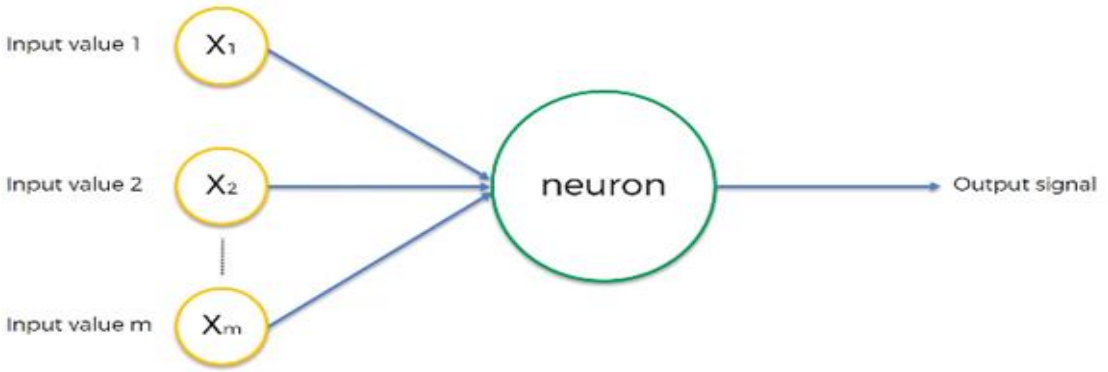
Artificial Neural Network in a Nutshell

The whole purpose of Deep Learning is how we can mimic the human brain. In this article, we will learn about Artificial Neural Network, understand its concept in crisp and then we will implement it in Python using Tensorflow 2.0
Let's get started.

Let's start with the basics. The following article talks in detail about what a Neuron is and its structure.

The Neuron:

Neurons are the fundamental units of the brain and nervous system. So how is it useful in Deep Learning? We will try to mimic the working of a neuron in Deep Learning, which is the preliminary step in understanding deep learning.



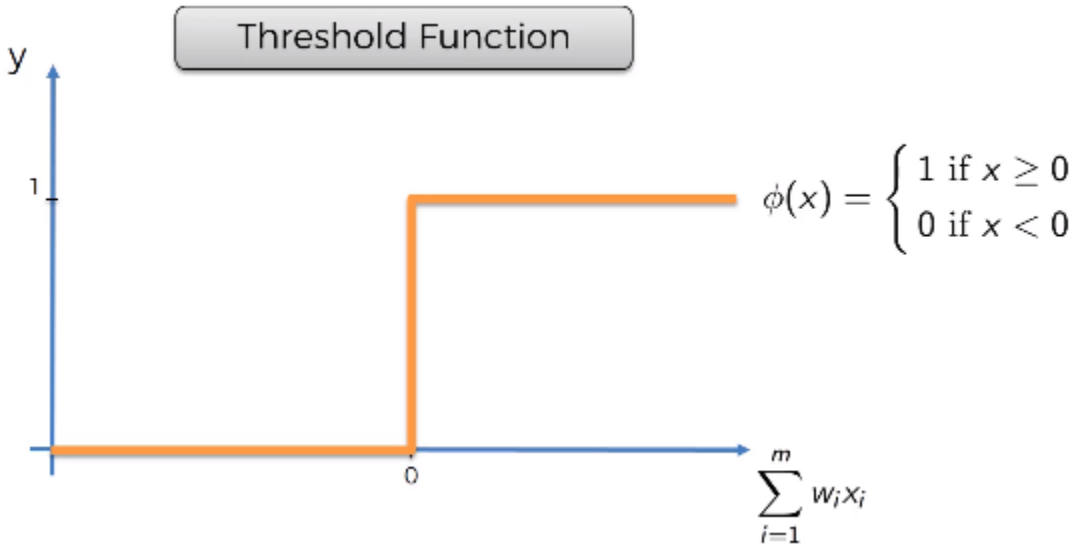
In a neural network, a neuron receives input data to do some calculations on it and gives the output. Taking a simple example of Machine Learning we can say that Input values X1, X2, X3 are different independent variables in our dataset. These are the single row of our dataset. One thing we should note down regarding the input variables is that our input data should be standardize meaning the data should have a mean of zero and variance of one (sometime we might need to normalized data instead of standardizing the data, by normalizing we mean that transforming the data between values 0 and 1).
Now, talking about the output from the neuron. The output from the neuron can be a continuous ex price, it can be binary ex. Yes/No or it can be categorical. In the case of categorical data as output, we will have multiple categories of output from neurons instead of single output.

