Welcome to Part 8 - Deep Learning!

Deep Learning is the most exciting and powerful branch of Machine Learning. Deep Learning models can be used for a variety of complex tasks:

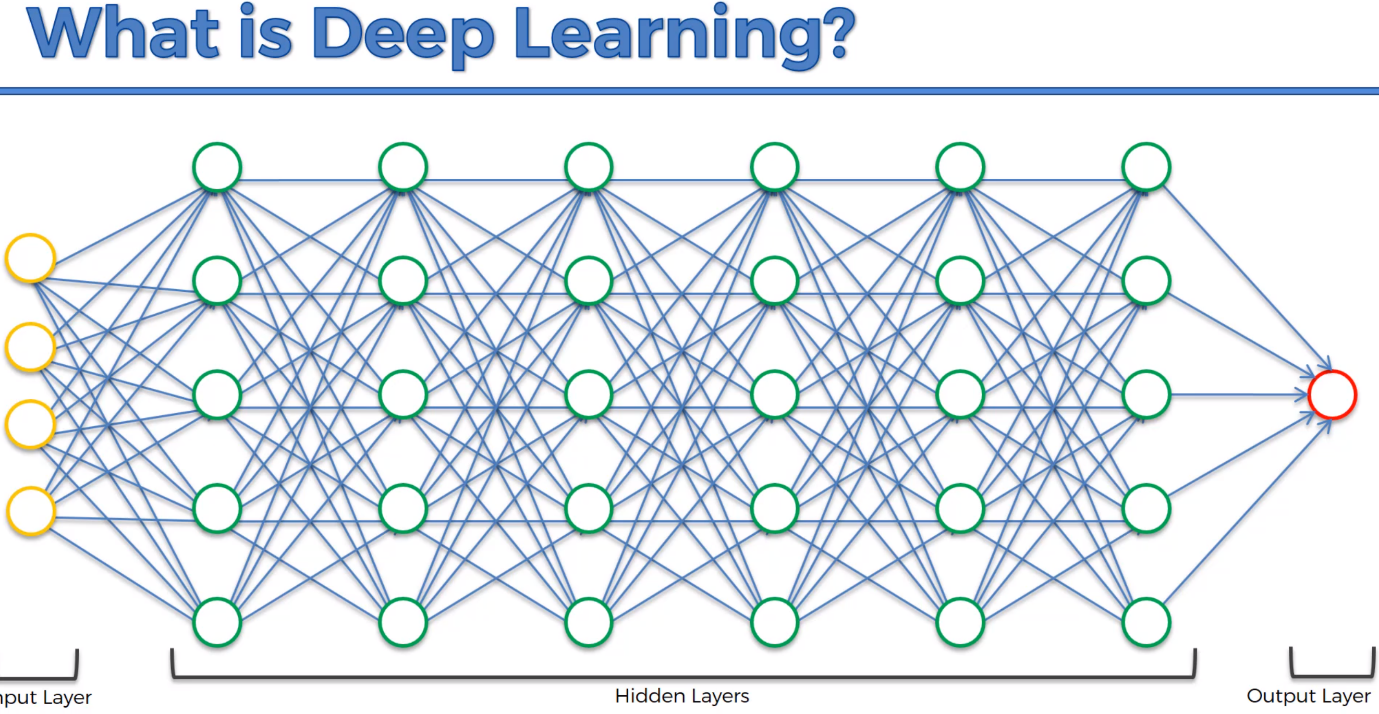
* Artificial Neural Networks for Regression and Classification
* Convolutional Neural Networks for Computer Vision
* Recurrent Neural Networks for Time Series Analysis
* Self Organizing Maps for Feature Extraction
* Deep Boltzmann Machines for Recommendation Systems
* Auto Encoders for Recommendation Systems

In this part, you will understand and learn how to implement the following Deep Learning models:

1. Artificial Neural Networks for a Business Problem
2. Convolutional Neural Networks for a Computer Vision task

Deep Learning –

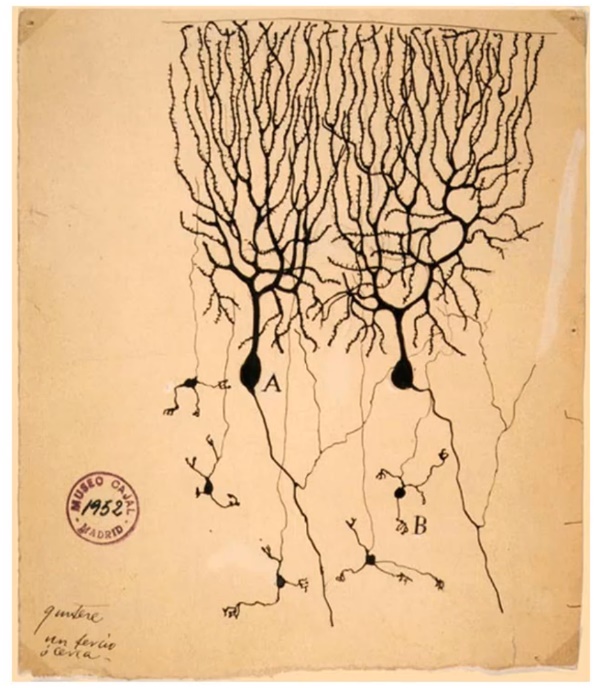
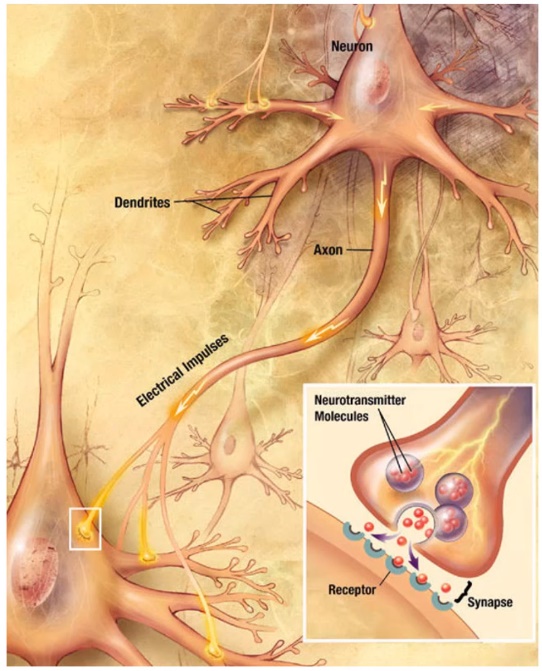
Storage



