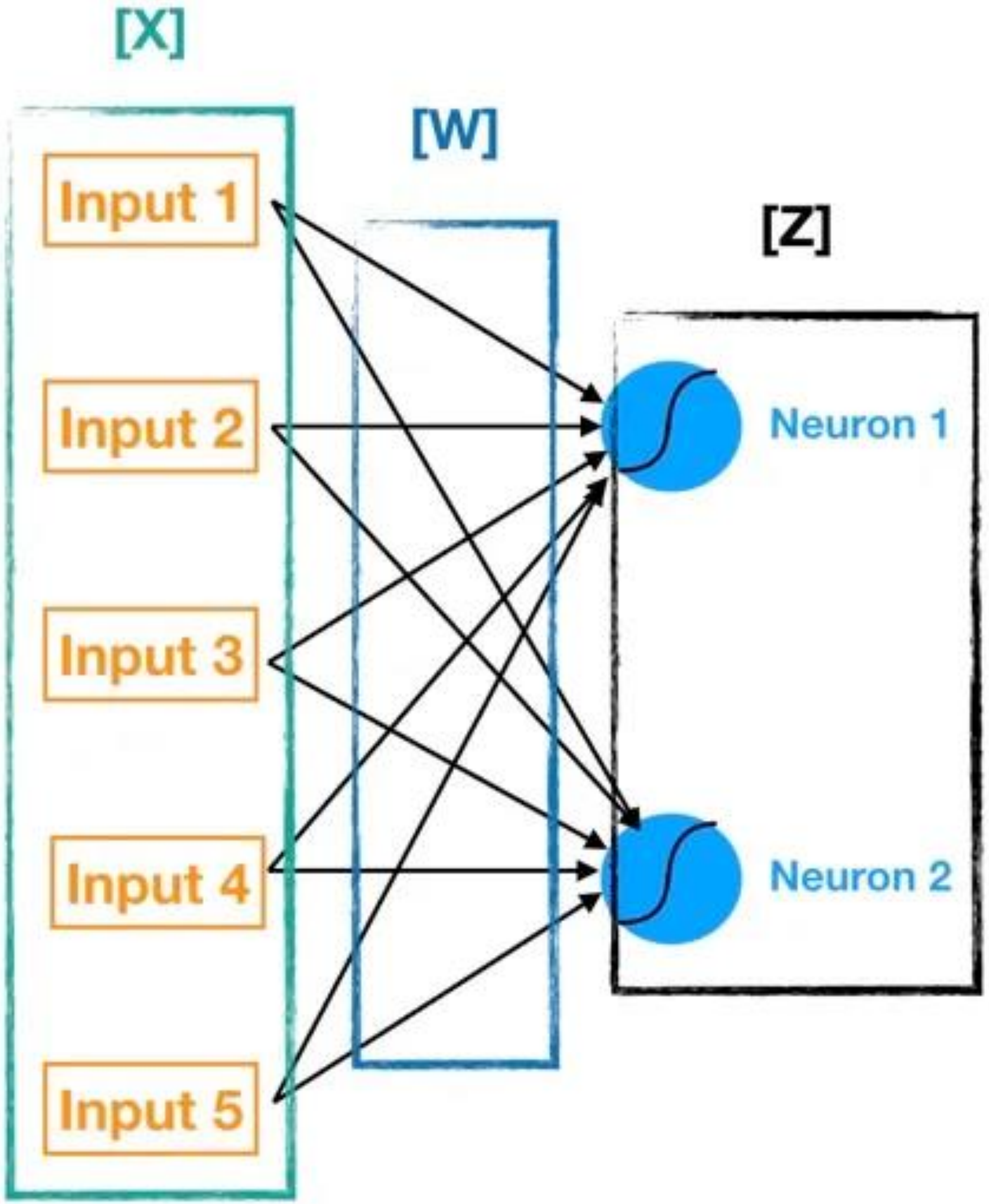


Now let’s calculate the outputs of each neuron in Hidden Layer 1.

We do so using the following formulas (**W** denotes weight, **In** denotes input).

Z1 = W1*In1 + W2*In2 + W3*In3 + W4*In4 + W5*In5 + Bias_Neuron1

Neuron 1 Activation = Sigmoid(Z1)


[X]
[W]
[Z]
Input 1
Input 2
Input 3
Input 4
Input 5
Neuron 1
Neuron 2

We can use matrix math to summarize this calculation.
Here, W4,2 denotes the weight that lives in the connection between Input 4 and Neuron 2

$$\begin{bmatrix} W1,1 & W2,1 & W3,1 & W4,1 & W5,1 \\ W1,2 & W2,2 & W3,2 & W4,2 & W5,2 \end{bmatrix} \times \begin{bmatrix} X1 \\ X2 \\ X3 \\ X4 \\ X5 \end{bmatrix} + \begin{bmatrix} Bias1 \\ Bias2 \end{bmatrix} = \begin{bmatrix} Z1 \\ Z2 \end{bmatrix}$$

Training Neural Network

The training process of a neural network is like that of many other data science models —

- **Define a cost function and**
- **Use <u>gradient descent optimization</u> to minimize it.**

Training a neural network refers to the process of optimizing the parameters (weights and biases) of the network using a labeled training dataset.

