# Introduction to Neural Network

Backpropagation is a method to **update the weights in the neural network** by considering the **actual output and the desired output.**

Backpropagation neural networks can be divided into two steps: feedforward and Backpropagation.

- In the feedforward step, an input pattern is applied to the input layer and its effect **propagates, layer by layer**, through the network until the **output** is produced.
- The network's actual output value is then compared to the expected output, and an **error signal** is computed for each of the output nodes

## Common Interview Questions

- What are key components of neuron ?
- What is forward propagation ?
- What is backword propagation ?
- Why Deep Learning is Non-Convex Optimization problem ?
- What is impact of bias ?
- What are similarities to human brain and back propagation ?

## CNN Key Concepts

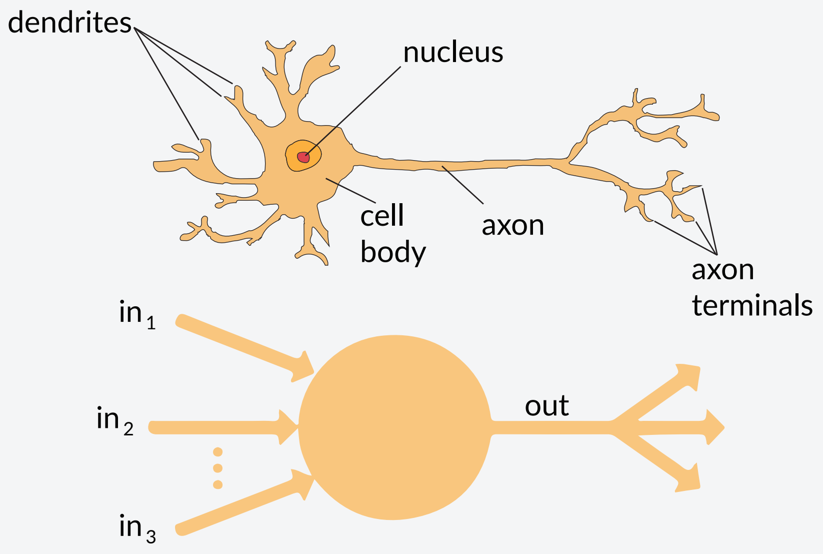
- Forward propagation
- Back Propagation
- Gradient Descent
- Activation Function
- Drop outs

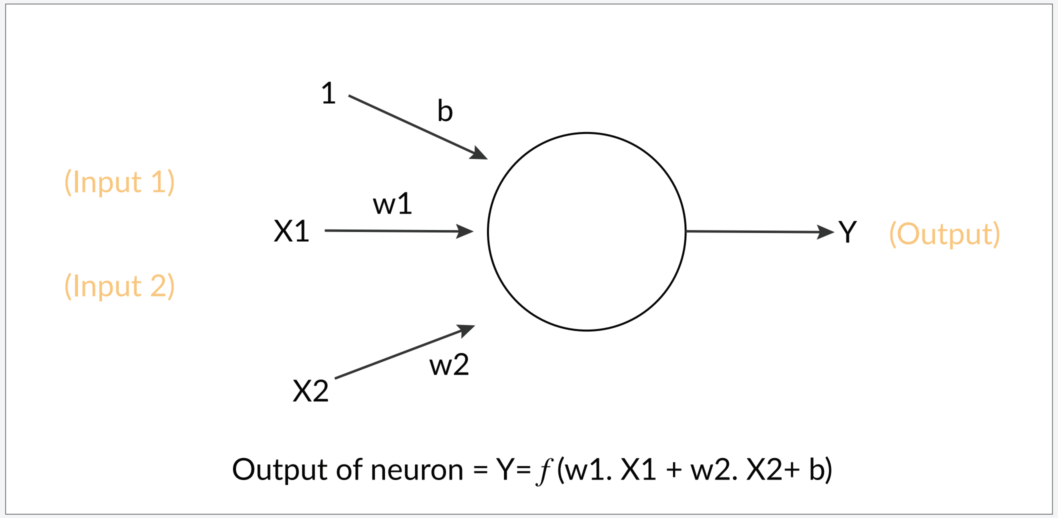
# Human Brain Neuron

**Neurons** (also called neurons or nerve cells) are the fundamental units of the brain and nervous system

Key Components of Neuron

- **Dendrites** - Dendrites bringinformation to the cell body. The receiving part of the neuron
- **Axon** - axons take information away from the cell body.
- **Synapse** - Information from one neuron flows to another neuron across a synapse