Artificial Neural Network

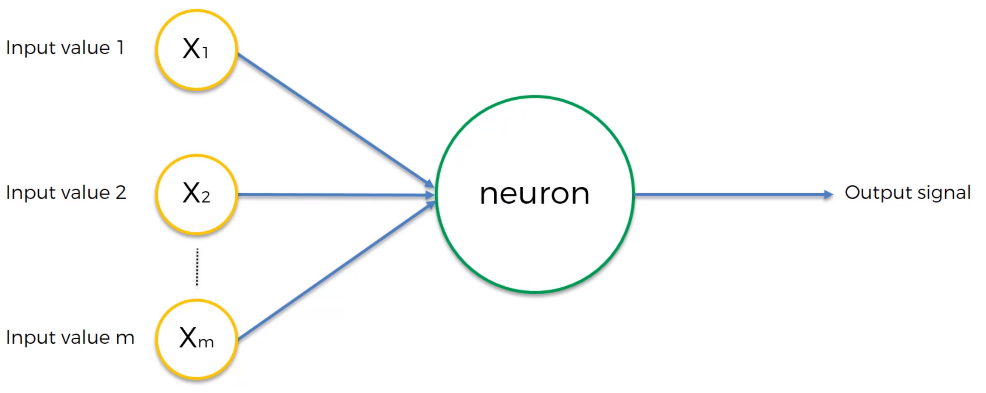
* + The Neuron
  + The Activation Function
  + How do Neural Networks Work? (example)
  + How do Neural Networks Learn?
  + Gradient Descent (Instead of Brute Force)
  + Stochastic Gradient Descent (better method)
  + Backpropagation

The Neuron

How do I recreate a Neuron?

Yellow means INPUT Values LAYER, OUTER LAYER

Green means Hidden Layer

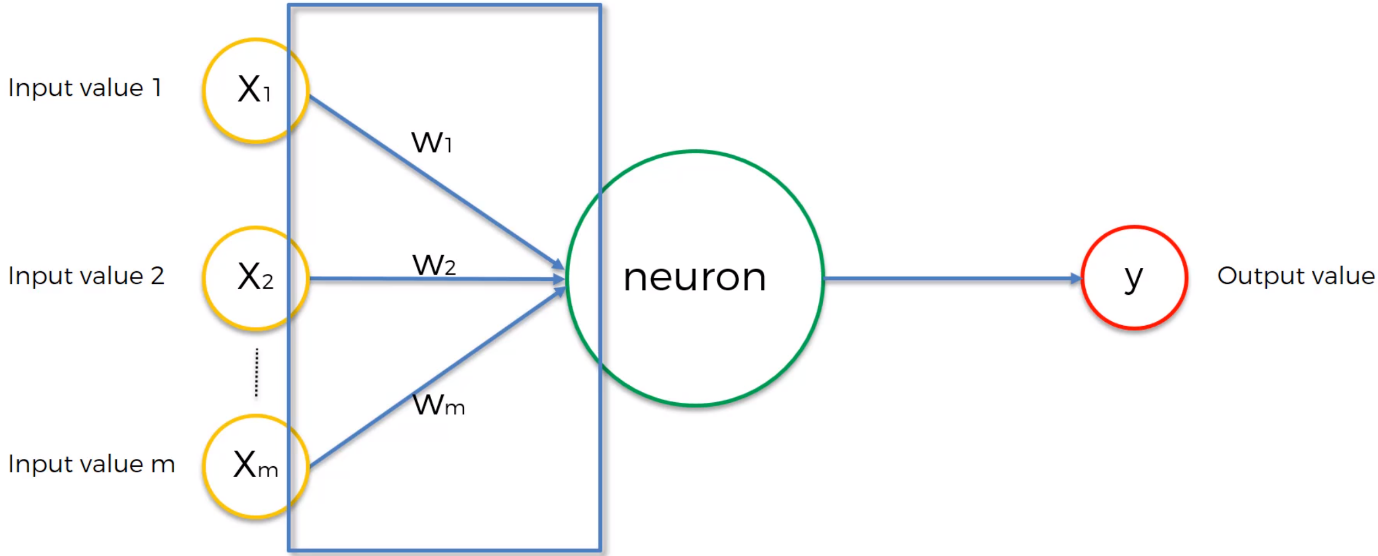
Red is the Output Layer.

Input values are independent variables, One row, one observation. Additional Resources. Efficient BackProp by Yann LeCun. Mean Cacellation -> KL-Expansion -> Covariance Equalization.

<http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf>

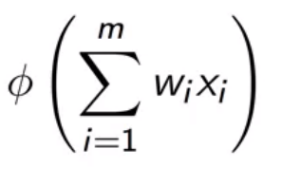
Output Value can be Continuous (Price) or Binary (Yes or no) or Categorical.

Note: Single Observation 🡨----------Same Observation-------🡪 Single Observation

Weights are crucial, they are how neural network learn, by adjusting the weights we can determine what signal is important and which are not. They will be adjusted when the machine is learning.

