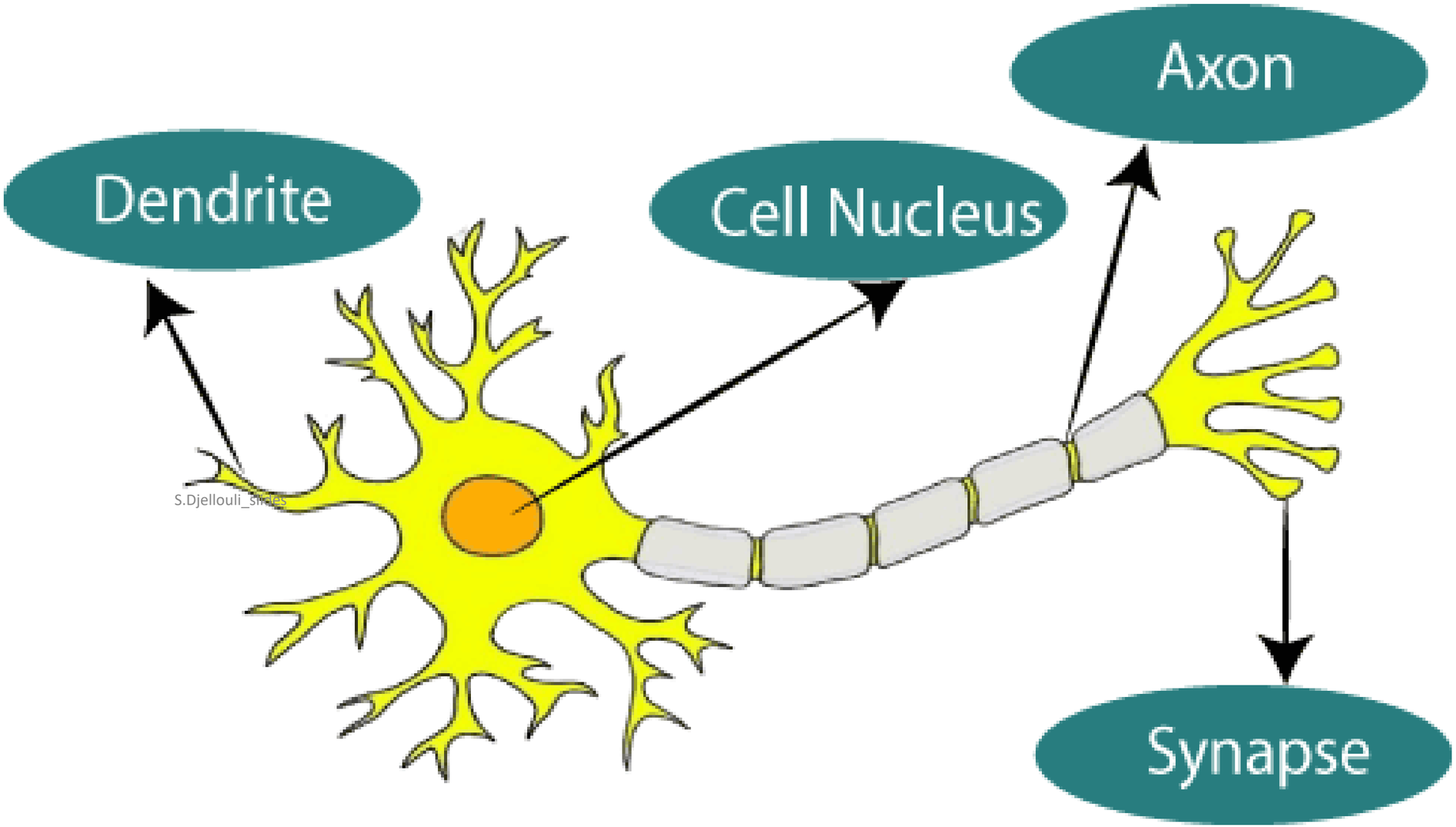
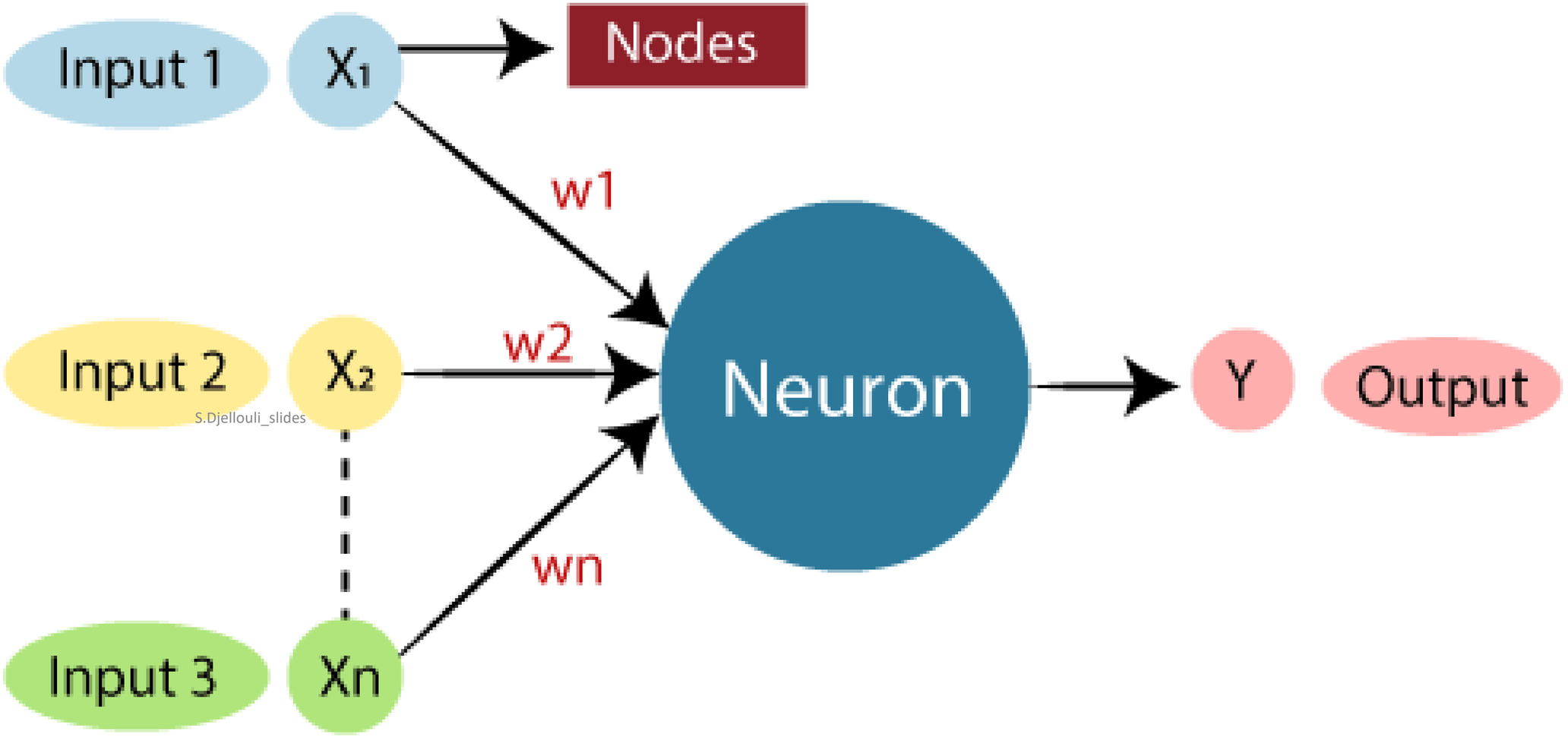
DEEP LEARNING

Artificial Neural Networks

S.Djellouli

HOW DO WE CREATE THE NEURAL NETWORKS IN COMPUTERS ?