Output of neuron = $Y = f(w1.\,X1 + w2.\,X2 + b)$

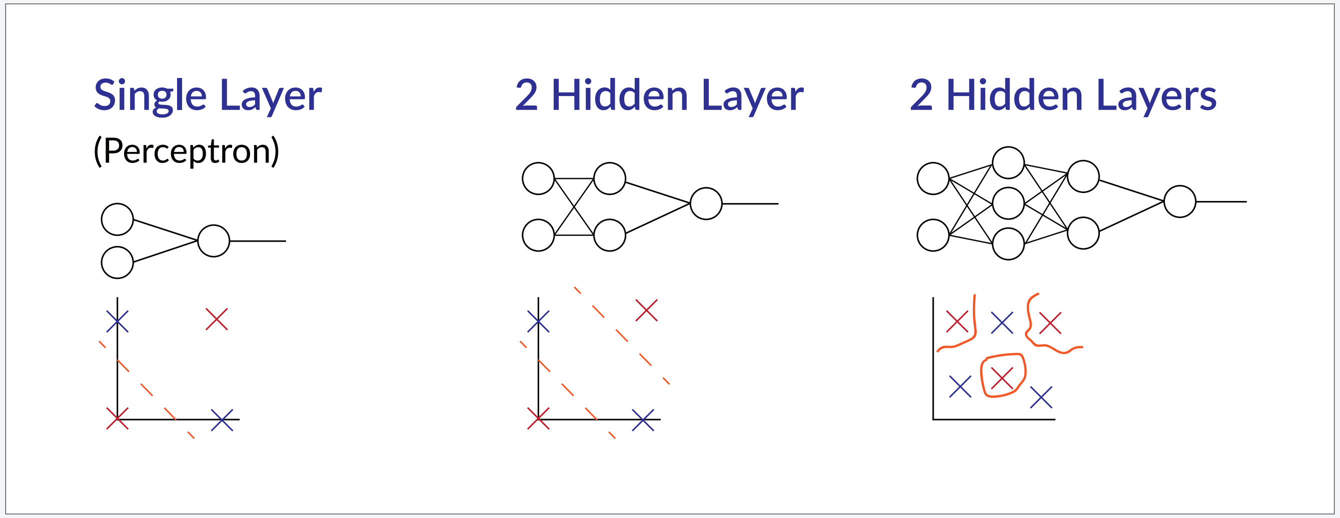
**Perceptron** is a type of artificial neuron

- Perceptron's were developed in the 1950s and 1960s by a scientist Frank Rosenblatt.
- A perceptron takes several binary inputs, x1, x2 ,...., and produces a single binary output
- Single-Layer Perceptrons cannot classify non-linearly separable data points.

# Multi Layer Perceptron

Perceptron works as a linear classifier.

- Single-Layer Perceptrons cannot classify non-linearly separable data points.
- Complex problems, that involve a lot of parameters cannot be solved by Single-Layer Perceptrons



**Neural networks (NN) are universal function approximators**

- One advantage of a Neural Network is the functioning of leveraging the power of **non-linearity** to interpret input information, which most traditional machine learning approaches lack. Neurons collectively form a neural network
- NN guarantees convergence to a local minimum instead of to the global minimum.
- The composition is via multiplications of functions of the **weights/biases and that is the main culprit for non- convexity**
- Non-convex optimization may have multiple locally optimal points. Hence, finding the global minimum is very difficult.

**What makes non-convex optimization hard?**

- Potentially many local minima, Saddle points, Very flat regions, and Widely varying curvature