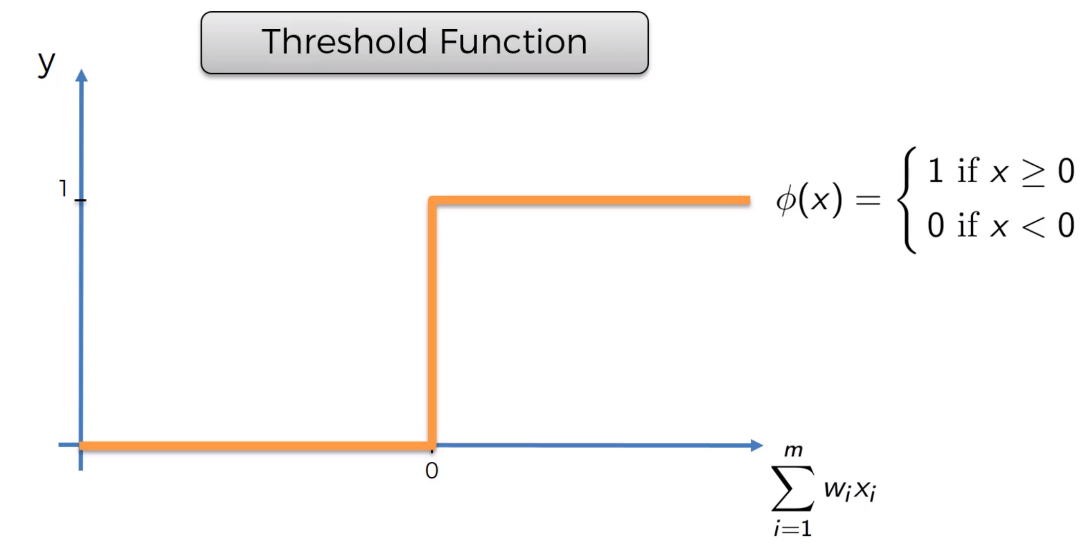
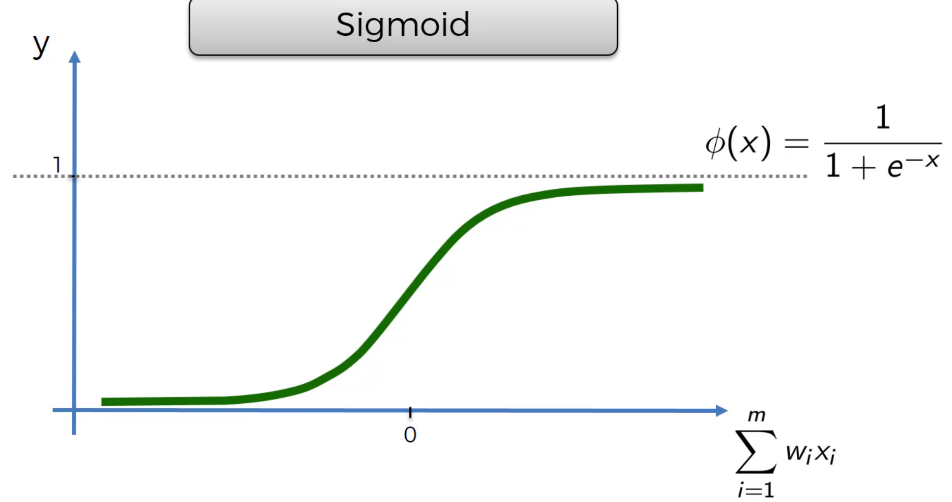
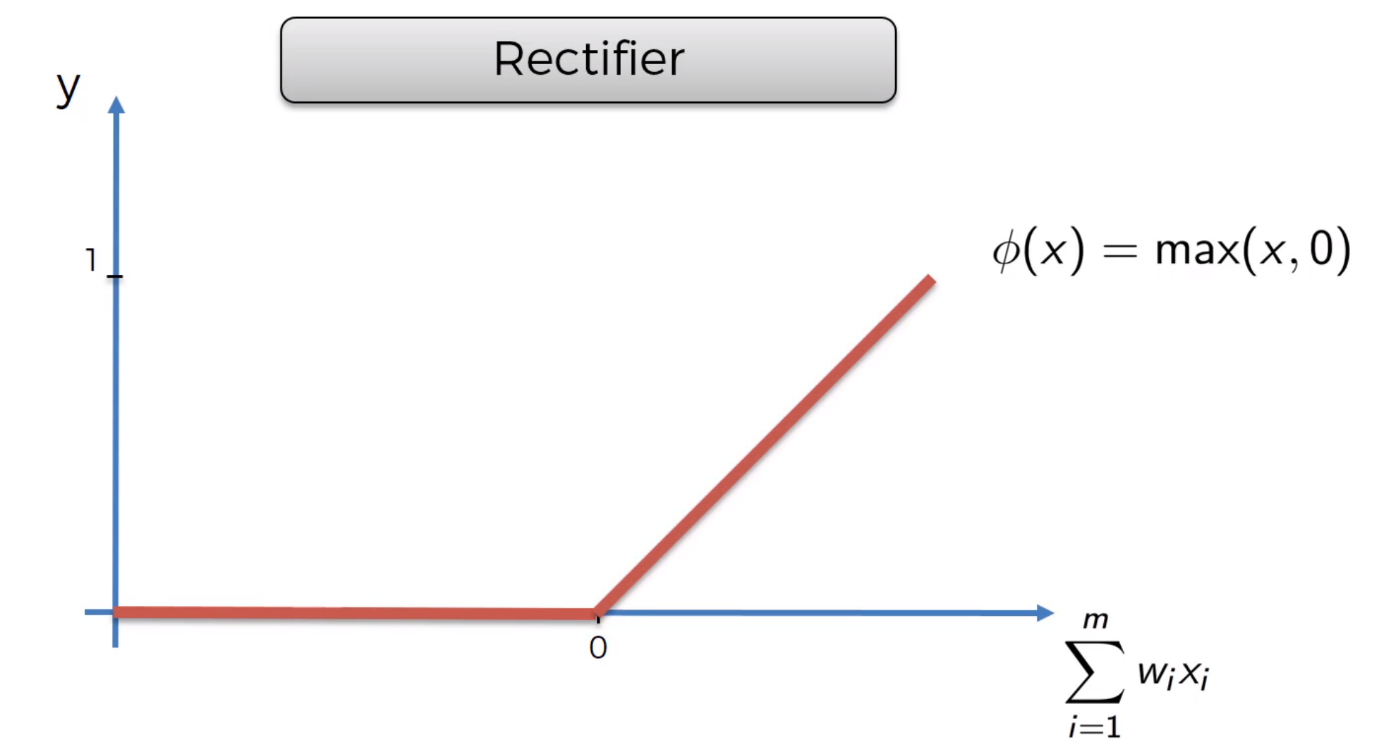
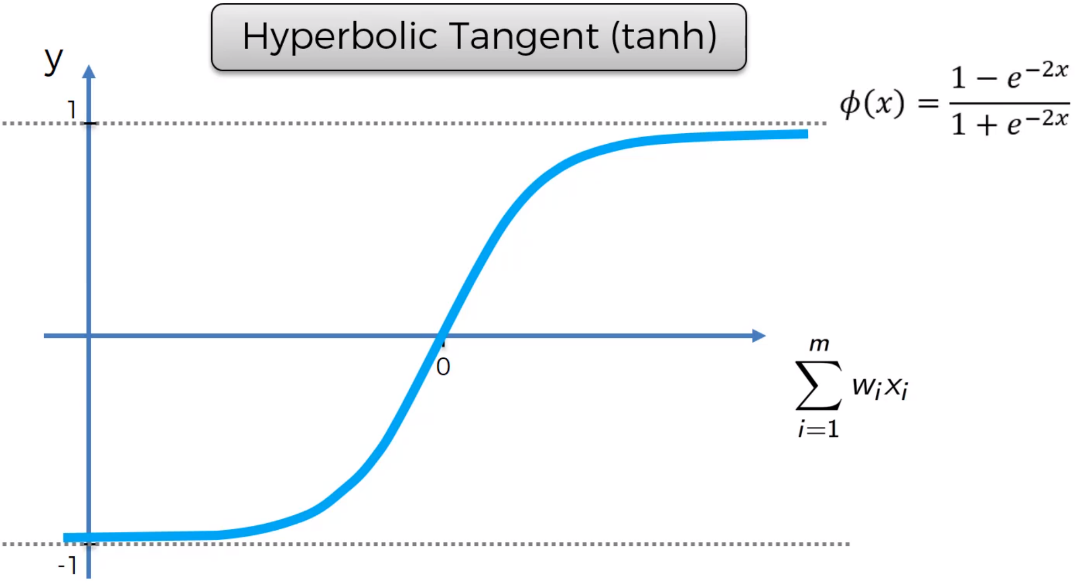
**Neuron? What happens in the Neuron?**

1. All of these values gets added up. Takes the weighted sum of all the input values that it is getting
2. Applies an activation function. Function that is assigned to the neuron and it is applied to the weighted sum. The neuron will either pass or won’t pass
3. Send the signal down the line towards output value.