Axon
Dendrite
Cell Nucleus
Synapse

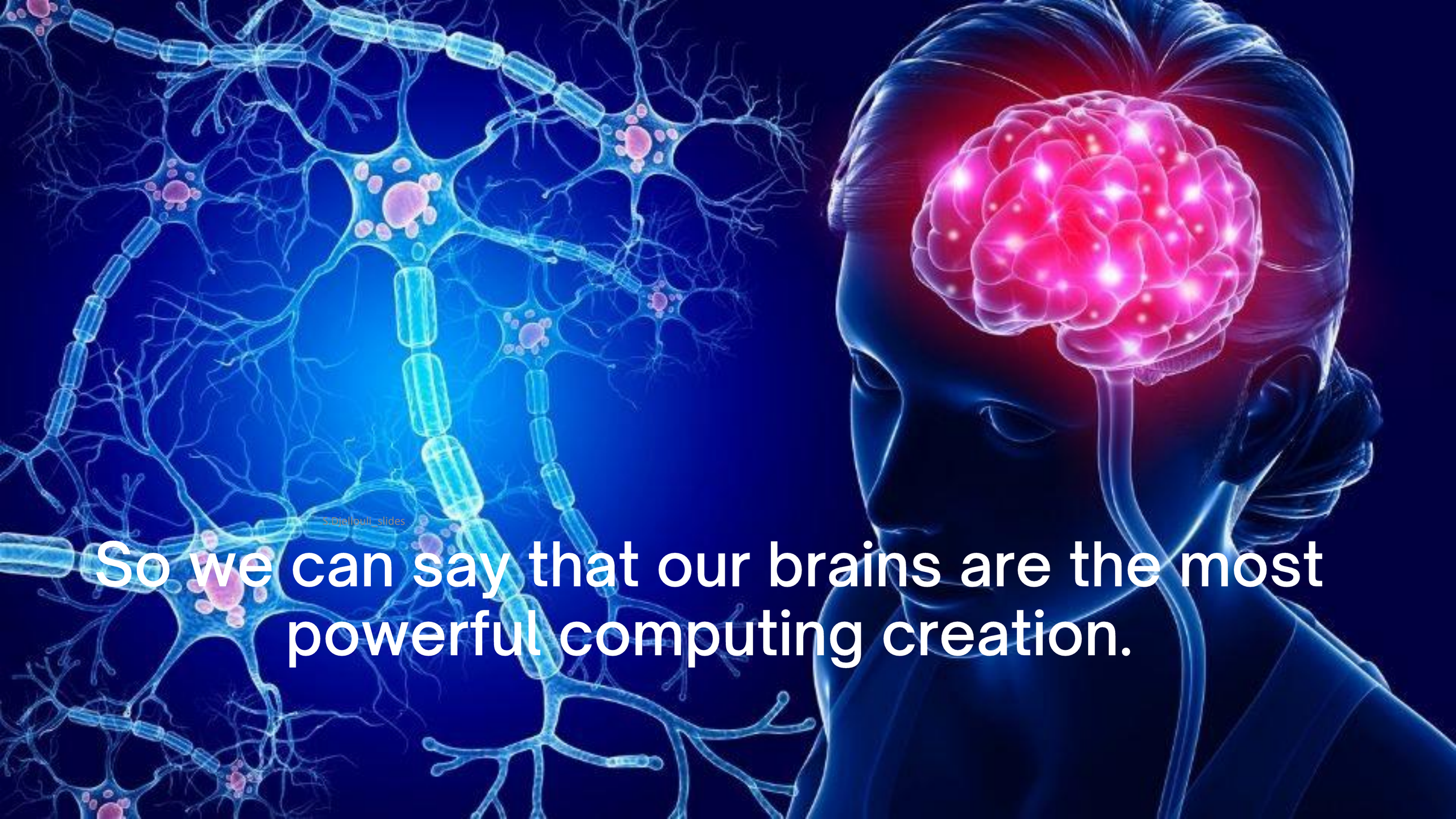

Input 1
X1
Nodes
w1
Input 2
X2
w2
Neuron
Y
Output
S.Djellouli_slides
wn
Input 3
Xn

BIOLOGICAL NEURAL NETWORK	ARTIFICIAL NEURAL NETWORK
DENDRITES	INPUTS
CELL NUCLEUS	NODES
SYNAPSE	WEIGHTS
AXON	OUTPUT

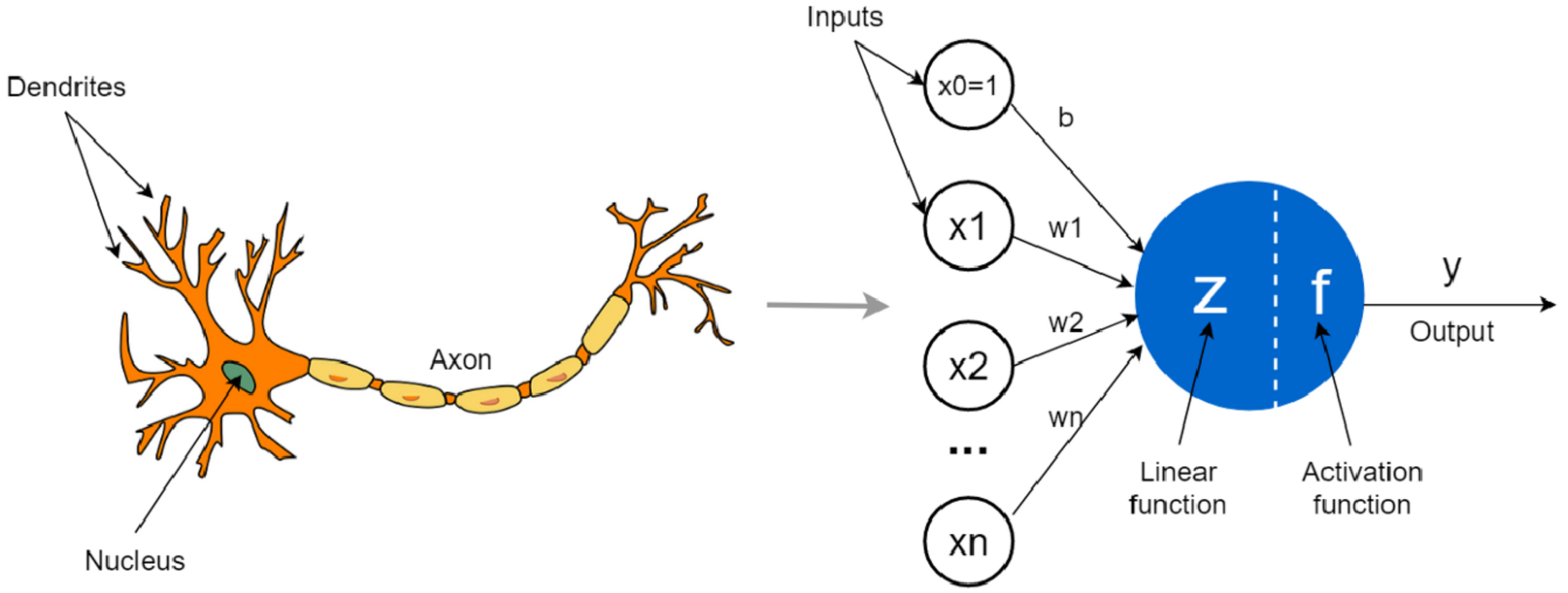
- The artificial neural network is designed by programming computers to behave simply like interconnected brain cells.

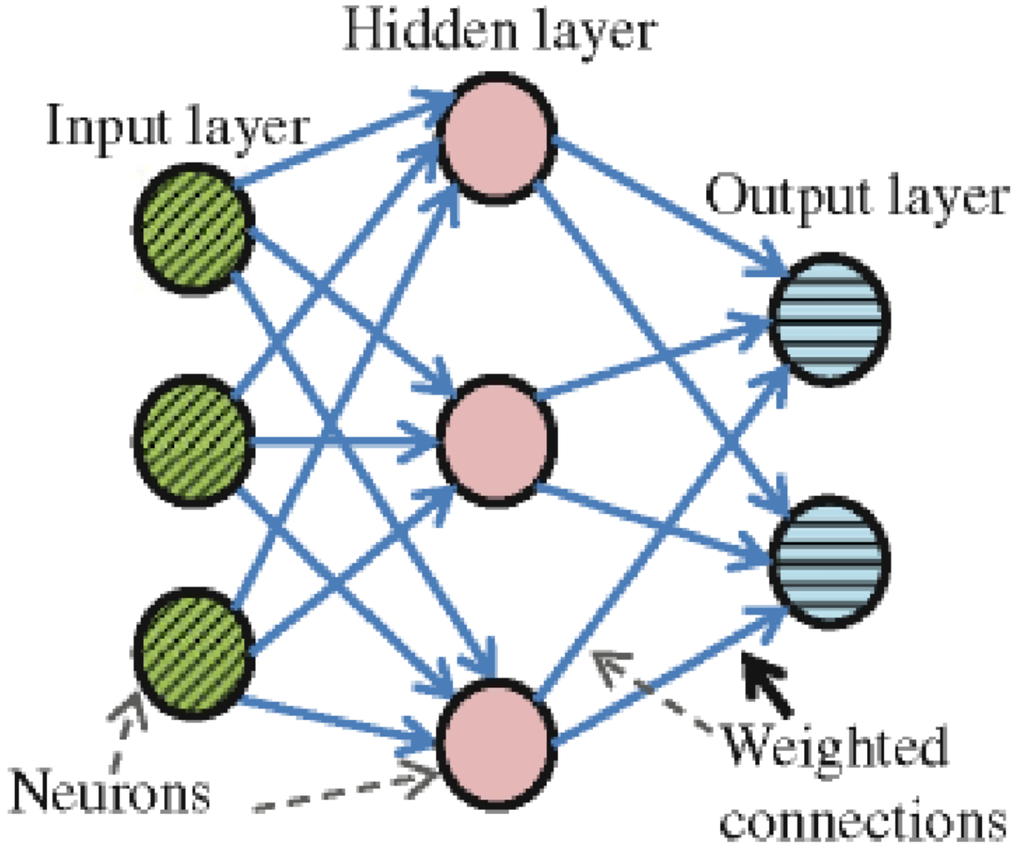
- There are around 1000 billion neurons in the human brain.