The goal is to adjust these parameters so that the network can accurately predict the correct output for a given input.

The training process involves several steps:

1. Initialization: The parameters of the neural network, such as weights and biases, are typically initialized randomly or with specific initialization techniques.

2. Forward Propagation: During the forward propagation step, the input data is passed through the network layer by layer. Each layer applies a set of linear and non-linear operations to produce an output.

3. Loss Calculation: Once the forward propagation is completed, the output of the neural network is compared to the actual expected output (ground truth) from the labeled training data. The loss function, also known as the objective function or the cost function, quantifies the discrepancy between the predicted output and the ground truth.

4. Backward Propagation (Backpropagation): Backpropagation is used to compute the gradients of the loss function with respect to the parameters of the network. It involves propagating the error gradients backward through the network, using the chain rule of calculus.

5. Parameter Updates: After obtaining the gradients, the parameters of the network are updated using an optimization algorithm, such as gradient descent or one of its variants.

The update rule adjusts the parameters in the direction that minimizes the loss function. The learning rate determines the step size of these updates.

6. Iterative Process: Steps 2 to 5 are repeated for multiple iterations or epochs until a stopping criterion is met. Each iteration or epoch consists of a forward pass, loss calculation, backward pass, and parameter updates.

7. Validation and Testing: During the training process, it is common to evaluate the performance of the network on a separate validation dataset to monitor its generalization ability and detect overfitting.

Once the training is completed, the final model can be tested on an independent testing dataset to assess its performance on unseen data.

Cost Function as Mean Squared Error (MSE)

The most common metric for regression tasks is MSE. It is the average of the squared difference between the predicted and actual value.

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

<u>Gradient descent optimization</u>

Gradient descent is an optimization algorithm used to minimize some function by iteratively moving in the direction of steepest descent as defined by the negative of the gradient. In machine learning, we use gradient descent to update the parameters of our model.

Parameters refer to coefficients in Linear Regression and weights in neural networks

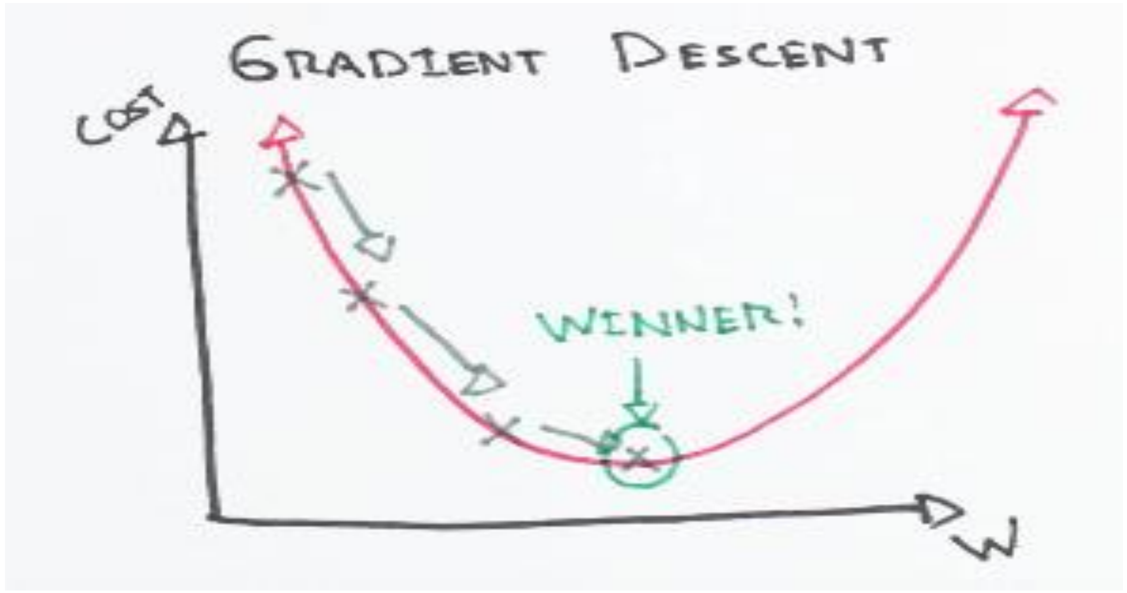
Gradient descent is a general optimization algorithm used to minimize the loss function in various machine learning models, including linear classifiers.

It iteratively adjusts the model parameters by taking steps proportional to the negative gradient of the loss function.

In the context of linear classifiers, gradient descent aims to find the optimal set of weights that minimize the classification error.

Starting at the top of the mountain, we take our first step downhill in the direction specified by the negative gradient.
Next we recalculate the negative gradient (passing in the coordinates of our new point) and take another step in the direction it specifies.

We continue this process iteratively until we get to the bottom of our graph, or to a point where we can no longer move downhill–a local minimum.