Activation Function

4 types of Activation Function:

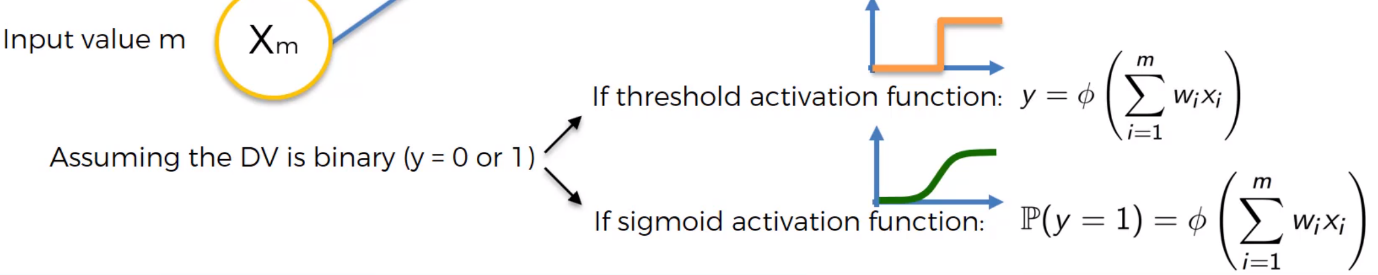
1. Threshold (Yes/No Fn)
2. Sigmoid Function (used in the Logistic Regression) it is smoother. Than the Threshold Function. It is useful in the Final Layer/Output Layer where you are trying to predict the probability
3. Rectifier Function – is one of the most used functions
4. Hyperbolic Tengent (tanh) Similar to Sigmoid Function
5. Additional Link:

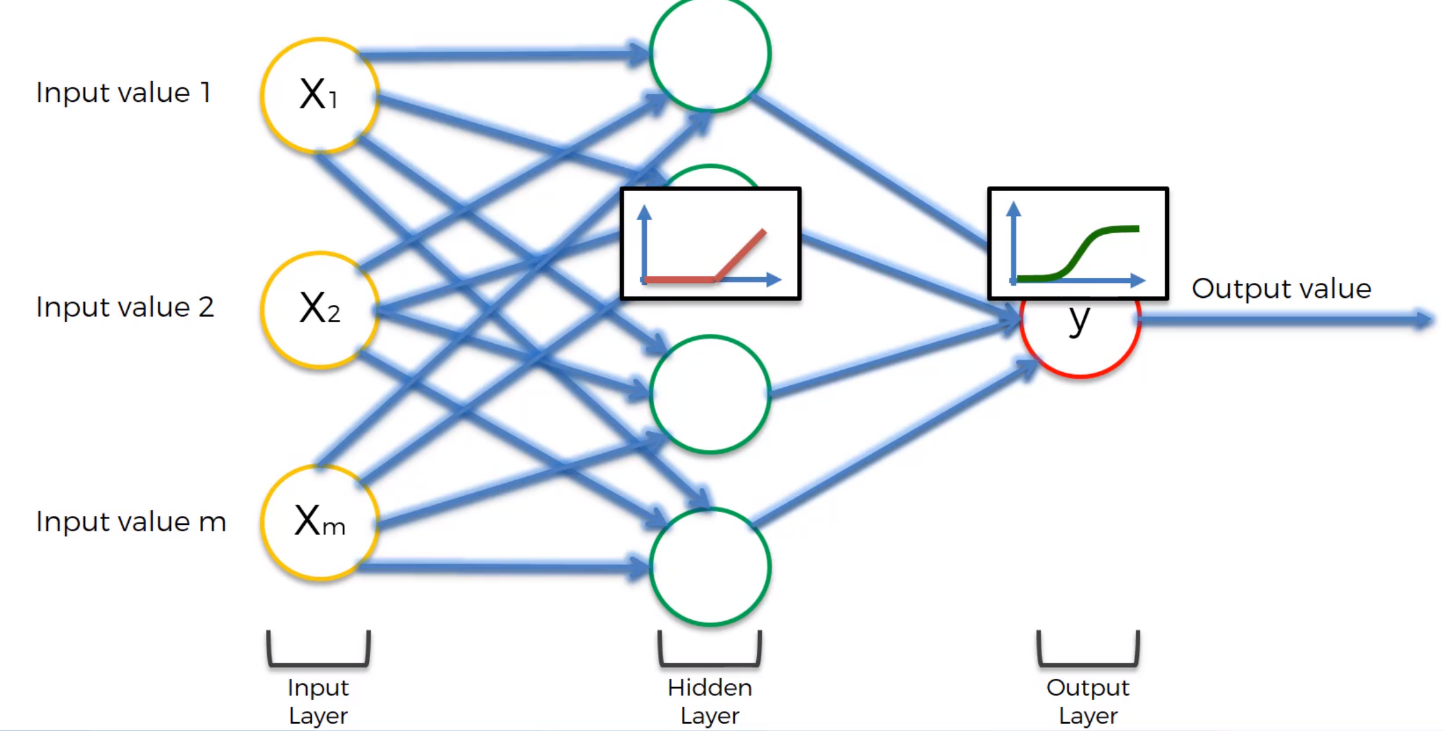
<http://jmlr.org/proceedings/papers/v15/glorot11a/glorot11a.pdf>

Assuming Dependent Variable is Binary (y=0 or 1) -> Which activation function will you use?

Two Options:

1. Threshold Activation Function because its between 0 and 1.
2. Sigmoid Activation Function: The probability of y being yes or no.