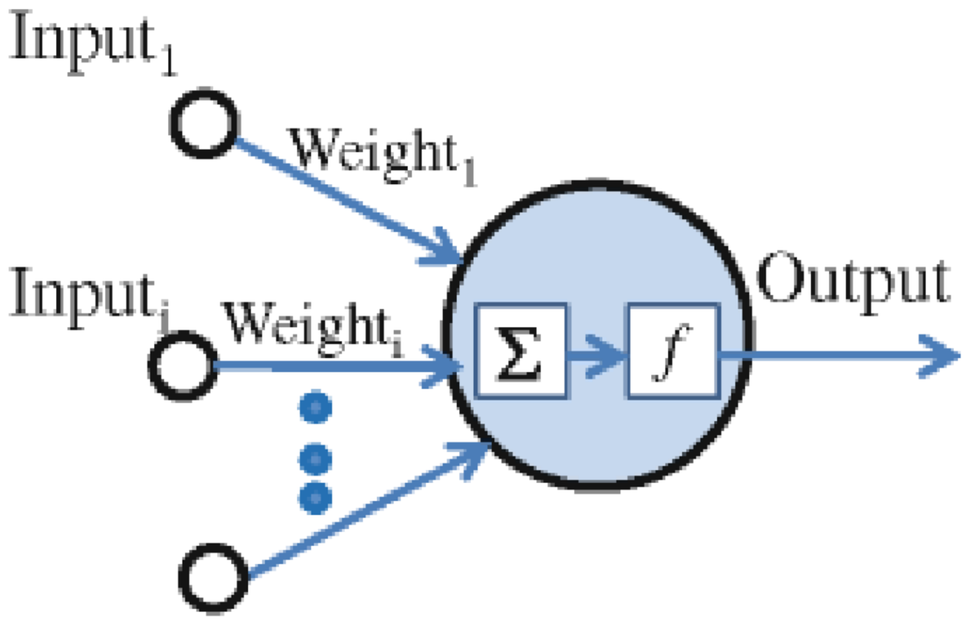
- Each neuron has an association point somewhere in the range of 1,000 and 100,000. In the human brain, data is stored in such a manner as to be distributed, and we can extract more than one piece of this data when necessary from our memory parallelly. We can say that the human brain is made up of incredibly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



So we can say that our brains are the most powerful computing creation.

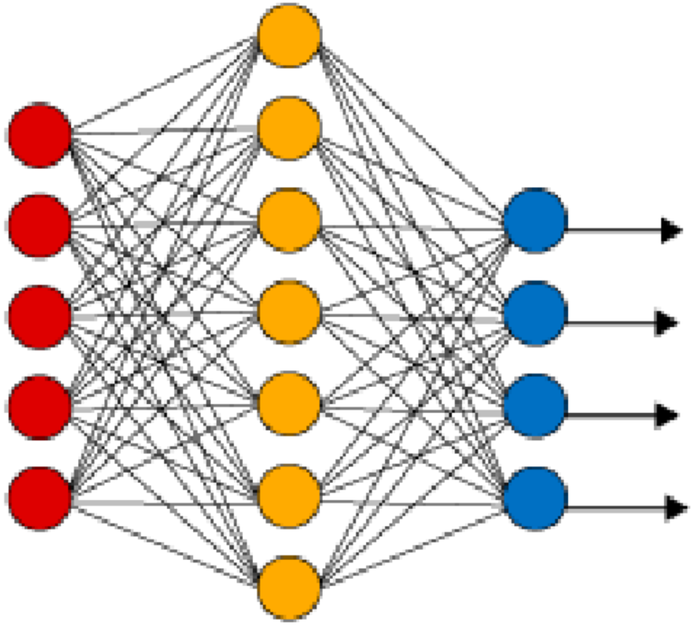

Dendrites
Axon
Nucleus
Inputs
x0=1
x1
x2
...
xn
b
w1
w2
wn
z
f
y
Output
Linear function
Activation function



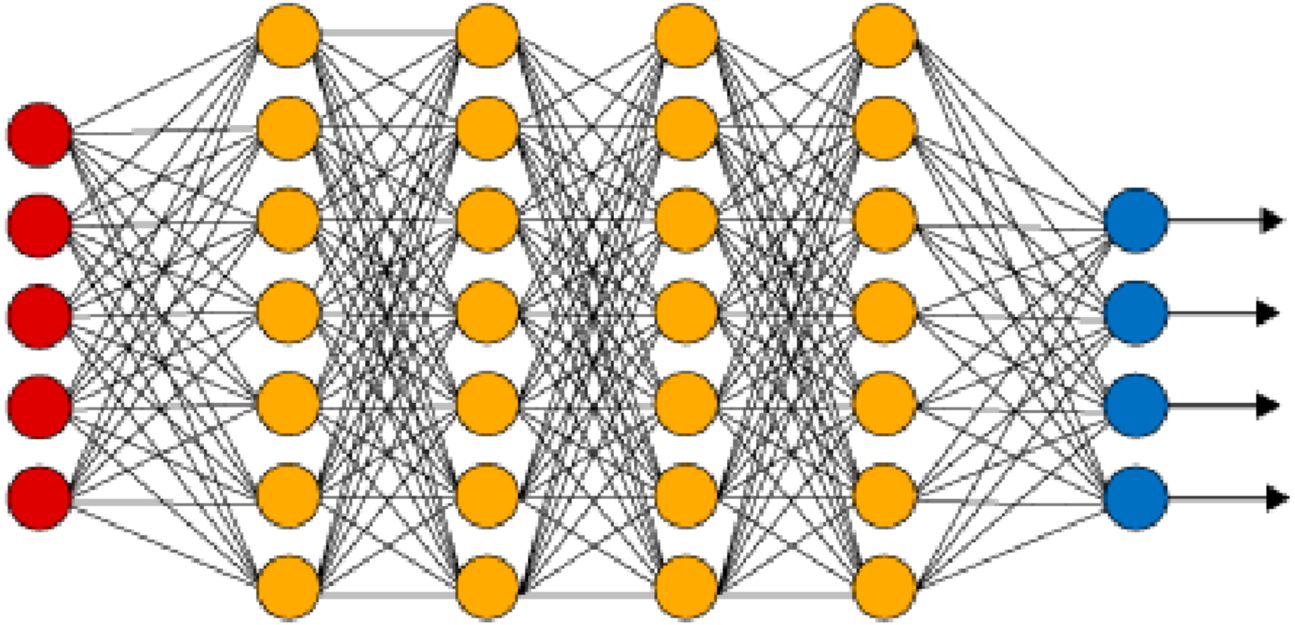


$$\text{Output} = f_{\text{transfer}}\left(\sum_i \text{Input}_i \times \text{Weight}_i\right)$$

Simple Neural Network



Deep Learning Neural Network



Input Layer Hidden Layer Output Layer

THE INPUT LAYER

Standardizing the inputs

Standardizing the inputs, also known as feature scaling or normalization, is an important preprocessing step before feeding data into an Artificial Neural Network (ANN) model.