**How to solve non-convex problems?**

- Stochastic gradient descent, Mini-batching, SVRG, Momentum

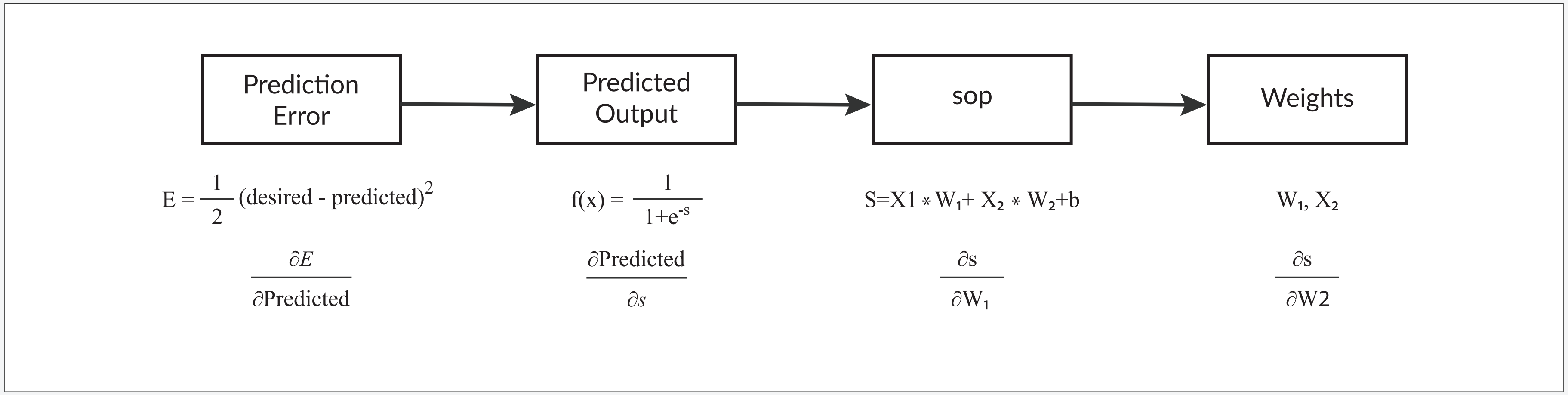
# Backpropagation Derivation

**Key Steps of Backpropagation**

- Step 1: Take a batch of training data and perform **forward propagation to compute the loss.**
- Step 2: **Backpropagate the loss to get the gradient of the loss with respect to each weight.**
- Step 3: **Use the gradients** to update the weights of the network.

Backpropagation is a method to update the weights in the neural network by considering the actual output and the desired output. The derivative with respect to **each weight w** is computed using the chain rule

**Backward propagation = chain rule differentiation. AI is nothing but the usage of math applied creatively for computing.**


Prediction Error
Predicted Output
sop
Weights
$E = \frac{1}{2}(\text{desired - predicted})^2$
$f(x) = \frac{1}{1+e^{-s}}$
$S=X1 * W_1 + X_2 * W_2 + b$
$W_1, X_2$
$\frac{\partial E}{\partial \text{Predicted}}$
$\frac{\partial \text{Predicted}}{\partial s}$
$\frac{\partial s}{\partial W_1}$
$\frac{\partial s}{\partial W2}$

**Prediction Error =** Difference between expected and predicted output, Differentiation of Error function
**Predicted Output -** The Differentiation of Activation Function, In this case, the Sigmoid function
**SOP - Sum of products (SOP)** between each input and its corresponding weight W1
**Weights -** Compute gradient and newly updated weight with learning rate

$$w \leftarrow w - \eta \frac{\partial C}{\partial W}$$

The gradient descent method involves calculating the derivative of the loss function with respect to the network's weights. This is normally done using backpropagation.

# Activation Function Insights

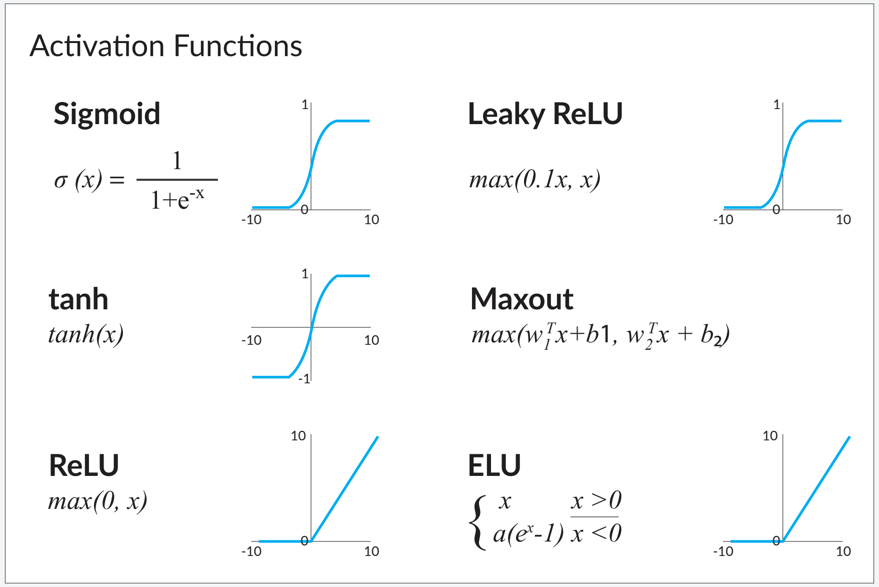
## Activation Functions

The activation function serves as a gate between the current neuron input and its output. Activation functions are added to introduce non-linearity to the network