Hidden Layer – Rectifier Function

Output Layer – Sigmoid Function

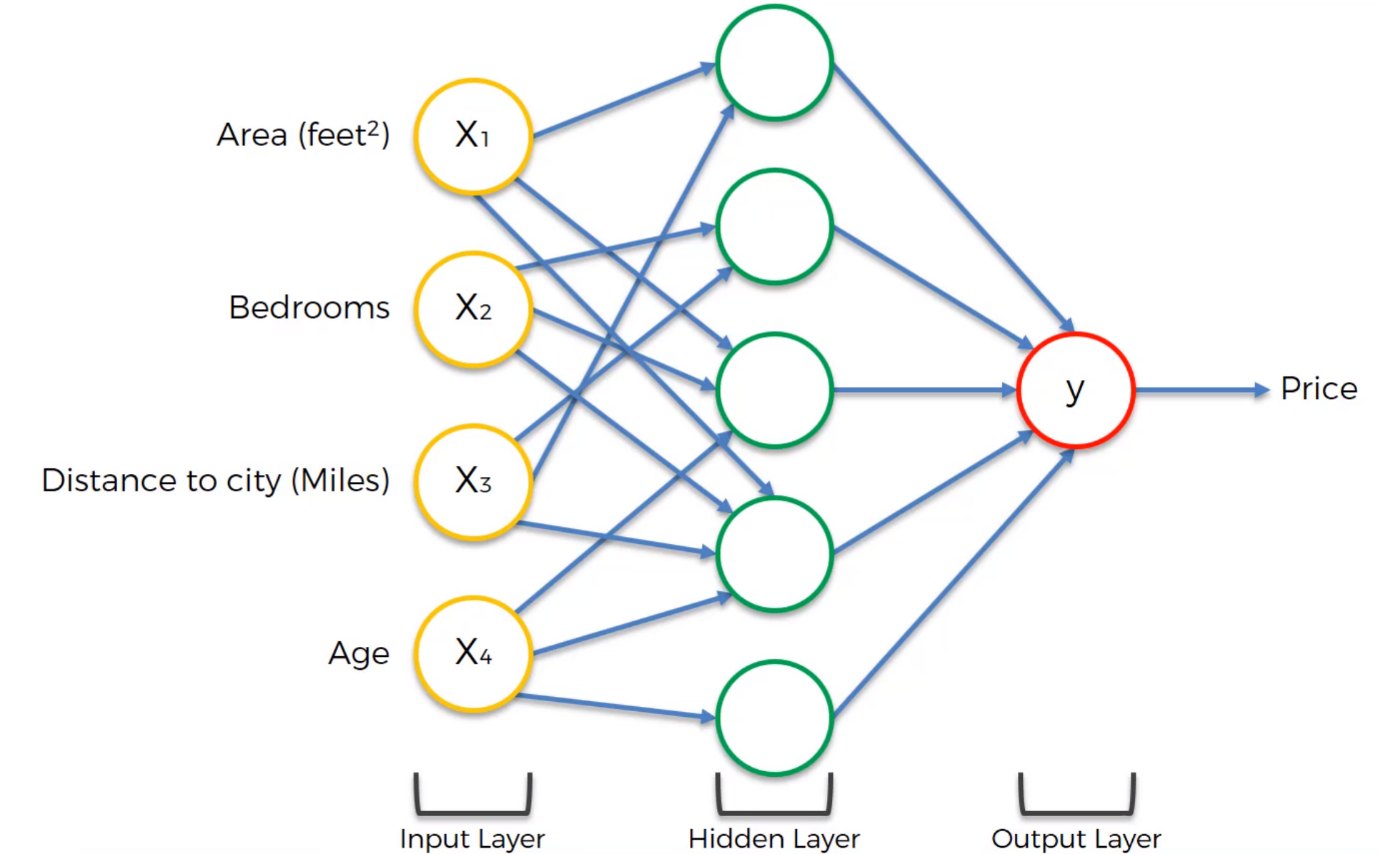
**How do Neural network Work?** Assuming that the Neural Network is Trained.

Input layer- Area, Bedrooms, Distance to City, Age

Hidden Layer -> How doe Hidden layer give us additional Power? Every Neuron takes all the Input Values, some weights will have non-zero value or Some weights values will have zero values. A certain neuron only takes in Area, Distance to miles, that neuron may have a relationship between the Area and Distance to Miles and Price.

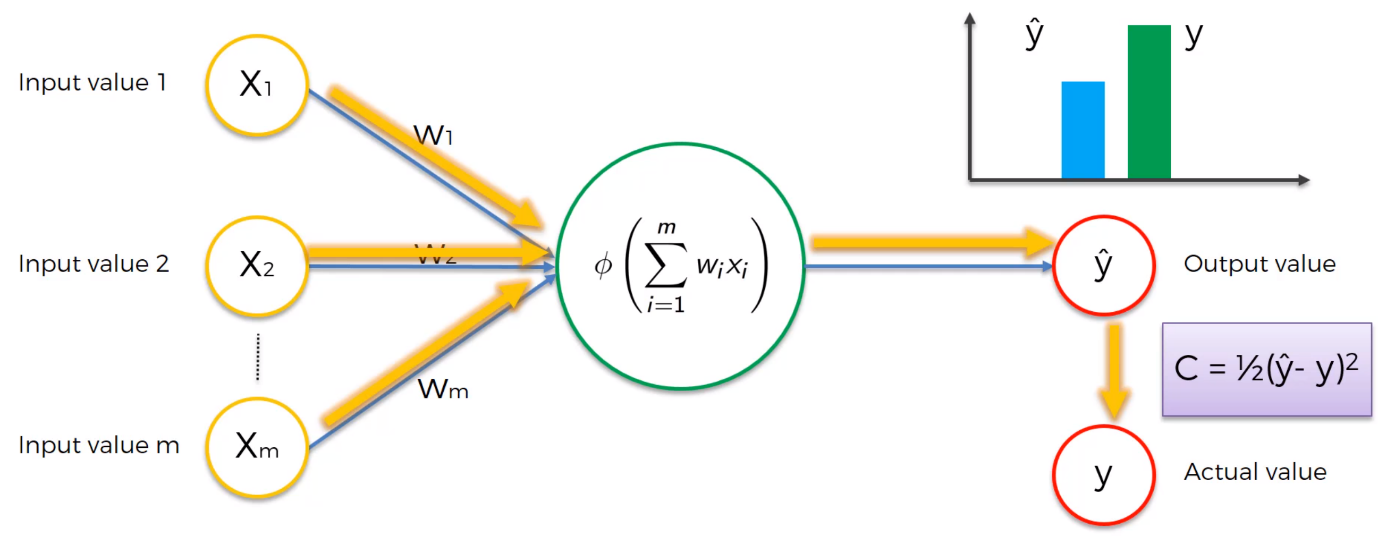
Need to understand what a particular Neuron is thinking?

Output Layer -> Price that we are predicting by weighting with the synapses.

**How do Neural Network Learn?**

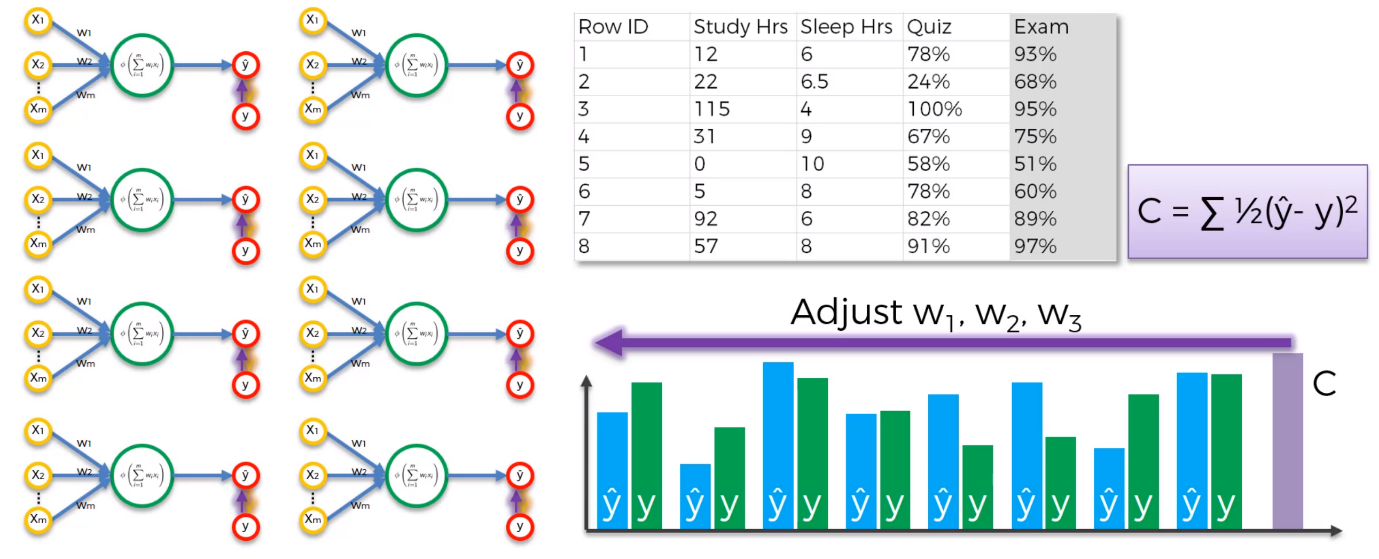
1. Hard Coded Coding
2. Create a facility for the program’s ability to understand the inputs and output and let it figure it out on its own. Give the folder of Cats and Dogs, go and learn what a Cat is and go and understand what a Dog is. If there is a new image of cat or dogs, classify them as cats or Dogs.