THE INPUT LAYER

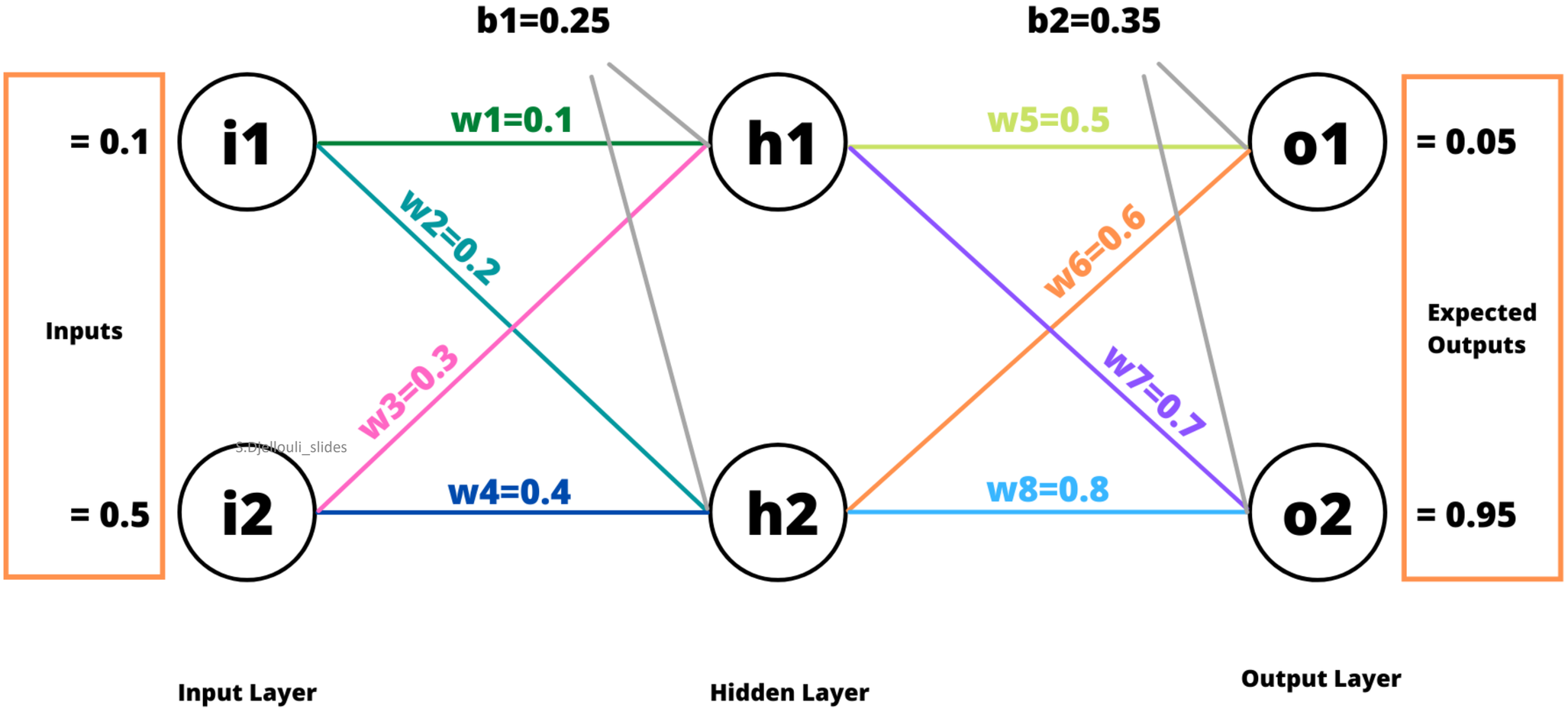
Standardizing the inputs

Overall, standardizing inputs is a good practice in machine learning to improve model performance, prevent bias, and aid in the interpretability of results. It helps the model converge faster, avoids feature dominance, and ensures fair consideration of all features during training.

FORWARD PROPAGATION

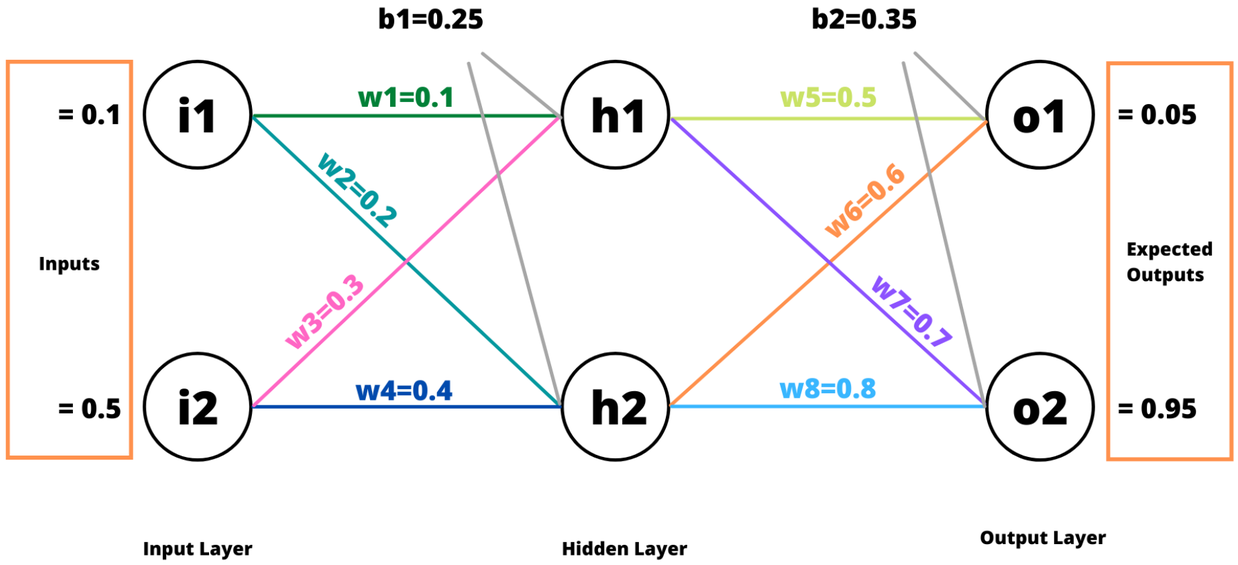
- Forward propagation refers to the process of computing the output of a neural network given a set of input values. It involves propagating the input data through the network layer by layer, from the input layer to the output layer. Here's how it works:

FORWARD PROPAGATION



FORWARD PROPAGATION

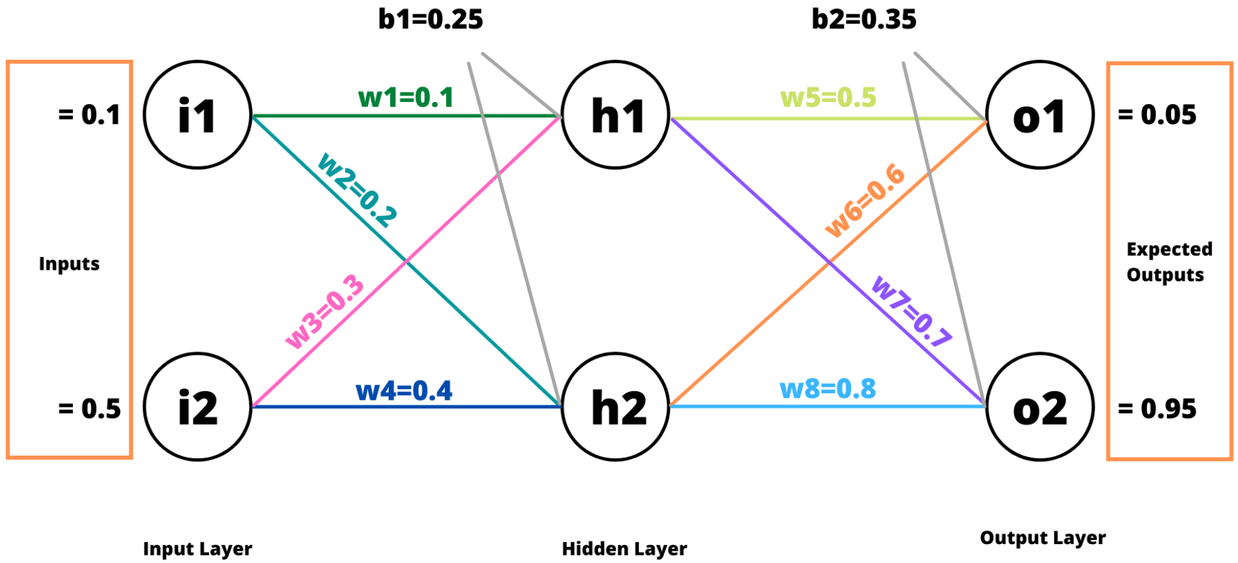
- Each neuron in a given layer receives inputs from the previous layer (or directly from the input layer).



FORWARD PROPAGATION

- The weighted inputs are summed up, including the bias term, at each neuron.

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



- z = (w1 * x1) + (w2 * x2) + ... + (wn * xn) + b

FORWARD PROPAGATION

What is Activation Function?