Ref:- https://ml-cheatsheet.readthedocs.io/en/latest/gradient_descent.html

The update rule in gradient descent involves computing the gradient of the loss function with respect to the model parameters and updating the parameters in the direction that minimizes the loss.

There are variations of gradient descent, such as batch gradient descent, stochastic gradient descent (SGD), and mini-batch gradient descent, that differ in the amount of data used for each parameter update.

Gradient descent is a general optimization algorithm applicable to both linearly separable and non-linearly separable datasets.

Gradient descent (GD) Vs stochastic gradient descent (SGD)

Both are optimization algorithms used to update the parameters of a model based on the gradient of a loss function. However, they differ in the way they update the parameters and the amount of data they use for each update.

In GD, the entire training dataset is used to compute the gradient of the loss function. The gradient is averaged over all the training examples before updating the parameters. Therefore, GD performs one update per iteration using the complete dataset.

However in SGD, only a single training example (or a small subset) is randomly selected for each parameter update. SGD performs multiple updates per iteration, with each update based on the gradient of the loss function computed on a single or small batch of training examples.

In summary, GD computes gradients over the entire dataset, performs one update per iteration, and is computationally expensive but provides smoother convergence.

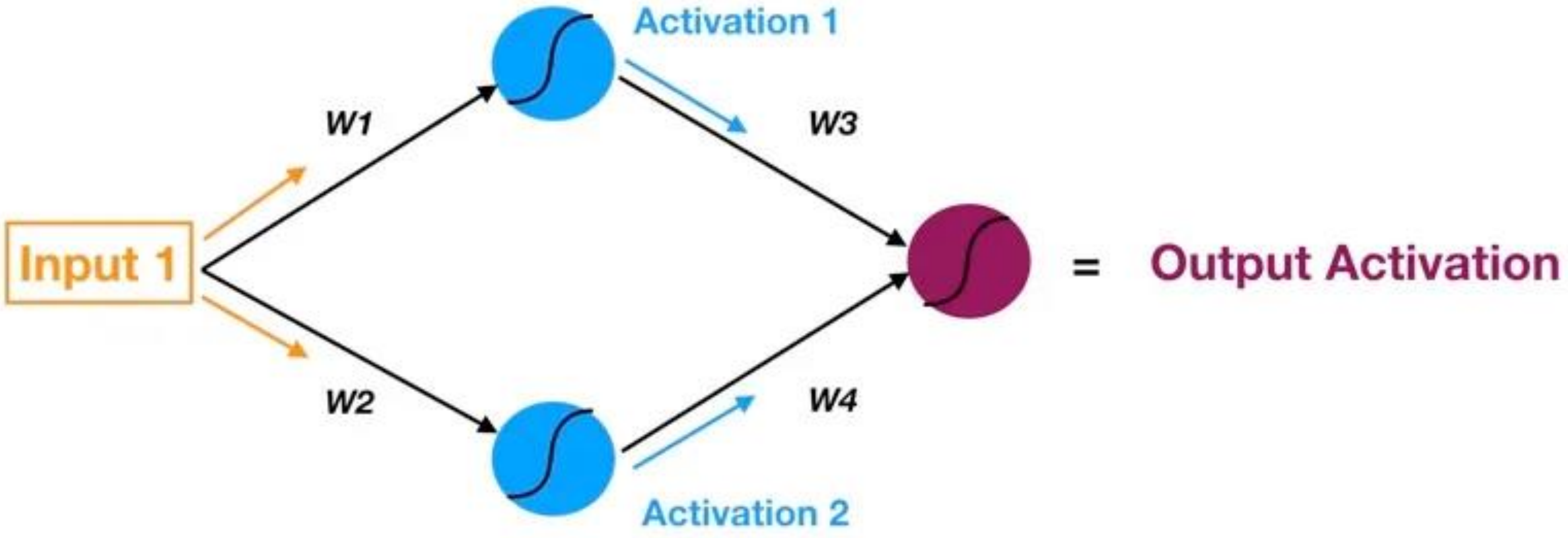
On the other hand, SGD updates the parameters based on a single or small subset of training examples, performs multiple updates per iteration, is faster, but introduces more noise and may exhibit oscillatory behavior.

Forward propagation

It is the process of moving forward through the neural network from inputs to the ultimate output or prediction

The objective of forward propagation is to calculate the activations at each neuron for each successive hidden layer until we arrive at the output.
At the output activation, we make our prediction, and compute the error in our model.