So, in a neuron, we give input to a neuron as a single row of our dataset and we get the output for that single row from the neuron. In a neural network, multiple neurons are connected to each other which transfer data to one another, input data to one neuron will be the output of another neuron.

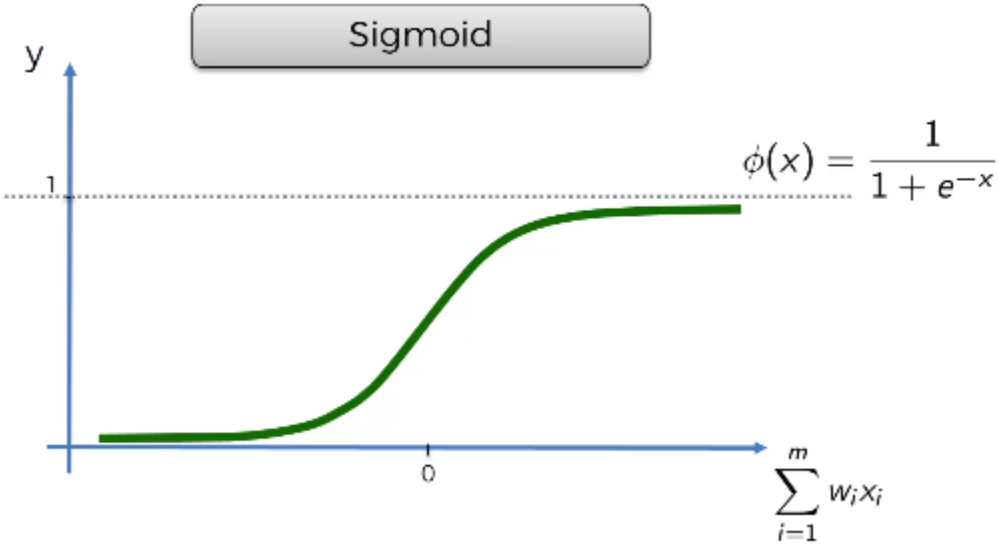
The Activation Function

In very simple terms, consider a neuron performing some calculations and producing an output Y, this output Y is input to other neurons so Activation Function helps to check whether the value Y produced by a neuron should be considered or not by the other neuron. In other terms, we say a neuron is activated or not.
There different types of Activation Functions:

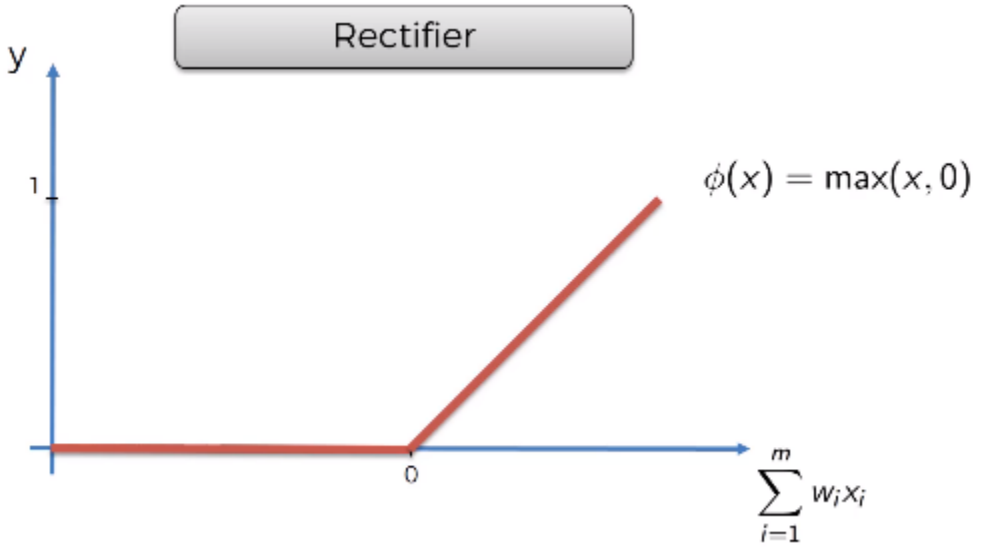
1. **Threshold Function:**



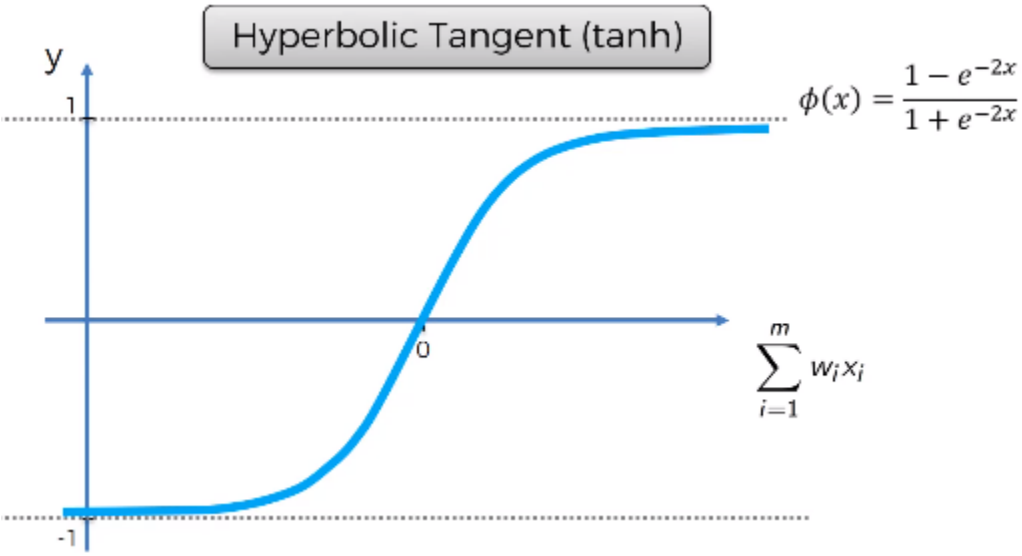
2. **Sigmoid Function:**



3. **Rectifier Function:**

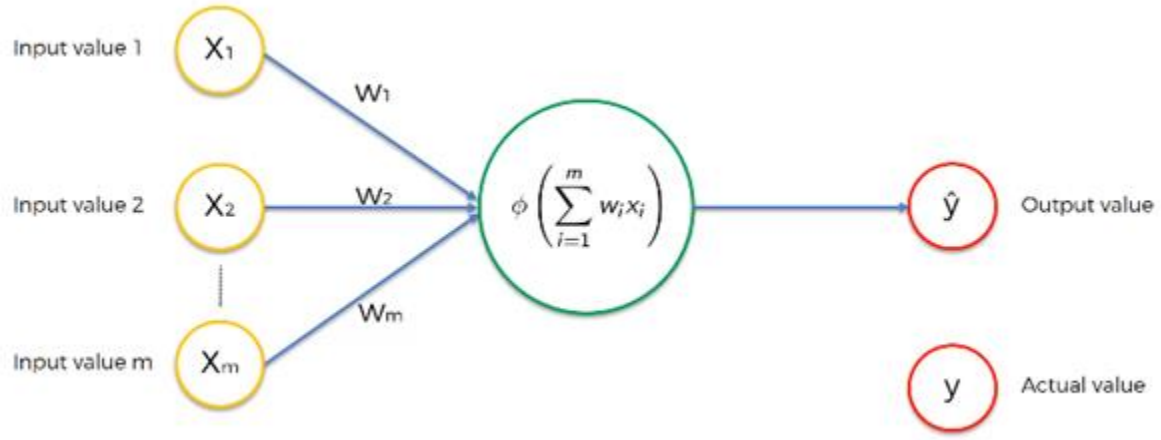


4. **Hyperbolic Tangent (tanh) Function:**

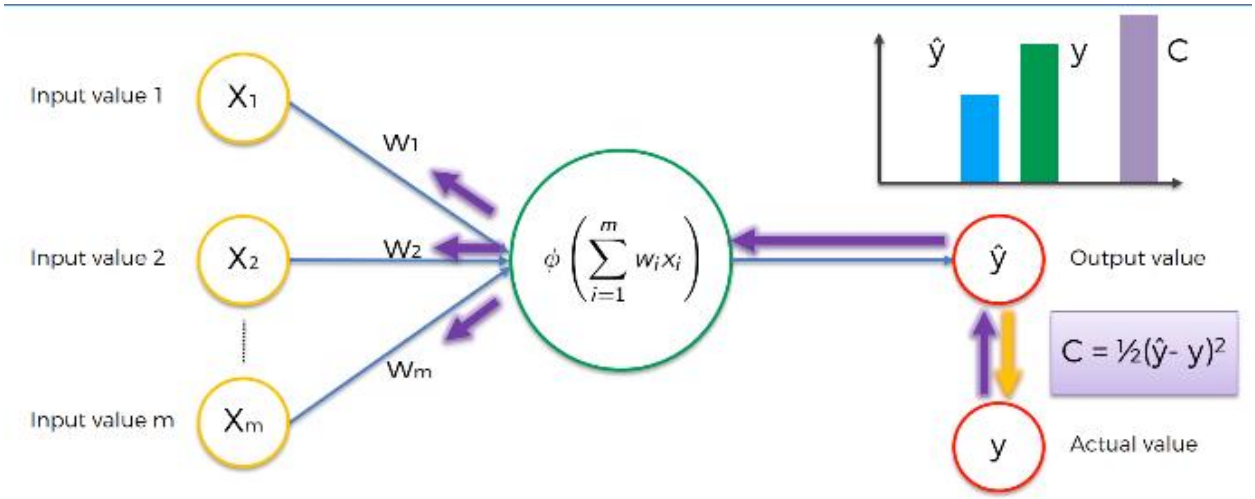


You can check out any Medium article to go into the details of each function. They explained it very well.

How do Neural Network Learns



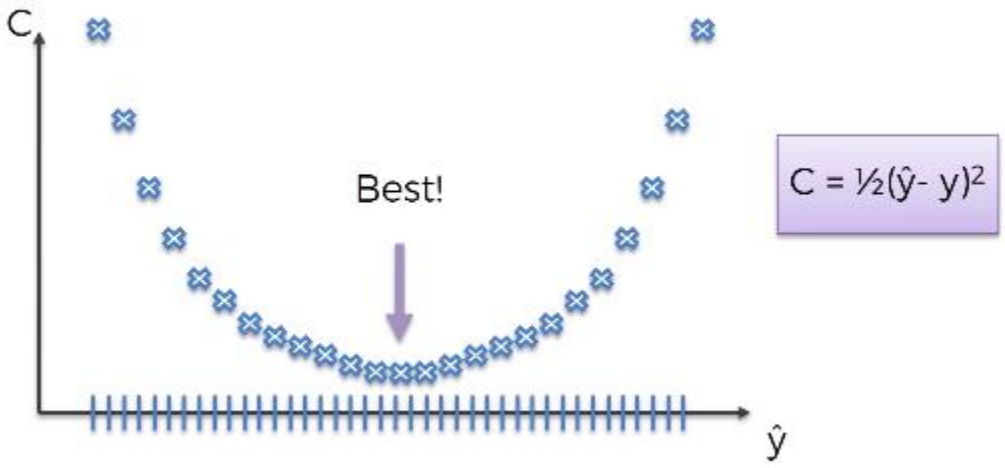
Here is our neural network with a single layer also called feed-forward network or perceptron. We have input values (X1, X2, X3...) and the weights associated (W1, W2, W3...). **y-hat** is the predicted output by this neural network and **y** is the actual value.
So our neuron with these inputs and associated weights predict the output value **y-hat**. Now the cost function is calculated, cost function calculates the difference in predicted value and the actual value ie difference in **y-hat** and **y**



So with this information on the cost function, we are going to feed cost function value back to the network and according to it, it will updates weights (W1, W2, W3 ...). All we have is the weights to tweak in a neural network and the network is going to update the weights and calculate y-hat. Then again we will compare the newly calculated predicted value **y-hat** with actual value **y**, again calculate the cost function backpropagate it to adjust the weights, this process will go on till it gets the minimum cost function.