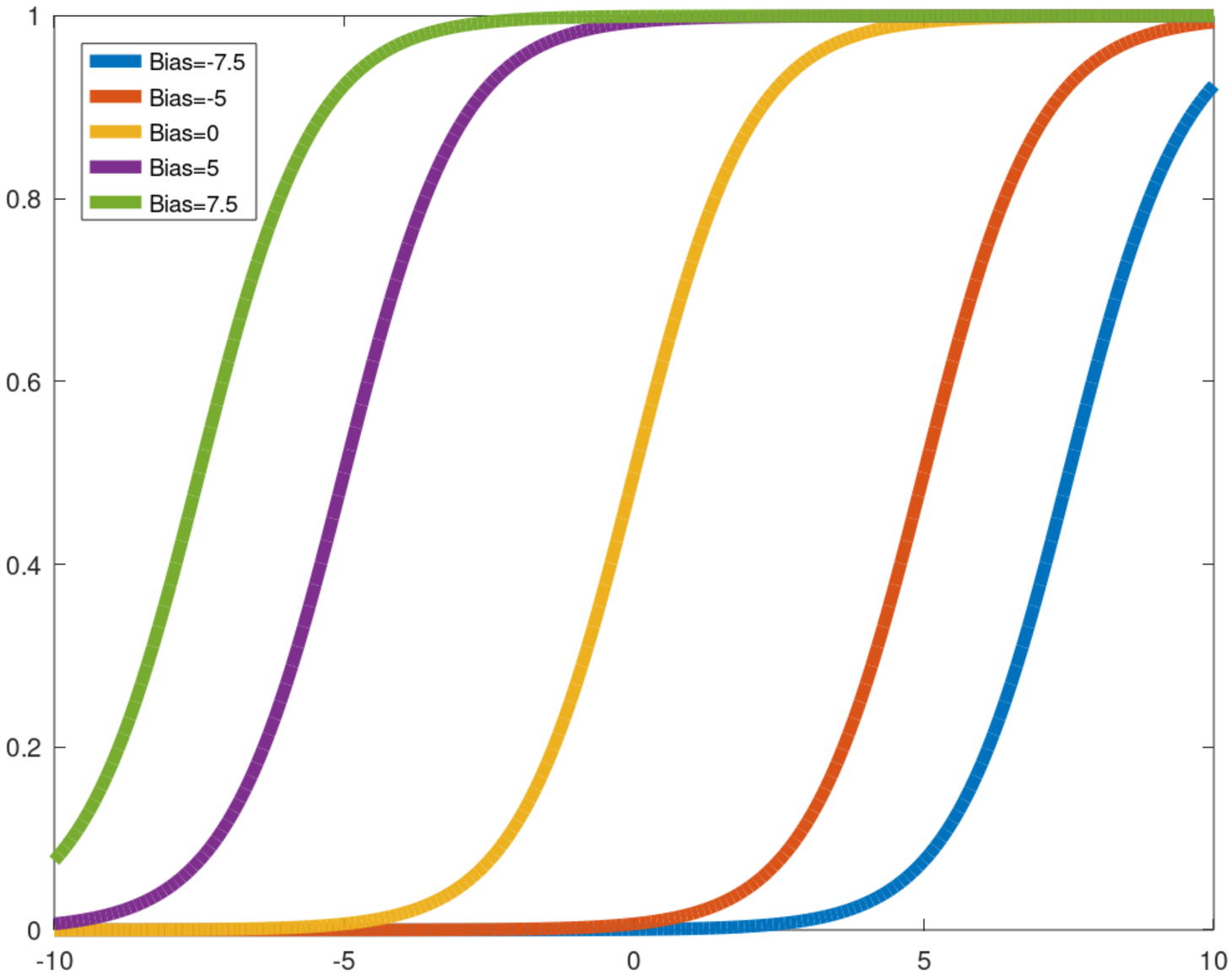
It’s just a thing function that you use to get the output of node. It is also known as Transfer Function.

FORWARD PROPAGATION

Why we need biases?

The bias term allows the network to shift the activation function's curve, affecting the neuron's output. It provides flexibility to the model to better fit the training data and capture more complex relationships between the inputs and the outputs.

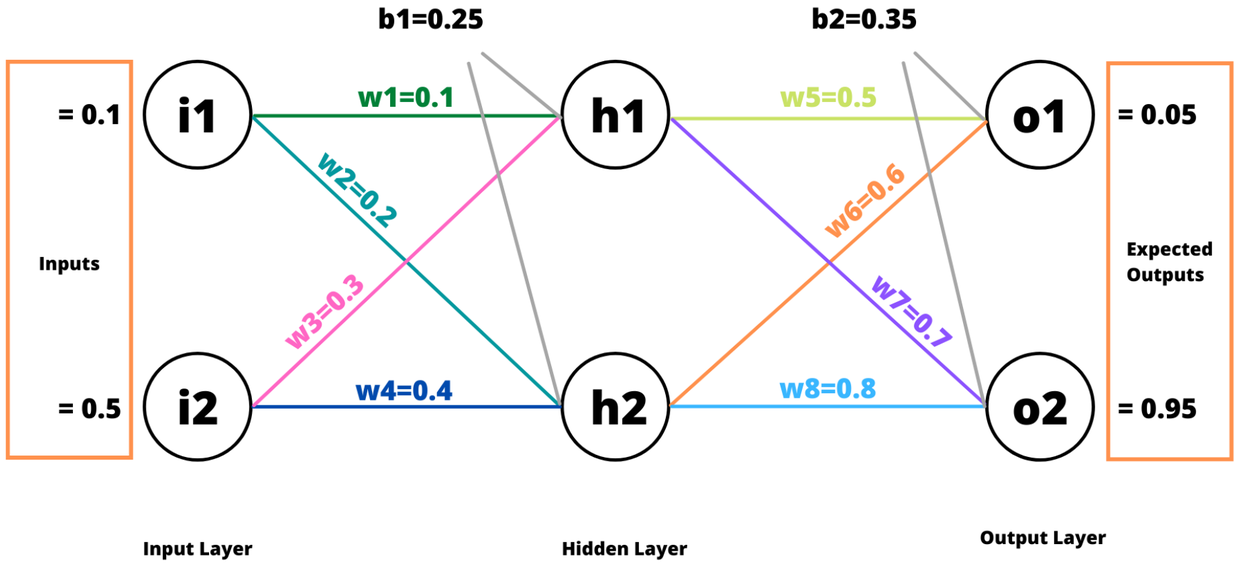
Why we need biases?



FORWARD PROPAGATION

- The weighted inputs are summed up, including the bias term, at each neuron.

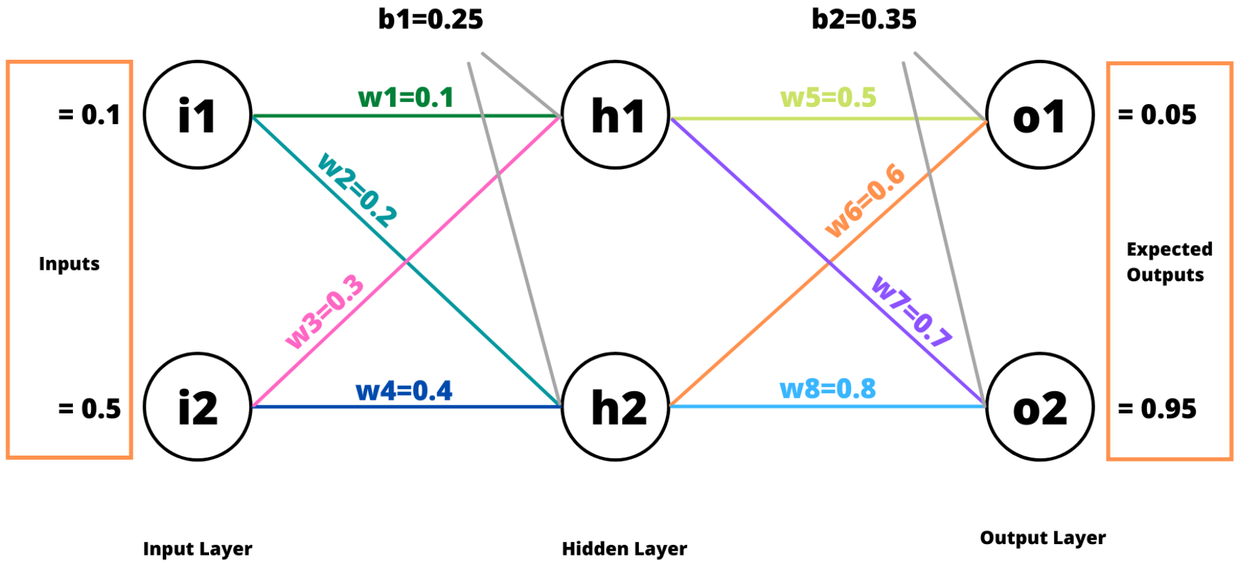
$$\sum_{i=1}^{n} Wi * Xi + b$$



- z = (w1 * x1) + (w2 * x2) + ... + (wn * xn) + b

FORWARD PROPAGATION

- The summed value is passed through an activation function to introduce non-linearity. The activation function determines the output of each neuron.



ACTIVATION = F(Z)

FORWARD PROPAGATION

What is Activation Function?

It’s just a thing function that you use to get the output of node. It is also known as Transfer Function.

Why we use Activation functions with Neural Networks?