Y^ is the output Value and Y is the actual value. Perceptron.

Cost function = ½ (y^ - y)2 It tells us the error in the predictions. The lower the cost function the closer the predicted to the actual results.

The cost function will be feeded into the neural network and goes to the weights and the weights gets updated and tweaked them a little bit. It will be under a recursive function that feeds the cost function and our predicted value keep changing until the predicted is equal to the actual result.

Multiple Rows example.

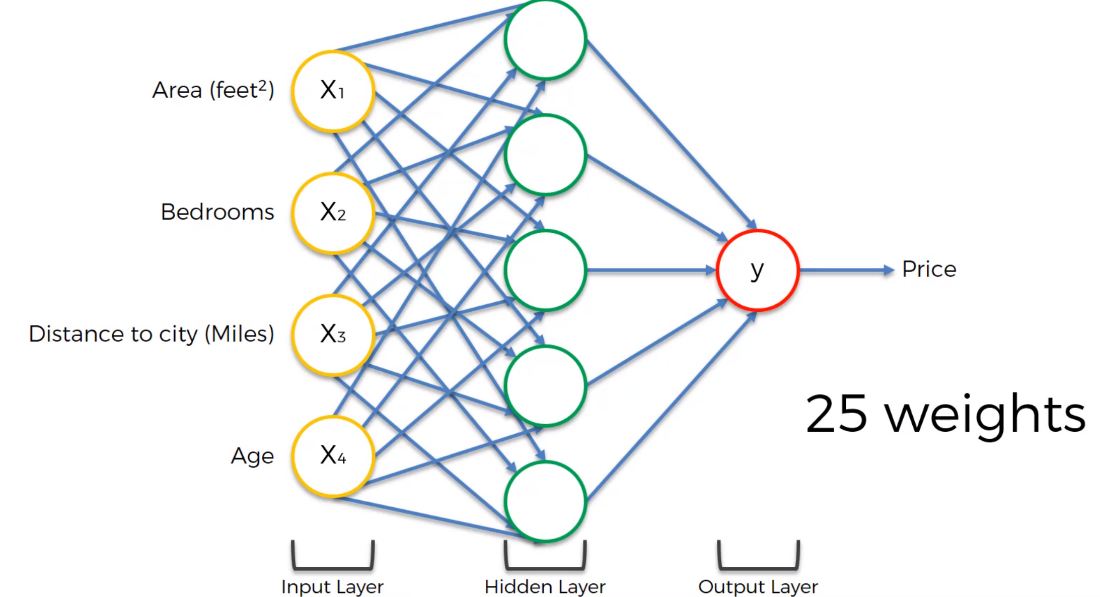
The main goal here is to minimize the Cost Function. And Found the optimal weights for the dataset. The whole process is called Backpropagation

Additional Reading CrossValidated – a list of cost functions used in neural networks, alongside applications – Google.

**Gradient Descent**

**How can we minimise the cost function?** One way will be a brute force approach. Do a lot of test and trials. And because it’s a x2 curve then take the minimum Cost function value?

Curse of Dimensionality – look at Practical Example.

Calculate the number of weights lets say 4 input variables in input layer, 5 neurons in the Hidden Layer

Total Number of weights (Blue Arrows) -> 4x5 at the start and + 5 more from the hidden layer to output layers

25 weights if we have 1000 combination, then its 1075 combinations..

Fastest computer does 93 PFLOPS (peta flops) flops – floating operation per second

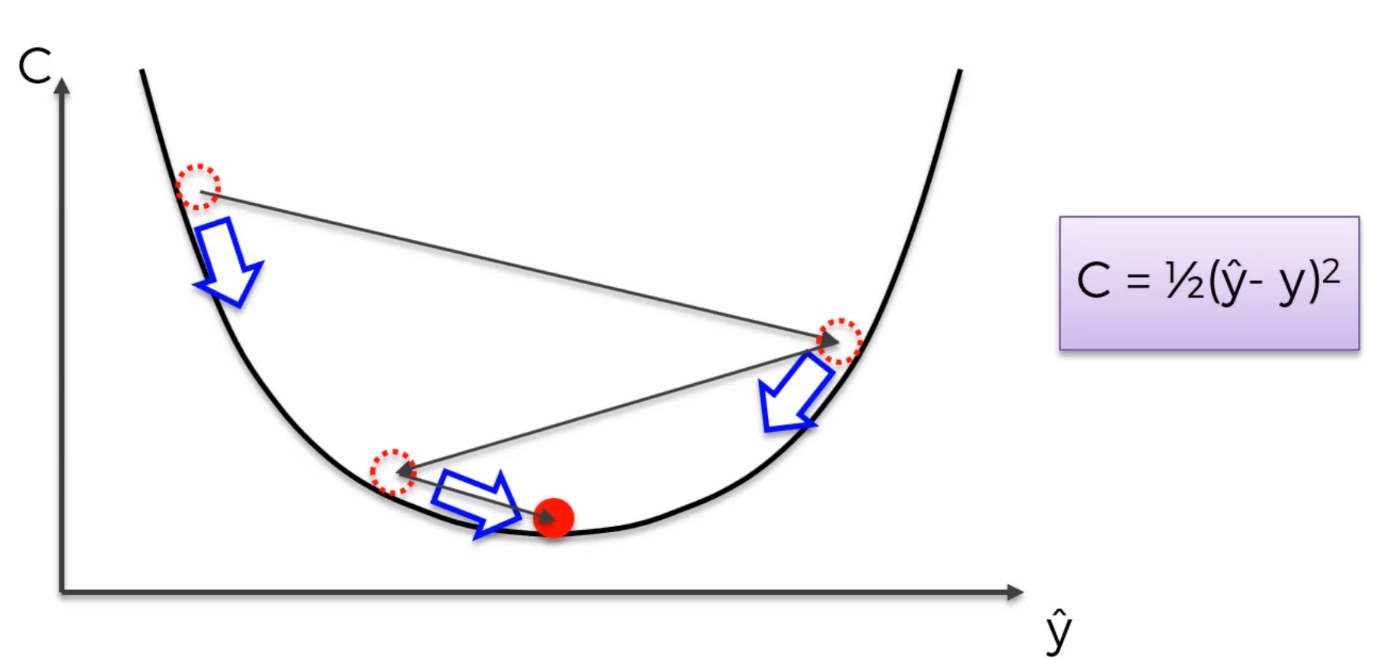
93x1015 per second.