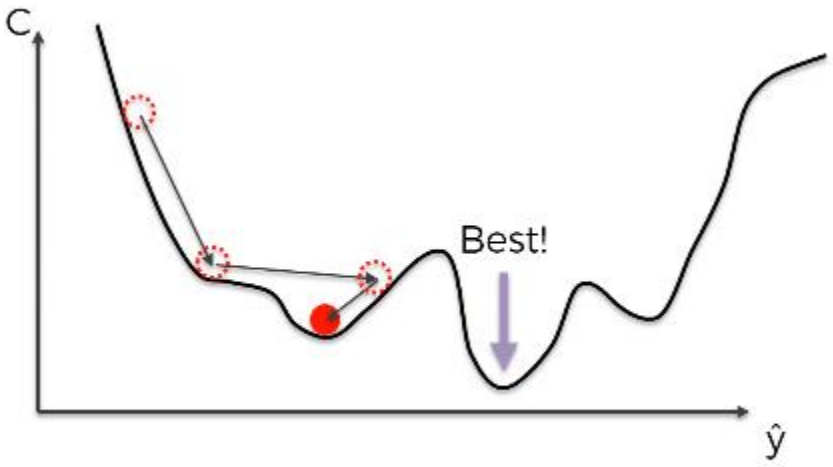
Gradient Descent

So till now, we know that in order for a Neural Network to learn it needs backpropagation. We backpropagate the value of Cost function ie. the sum of the squared difference of the y-hat and y is calculated and sent back to the network and weights are adjusted accordingly.
So how these weights are adjusted. To understand this let's plot the values of cost function on a graph we will get a structure like this.



We want to achieve the minimum cost function value. The gradient descent algorithm helps us to achieve that minimum value. Gradient descent algorithm is an iterative process that takes us to the minimum of a function.
I am not going into the details math behind the algorithm that how it achieves the optimal value. Just remember that a Gradient descent algorithm helps us achieve the global minimum value.
Point to be noted that in Gradient Descent or Batch Gradient Descent we calculate the cost function of the whole dataset and then update then backpropagate its value.

Let us consider the other case where the cost function graph looks like this:



In this case, if we use gradient descent algorithm we won't be able to find the global minimum, instead, we will get some local minimum cost function value, so the solution here is **Stochastic Gradient Descent**. In Stochastic Gradient Descent, the algorithm won't take the whole dataset to calculate cost function instead it will calculate the cost function for each row of the dataset and then backpropagate it to adjust weights.

Go through this stat quest youtube video to understand in detail about the gradient descent: https://www.youtube.com/watch?v=sDv4f4s2SB8

Summary

Finally, let's go through step by step on how we are going to model our Artificial Neural Network.

Step1: Randomly initialize the weights to small numbers close to 0(bit not 0).

Step2: Input the first observation of your dataset in the input layer, each feature in one input node.

Step 3: Forward – Propagation: from left to right the neuron are activated in a way that the impact of each neuron's activation is limited by the weights. Propagate the activations until getting the predicted result y.

Step4: Compare the predicted result to the actual result. Measure the generated error.

Step5: Back Propagation from right to left, the error is back propagated. Update the weights according to how much they are responsible for the error. The learning rate decides by how much we update the weights.

Step6: Repeat Steps 1 to 5 and update the weights after each observation (Reinforcement Learning) Or:
Repeat Steps 1 to 5 but update the weights only after a batch of observations (Batch Learning).

Step7: When the whole training set passed through the ANN that makes an epoch. Redo more epochs