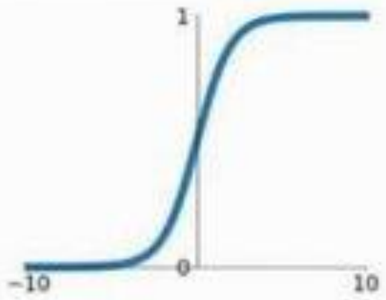
It is used to determine the output of neural network like yes or no. It maps the resulting values in between 0 to 1 or -1 to 1 etc. (depending upon the function).
The Activation Functions can be basically divided into 2 types:

- Linear Activation Function
- Non-linear Activation Functions

SOME FAMOUS ACTIVATION FUNCTIONS

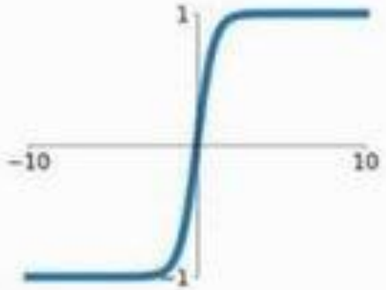
Sigmoid

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



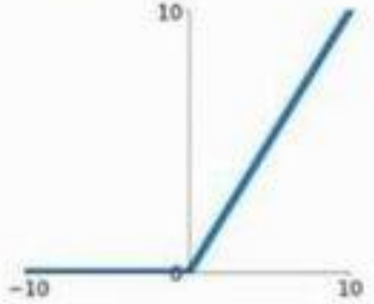
tanh

$$\tanh(x)$$



ReLU

$$\max(0, x)$$



Leaky ReLU

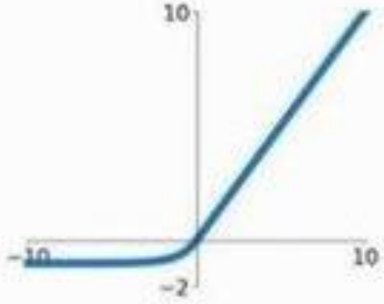
$$\max(0.1x, x)$$

Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

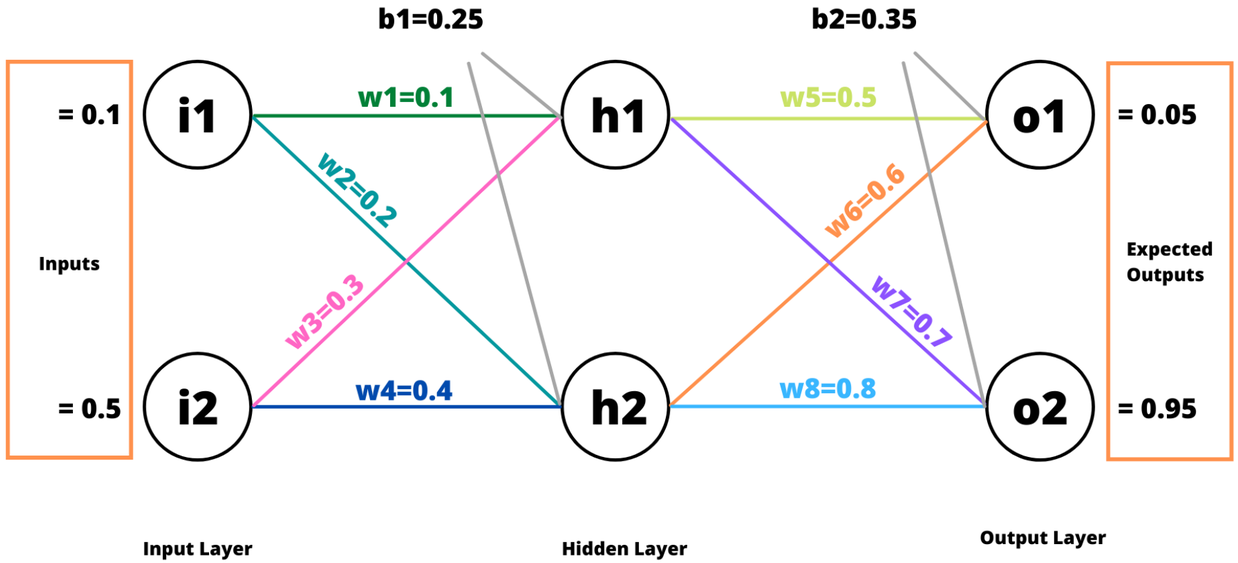
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



FORWARD PROPAGATION

- The output of each neuron becomes the input to the neurons in the next layer, and the process continues until the output layer is reached.



FORWARD PROPAGATION

- Forward propagation is crucial as it allows the network to compute predictions or outputs based on the given inputs. During the training phase, the predicted outputs are compared to the actual outputs (labels) of the training data, and the resulting error is used to update the network's weights and biases

BACKWARD PROPAGATION

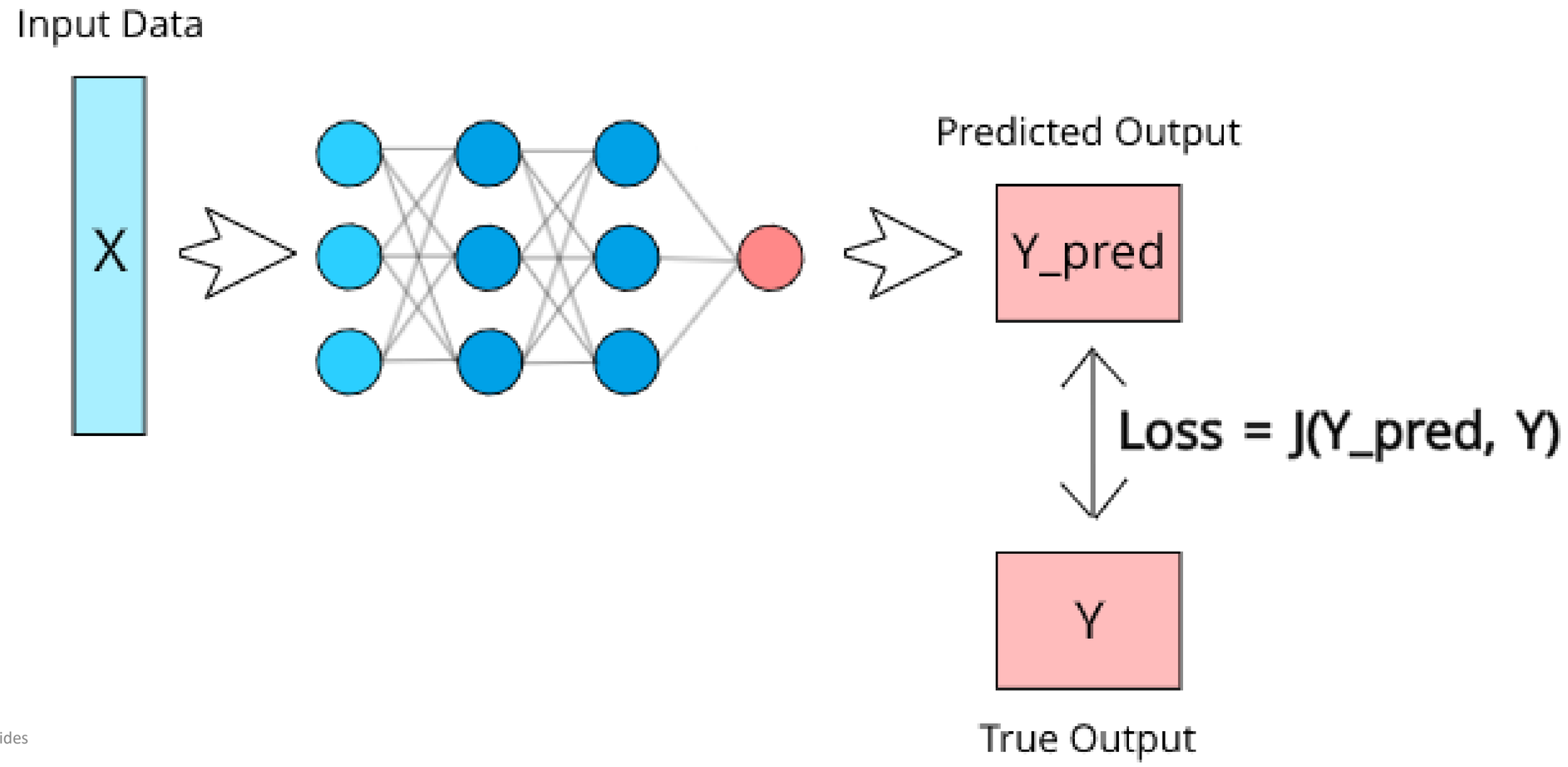
- Backward propagation, also known as backpropagation, is the process of computing the gradients of the network's weights and biases with respect to the loss or error. It involves propagating the error from the output layer back to the input layer. Here's how it works:

BACKWARD PROPAGATION

- Initially, the error or loss between the predicted outputs and the actual outputs (labels) is calculated using a loss function, such as mean squared error (MSE) or cross-entropy loss.

COST VS LOSS_FUNCTION

The cost function and loss function refer to the same context (i.e. the training process that uses backpropagation to minimize the error between the actual and predicted outcome). We calculate the cost function as the average of all loss function values whereas we calculate the loss function for each sample output compared to its actual value.