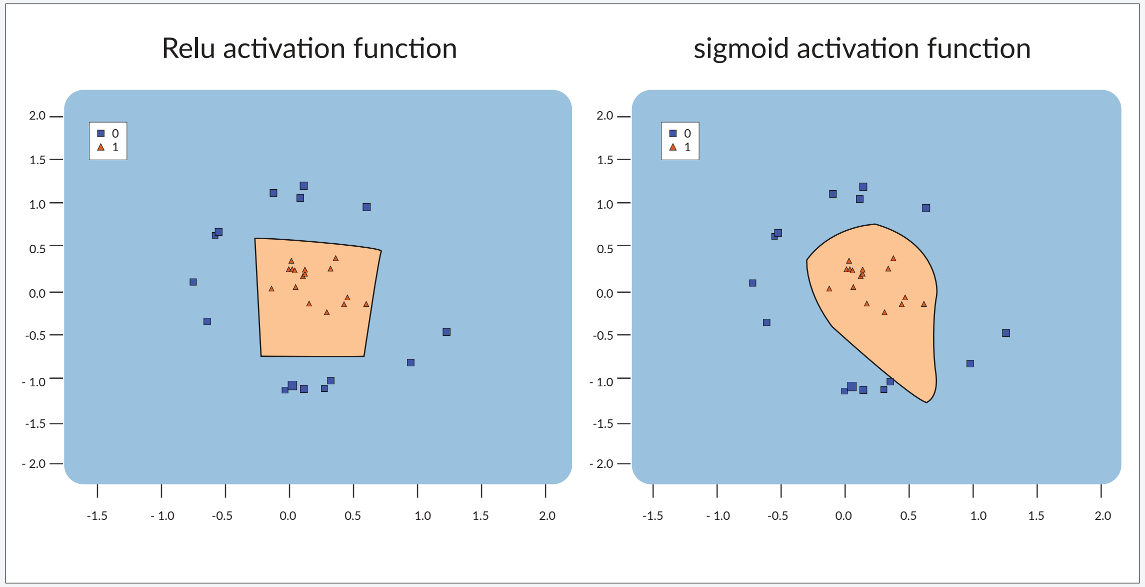
**Gradient** - The gradient is the partial derivative of a function that takes in multiple vectors and outputs a single value (i.e. our cost functions in Neural Networks). The gradient tells us which direction to go on the

graph to increase our output if we increase our variable input.

Back prop, this is the process of back tracking errors through the weights of the network after forward propagating inputs through the network. Below Example different boundary based on activation functions



Activation Functions
Sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$
Leaky ReLU
$max(0.1x, x)$
tanh
$tanh(x)$
Maxout
$max(w_1^T x + b1, w_2^T x + b_2)$
ReLU
$max(0, x)$
ELU
$\begin{cases} x & x>0 \\ a(e^x-1) & x<0 \end{cases}$
-10
10
0
1
-1



# Weights update

The gradient descent method involves calculating the derivative of the loss function with respect to the weights of the network. This is normally done using backpropagation.

- Use chain rule to compute derivatives through network

$$\frac{\partial E\,(0)}{\partial W_i} = \left(\frac{\partial E\,(0)}{\partial y\,(x)}\right)\left(\frac{\partial y\,(x)}{\partial z}\right)\left(\frac{\partial z}{\partial W_i}\right)$$

- change of error function with network output (cost function)
- change of network output with z (activation function)
- change of z with network weight (parameter to estimate)function)

W(new) = W(old) — (learning factor)*(Differentiation of the Objective Function with respect to W)

- **Batch Gradient Descent** is when we sum up over all examples on each iteration when performing the updates to the parameters. Therefore, for each update, we have to sum over all examples.
- **Mini-batch Gradient Descent** - Instead of going over all examples, Mini-batch Gradient Descent sums up over lower number of examples based on batch size.
- **Stochastic Gradient Descent (SGD)** - Instead of going through all examples, SGD performs the parameters update on each example

**Regularization**

- **Regularization** is the process of adding information in order to solve an ill-posed problem or to prevent overfitting
- L2 regularization: complexity = sum of squares of weights Combine with L2 loss to get ridge regression
- L2 regularization leads to small weights, •L1 regularization leads to many zero weights (sparsity)