Forward Propagation



Backward Propagation:

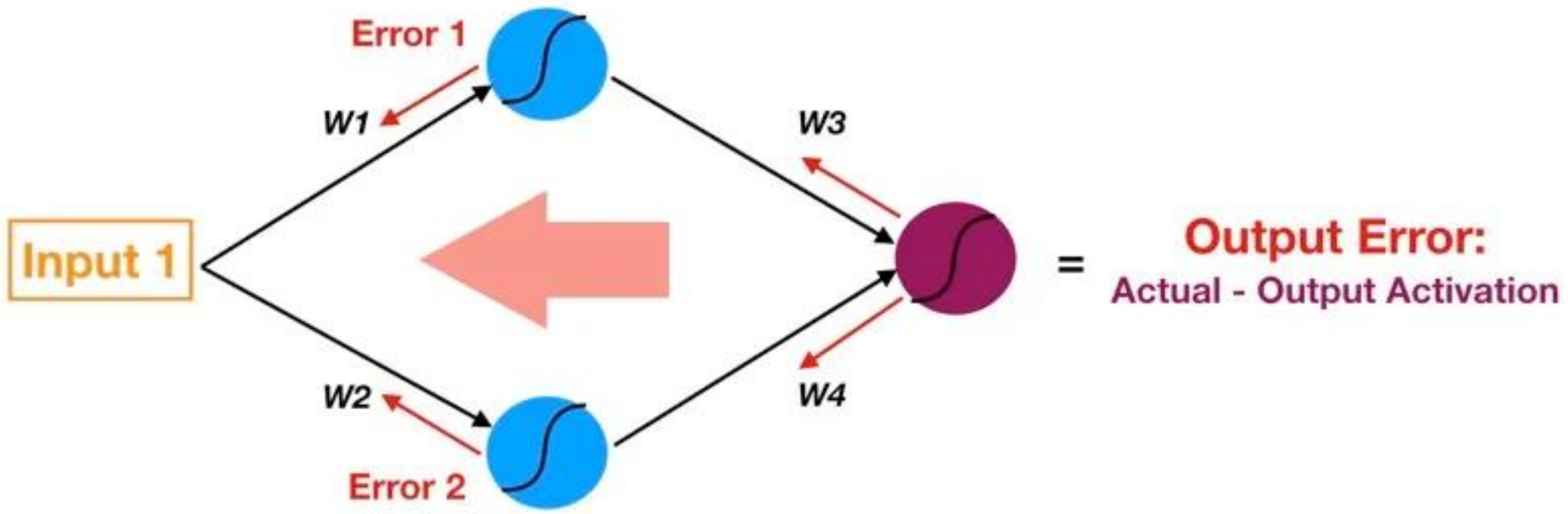
We then move the error backwards through our model via the same weights and connections that we use for forward propagating our signal

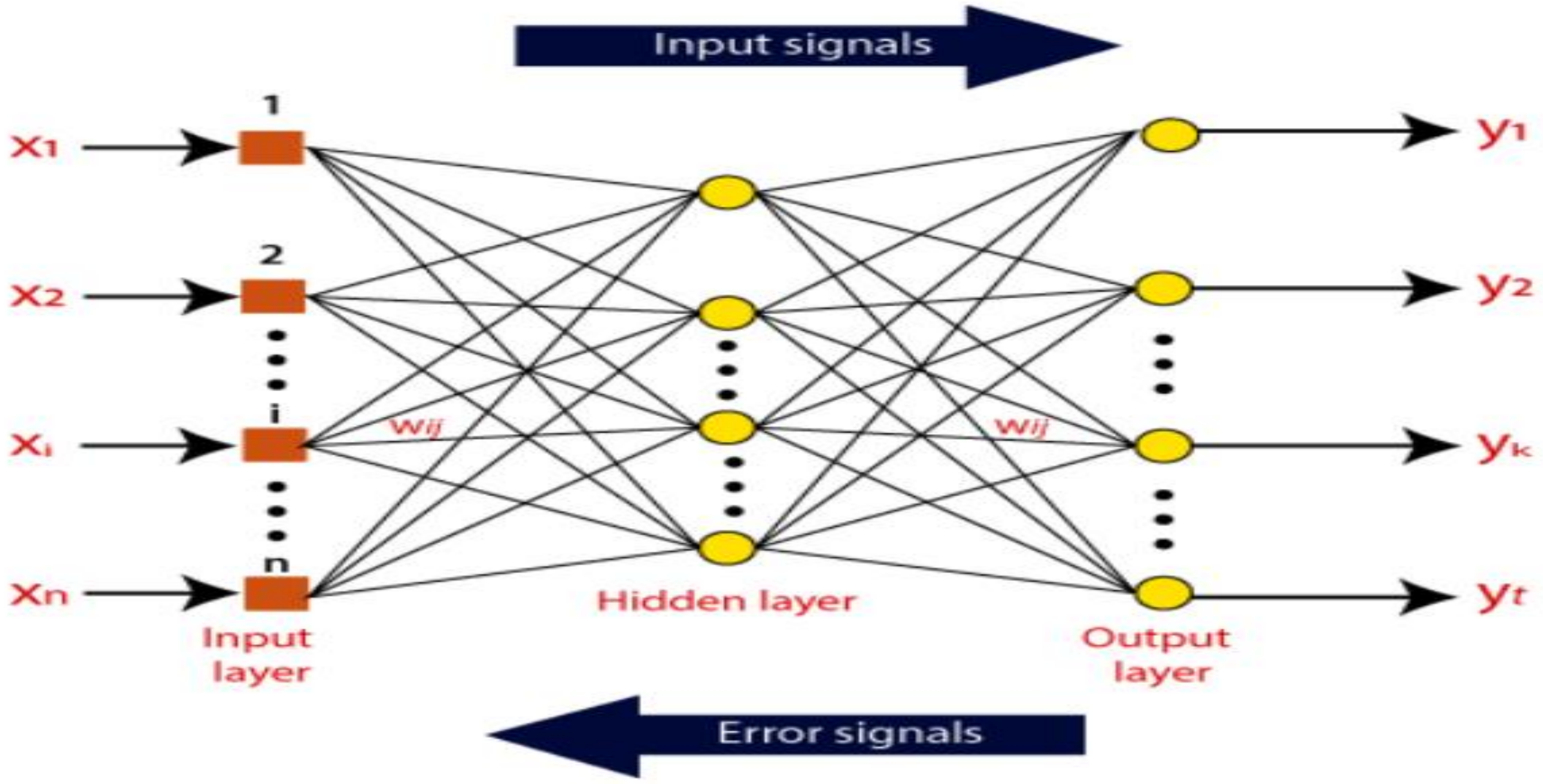
In backward propagation, we calculate the error attributable to each neuron starting from the layer closest to the output all the way back to the starting layer of our model.