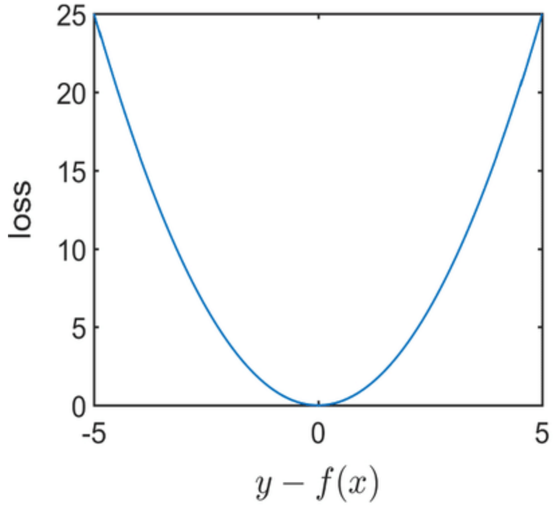

Input Data
X
Predicted Output
Y_pred
Loss = J(Y_pred, Y)
Y
True Output

LOSS_FUNCTION

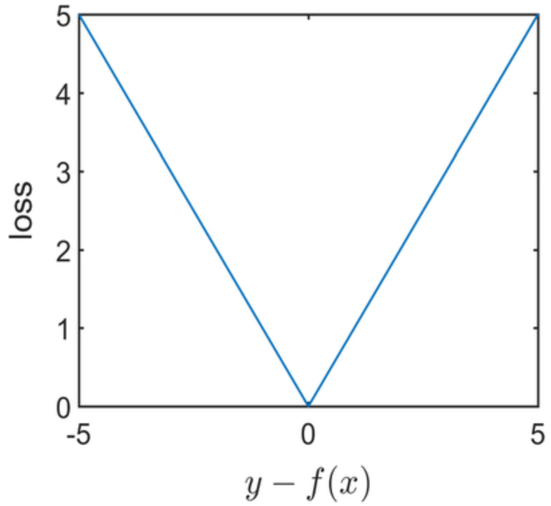
- Regression Loss Functions
- 1.1. Mean Square Error (MSE — L2)
- 1.2. Mean Absolute Error (MAE — L1)
- 1.3. Huber Loss

- Binary Classification Loss
- 2.1. Binary Cross Entropy Loss
- 2.2 Hinge Loss

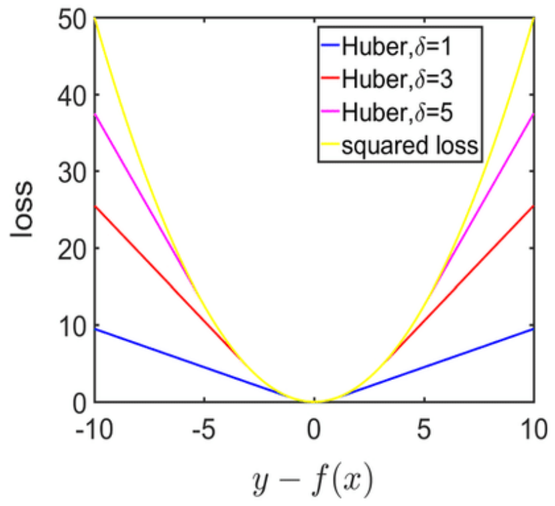
LOSS_FUNCTION



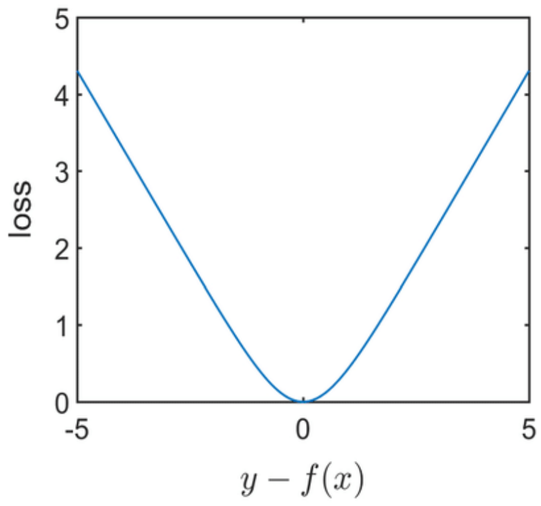
(a) square loss



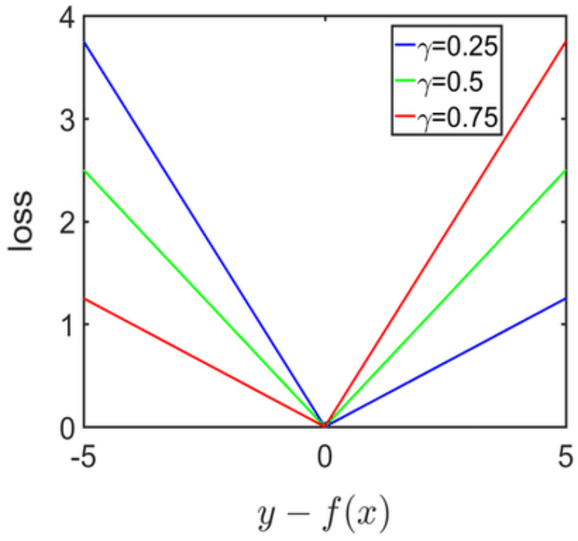
(b) absolute loss



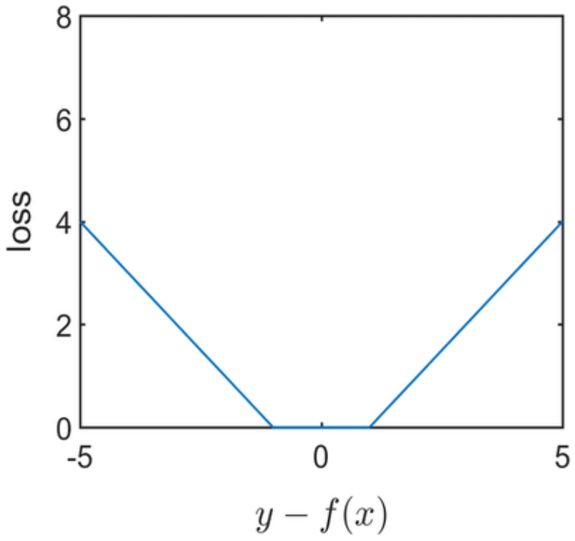
(c) Huber loss



(d) log-cosh loss



(e) quantile loss



(f) $\epsilon-$insensitive loss($\epsilon = 1$)