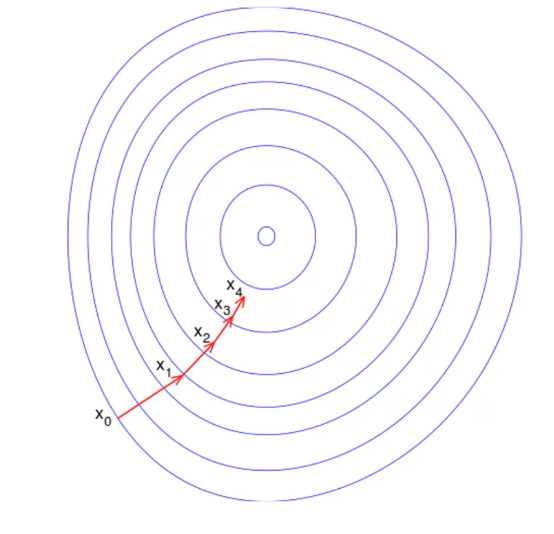
1075/(93x1015) which is 1.08 x 1058 = 3.42x1050 years

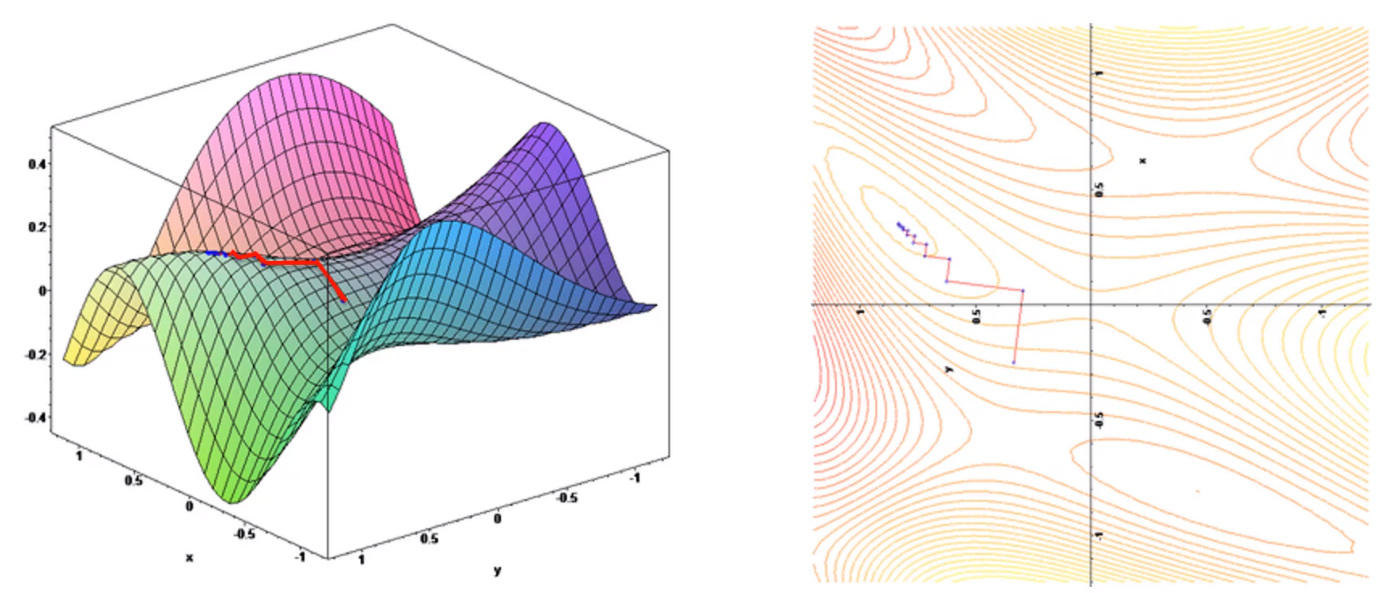
It cant support the brute force method. It will take Decades to go through the Brute Force in a Super Computer.

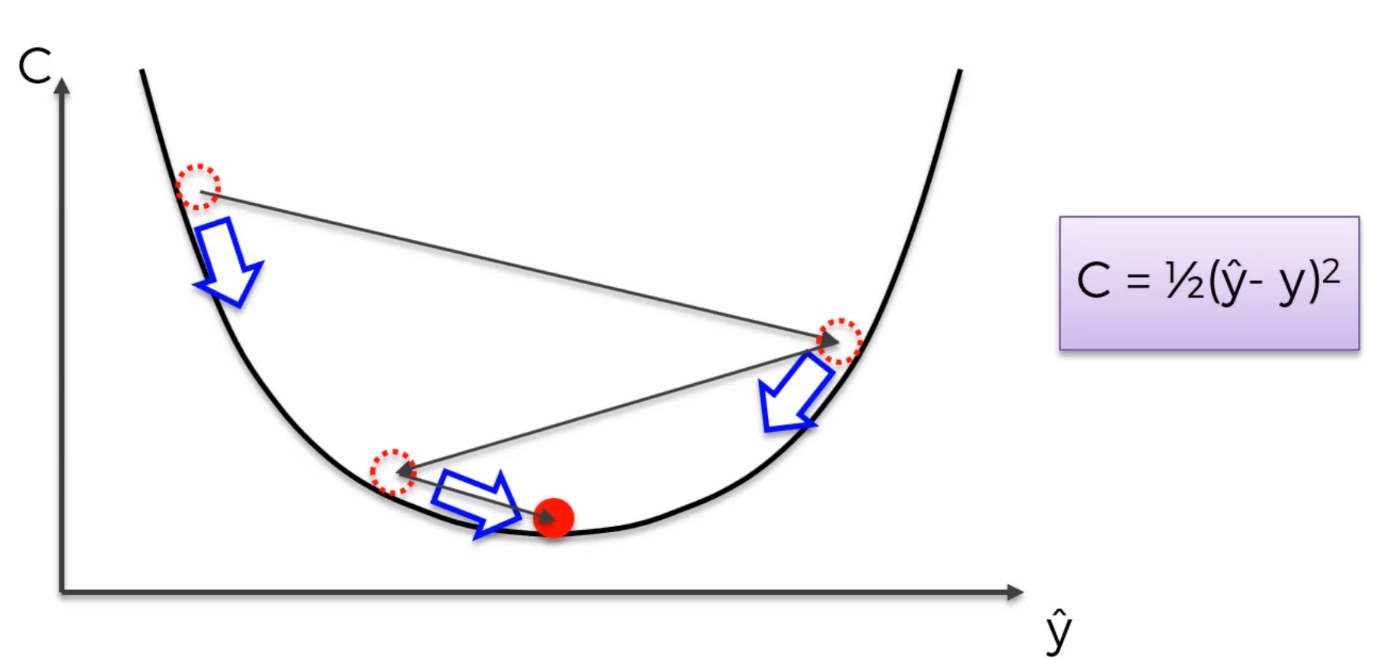
Gradient Descent -> Take the cost function and Find a point and calculate the gradient by differentiate and find out the slop is positive or negative, see if you are going downhill or uphill,

It is just drop a ball in a X2 curve and find the best weight.