It is the process of shifting the error backwards layer by layer and attributing the correct amount of error to each neuron in the neural network.

Hence, backpropagation allows us to calculate the error attributable to each neuron and that in turn allows us to calculate the partial derivatives and ultimately the gradient so that we can utilize gradient descent.

Backward Propagation




Input signals
1
2
i
n
x_1
x_2
x_i
x_n
w_{ij}
w_{ij}
y_1
y_2
y_k
y_t
Hidden layer
Input layer
Output layer
Error signals

Activation Function

- Activation functions refer to the functions used in neural networks to compute the weighted sum of input and biases, which is used to choose the neuron that can be fire or not.

- It controls the presented information through some gradient processing, normally gradient descent. It produces an output for the neural network that includes the parameters in the data.

- Activation function can either be linear or non-linear, relying on the function it shows. It is used to control the output of outer neural networks across various areas, such as speech recognition, segmentation, fingerprint detection, cancer detection system, etc.

- In the artificial neural network, we can use activation functions over the input to get the precise output. These are some activation functions that are used in ANN.

Linear Activation Function:

- The equation of the linear activation function is the same as the equation of a straight line i.e.

$$\mathbf{Y= mx+ c}$$

- If we have many layers and all the layers are linear in nature, then the final activation function of the last layer is the same as the linear function of the first layer. The range of a linear function is –infinitive to + infinitive.

- Linear activation function can be used at only one place that is the output layer.