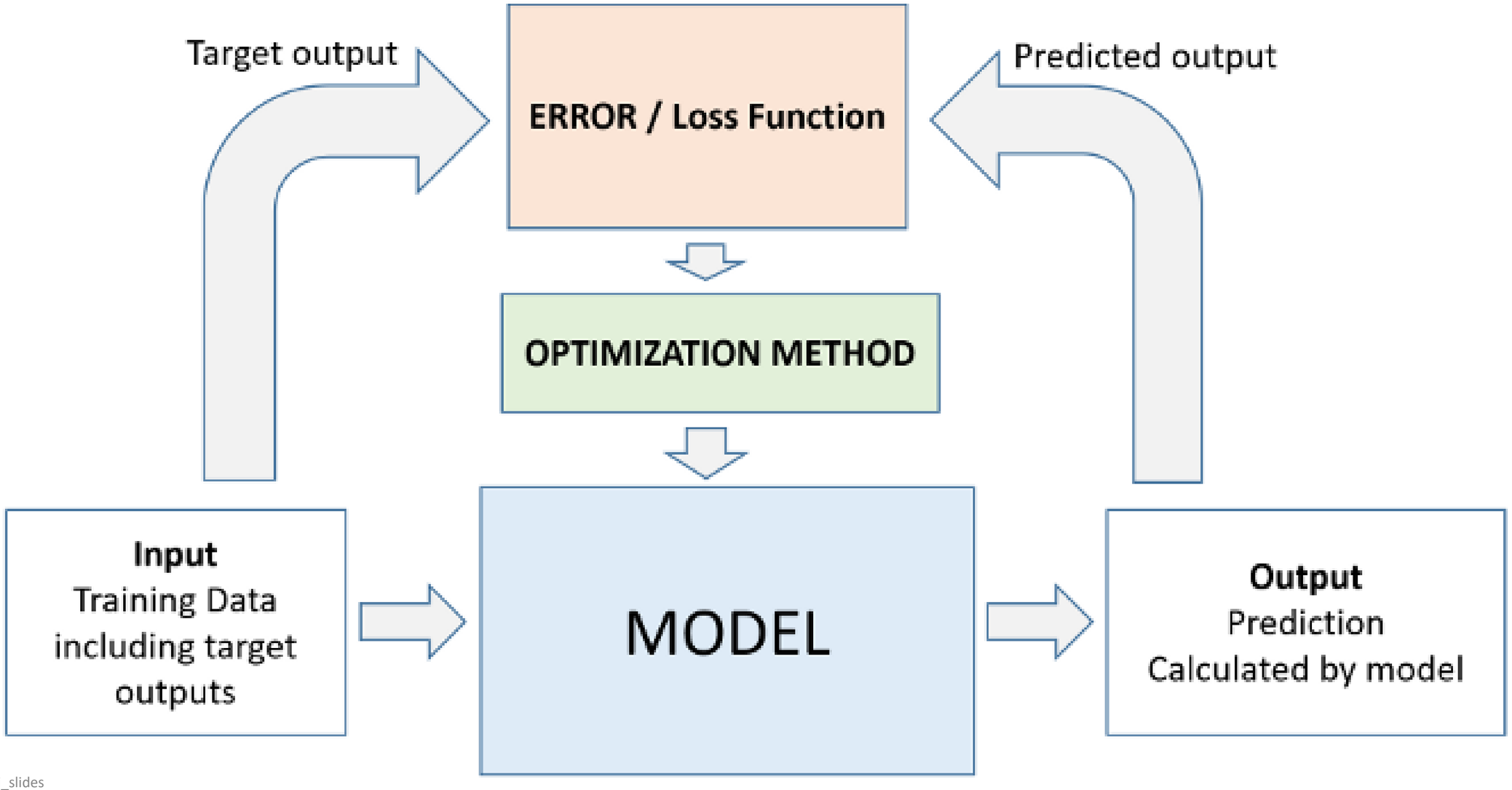
OPTIMIZATION

- Optimization in deep learning refers to the process of finding the optimal set of parameters or weights for a neural network model that minimizes a given objective function or loss function. The objective is to train the model to make accurate predictions or perform a specific task effectively.

OPTIMIZATION

Deep learning models are typically trained using a variant of gradient-based optimization algorithms, the most popular being stochastic gradient descent (SGD) and its many variations.

OPTIMIZATION

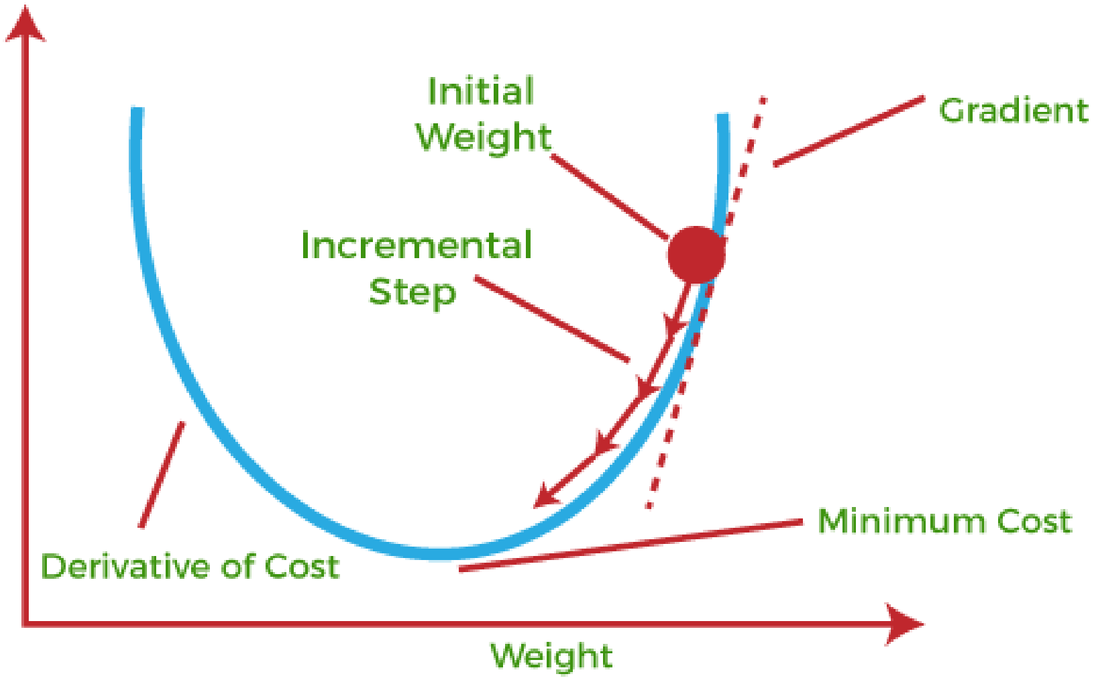


OPTIMIZATION_STEPS

- The gradients of the loss function with respect to the weights and biases are computed using techniques like gradient descent.
- The gradients are then propagated backward through the layers of the network, using the chain rule of calculus. At each layer, the gradients are multiplied by the derivatives of the activation functions to obtain the gradients of the weighted inputs.
- The gradients are used to update the weights and biases of the network, iteratively improving the network's ability to make accurate predictions.

OPTIMIZATION_STEPS

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.



HOW DOES GRADIENT DESCENT WORK

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.

HOW DOES GRADIENT DESCENT WORK

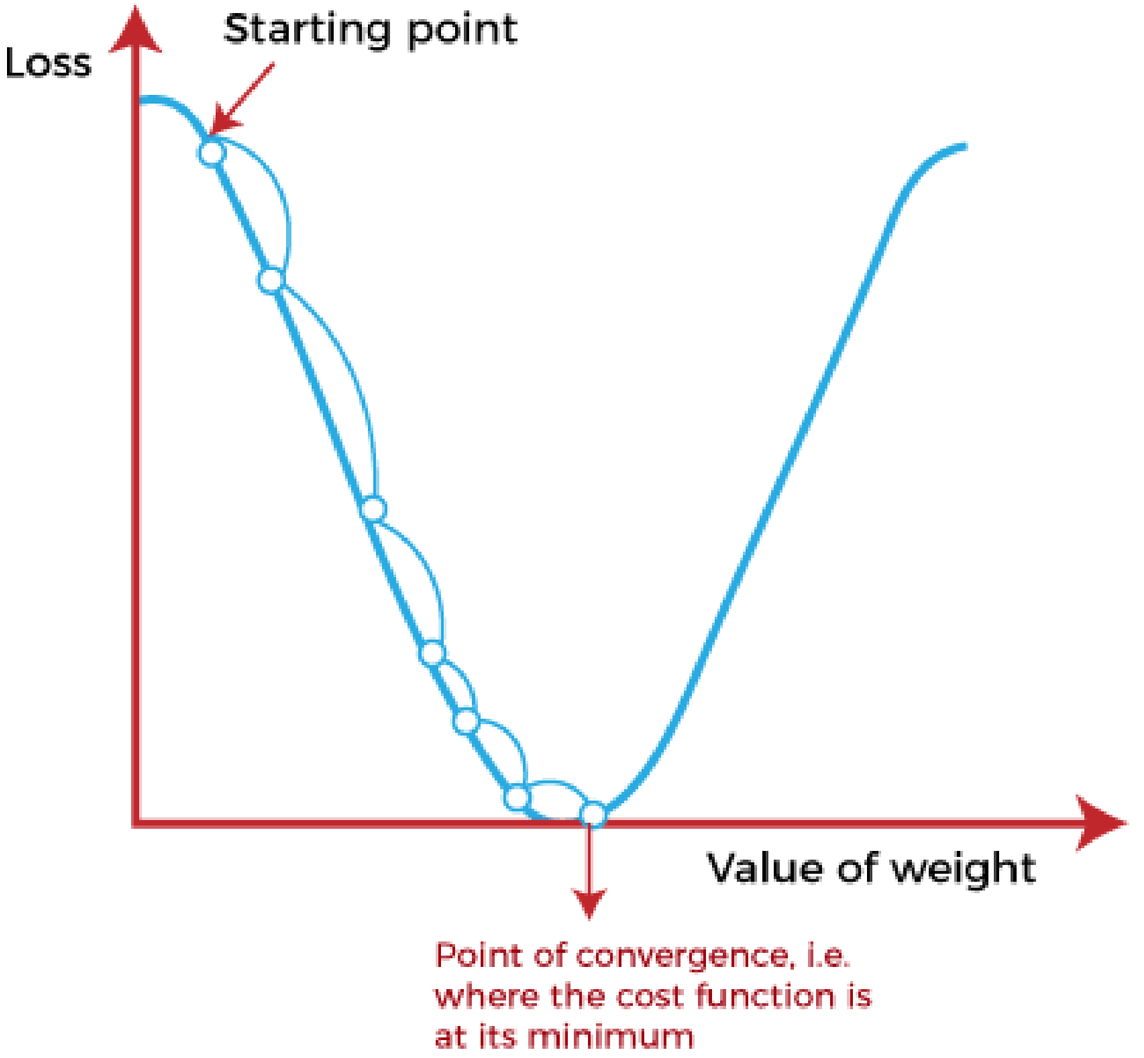
The best way to define the local minimum or local maximum of a function using gradient descent is as follows:

- 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.

HOW DOES GRADIENT DESCENT WORK

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.


Loss
Starting point
Value of weight
Point of convergence, i.e.
where the cost function is
at its minimum

HOW DOES GRADIENT DESCENT WORK