Gradient Descent in 2D.

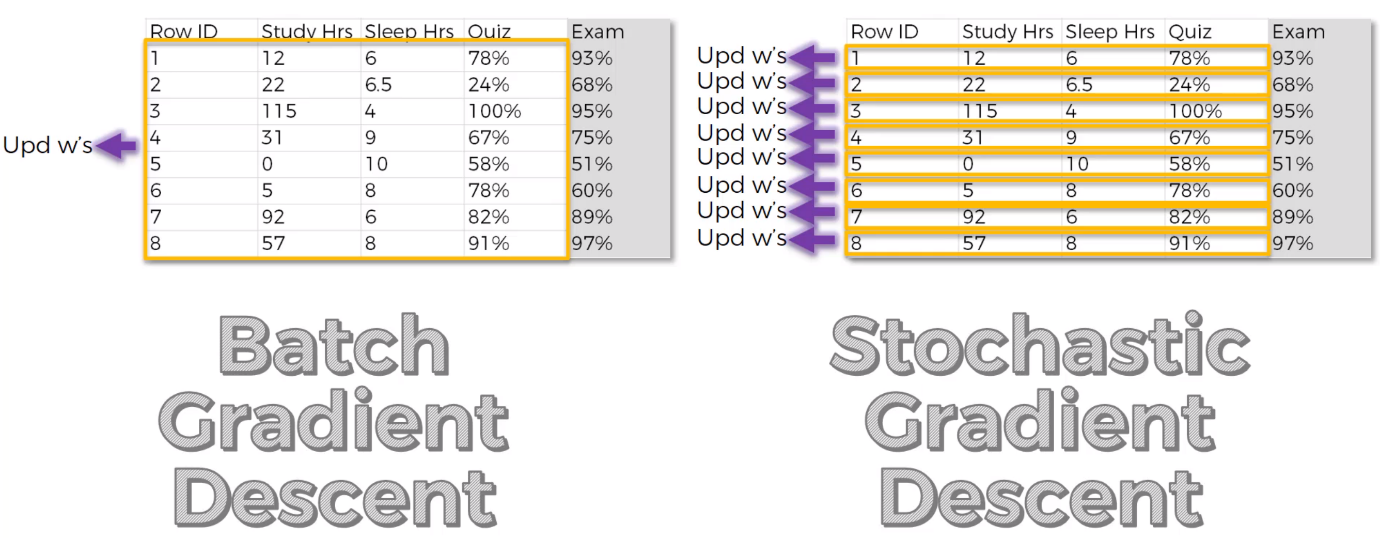
**Stochastic Gradient Descent**

Minimise the Cost function. The method requires **for the Cost function to be COVE****X.**

If we choose a cost function that is not like the above, then what will it happen? Then we apply the normal gradient descent and find the local minimum instead of global minimum.

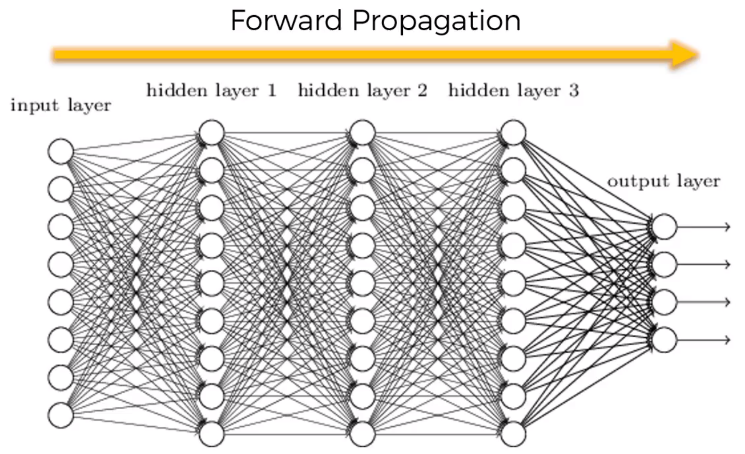
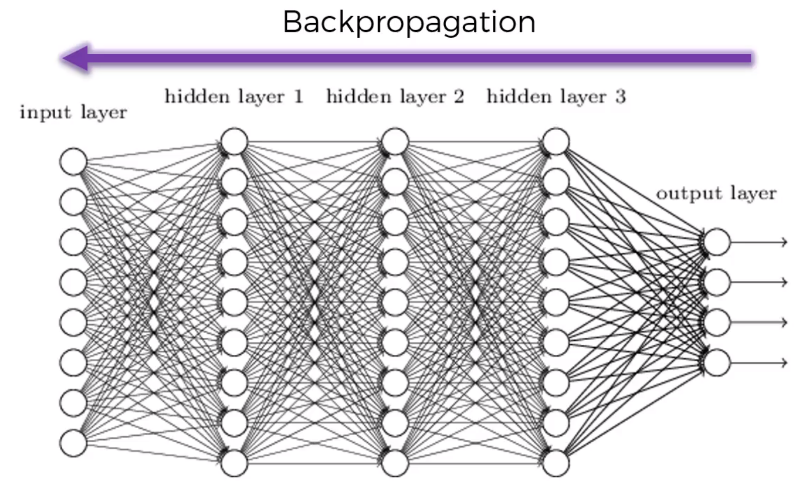
Normal Gradient Descent -> Batch Gradient Descent.

The answer here is Stochastic Gradient Descent. Which adjust the weights after every row.

SGD has much more fluctuations that’s why it is able to find the global minimum instead of local minimum. Also, it is faster than the Batch Gradient Descent. Batch is deterministic, and SGD is random. There is Mini-Batch Gradient Descent Method.

Additional Links – A Neural network in 13 Lines of Python (Part 2-Gradient Descent) Andrew Trask (2015)

Neural Networks and Deep Learning Michael Nielsen (2015)

**Backpropagation**

**What happens in Training of a Neural network?**

**Step 1: Randomly Initialise the weights to a small numbers close to 0 (but not 0).**

**Step 2: 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 neurons are activated in a way that the impact of each neuron’s activation is limited by the weights. Propagate the activations until getting predicted result y.**

**Step 4: Compare the predicted result to the actual result. Measure the generated error.**

**Step 5: 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.**

**Step 6: Repeat step 1 to 5 and update the weights after each observation (Reinforcement Learning). OR: Repeat step 1 to 5 but update the weights only after a batch of observations (Batch Learning).**

**Step 7: When the whole training set passed through the ANN, that makes an epoch. Redo more epochs.**