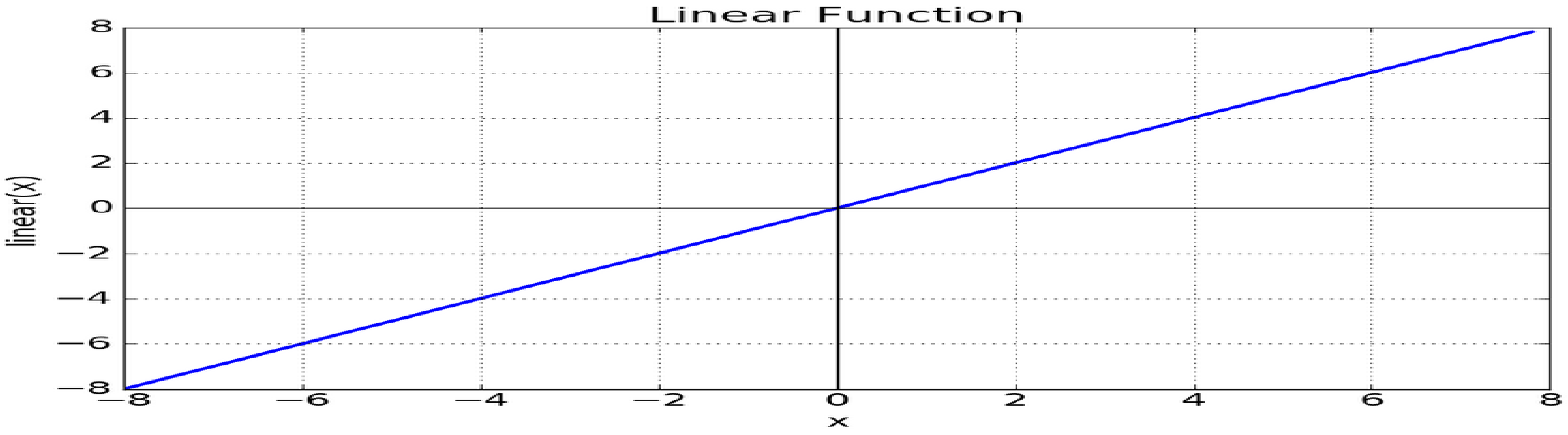
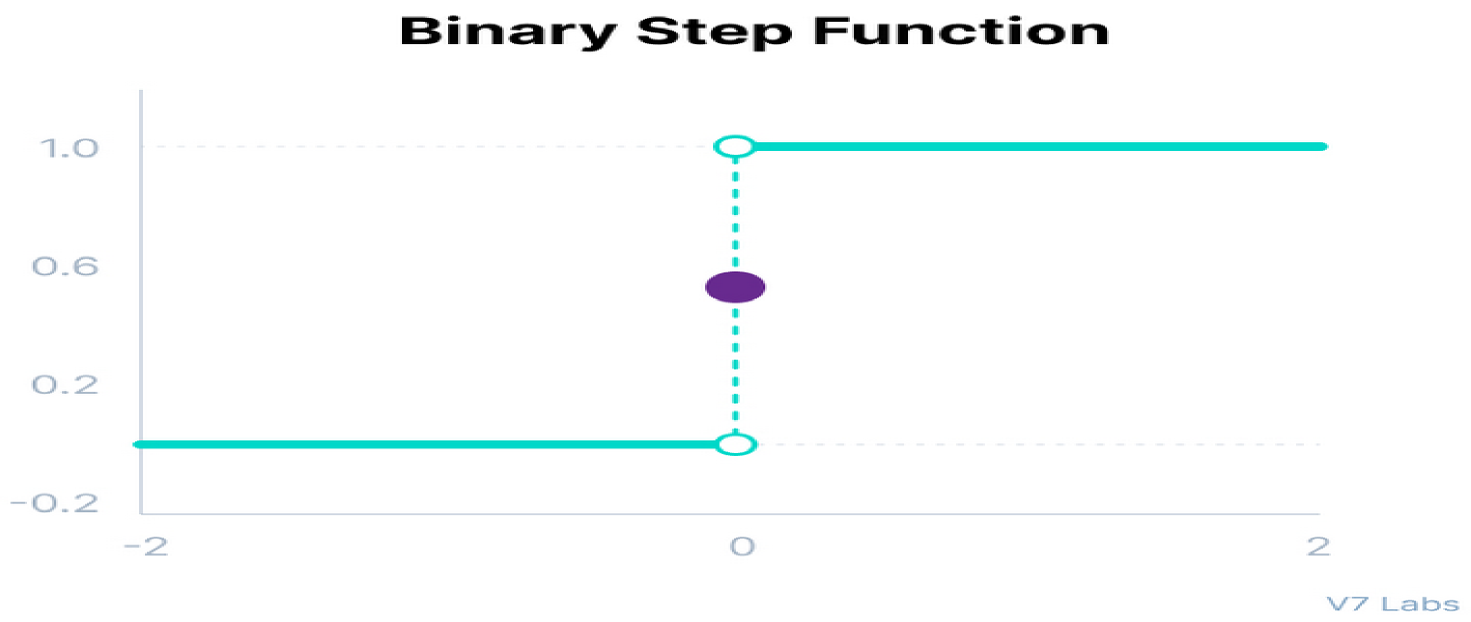


Fig: Linear Activation Function

Binary Step Function

Binary step function depends on a threshold value that decides whether a neuron should be activated or not.

The input fed to the activation function is compared to a certain threshold; if the input is greater than it, then the neuron is activated, else it is deactivated, meaning that its output is not passed on to the next hidden layer.



Binary Step Function

Mathematically it can be represented as:

Binary step

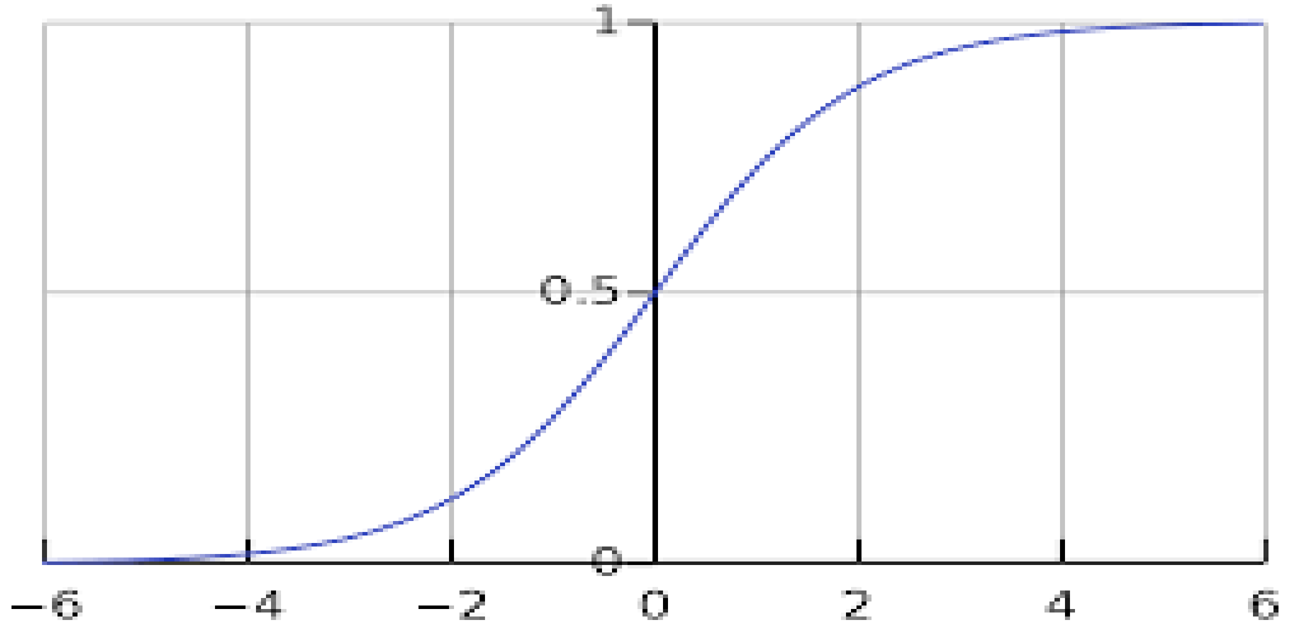
$$f(x) = \begin{cases} 0 & for\ x < 0 \\ 1 & for\ x \geqslant 0 \end{cases}$$

Sigmoid function:

- Sigmoid function refers to a function that is projected as S - shaped graph.

$$\mathbf{A = 1/(1+e^{-x})}$$

- This function is non-linear. The value of X is directly proportional to the values of Y. It means a slight change in the value of x would also bring about changes in the value of y.



Tanh Function:

- The activation function, which is more efficient than the sigmoid function is Tanh function.

- Tanh function is also known as Tangent Hyperbolic Function.

- It is a mathematical updated version of the sigmoid function.

- Sigmoid and Tanh function are similar to each other and can be derived from each other.

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

OR

$$\mathbf{Tanh\ (x) = 2 * sigmoid(2x) – 1}$$

- This function is non-linear, and the value range lies between -1 to +1

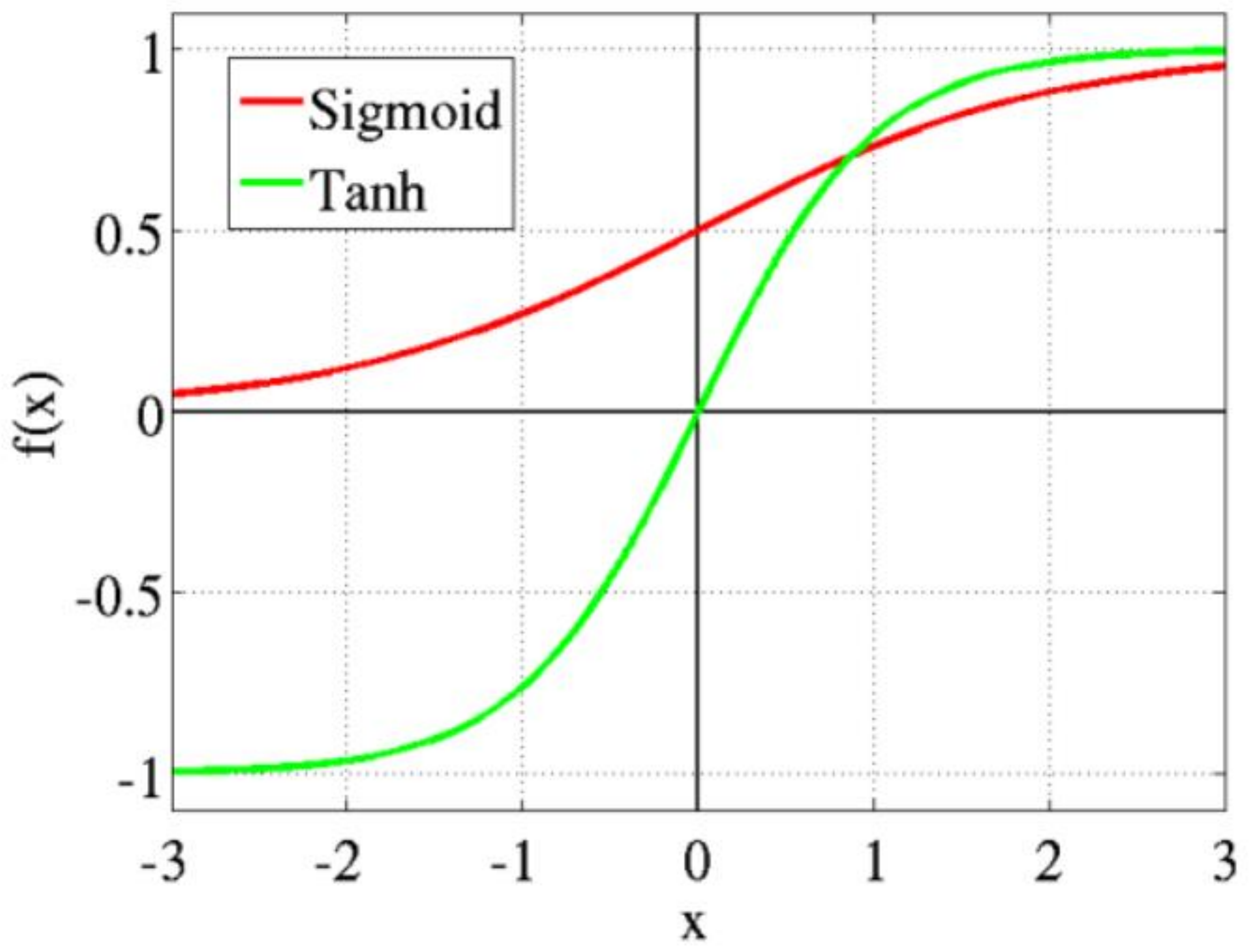
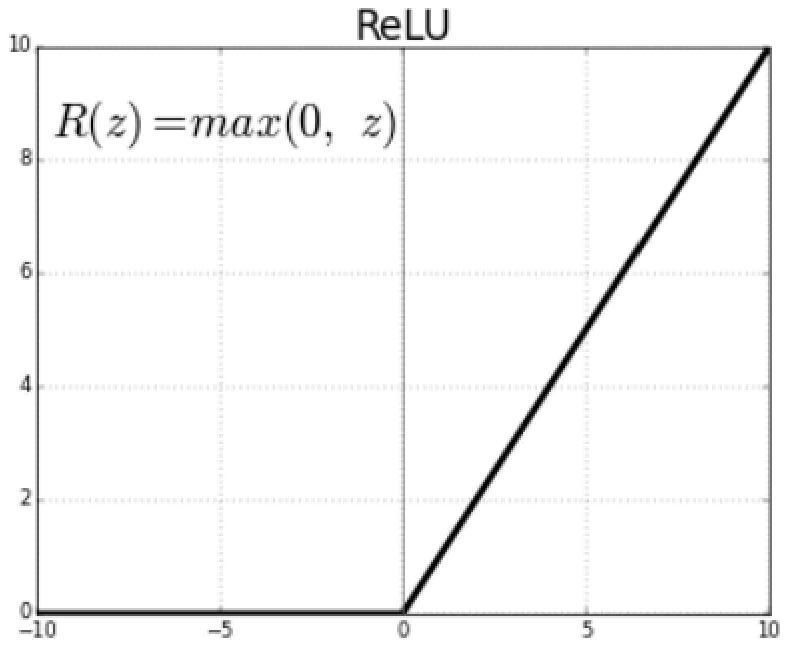


Fig: tanh v/s Logistic Sigmoid

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.



- As you can see, the ReLU is half rectified (from bottom). f(z) is zero when z is less than zero and f(z) is equal to z when z is above or equal to zero.

- **Range:** [0 to infinity)

Implementing ANN in Python

- Importing the libraries
- Importing Data

Part 1 - Data Pre-processing

Encoding categorical data

Splitting the dataset into the Training set and Test set

Feature Scaling

Part 2: Building the ANN

Initializing the ANN

Adding the input layer and the first hidden layer

Adding the output layer

Part 3: Training the ANN

Part 4 - Making the predictions and evaluating the model

LabelEncoder

Encode target labels with value between 0 and n_classes-1.
fit_transform(y):- Fit label encoder and return encoded labels

OneHotEncoder

Encode categorical features as a one-hot numeric array.

One-hot encoding can be used to transform one or more categorical features into numerical dummy features useful for training machine learning model.

A one hot encoding is **a representation of categorical variables as binary vectors.**

A Sequential model in Keras is appropriate for a plain stack of layers where each layer has exactly one input tensor and one output tensor.

TensorFlow is an open-sourced end-to-end platform, a library for multiple machine learning tasks, while Keras is a high-level neural network library that runs on top of TensorFlow.

Adam(Adaptive Moment Estimation) is **an adaptive optimization algorithm that was created specifically for deep neural network training**. It can be viewed as a fusion of momentum-based stochastic gradient descent

https://www.tensorflow.org/api_docs/python/tf/keras/Sequential

Batch size refers to the number of training instances in the batch

For example, batch_size=32 means that there are 32 training instances in each batch.

We should not get confused with batch size and the number of batches!

The number of batches is calculated as follows.

No. of batches = (Size of the entire dataset / batch size) + 1

What Is an Epoch?

The number of epochs is a hyperparameter that defines the number times that the learning algorithm will work through the entire training dataset.

What Is the Difference Between Batch and Epoch

The batch size is a number of samples processed before the model is updated.

The number of epochs is the number of complete passes through the training dataset.

ColumnTransformer

It applies transformers to columns of an array or pandas DataFrame.

Transformers: list of tuples

List of (name, transformer, columns) tuples specifying the transformer objects to be applied to subsets of the data.

By specifying remainder='passthrough' , **all remaining columns that were not specified in transformers , but present in the data passed to fit will be automatically passed through**.