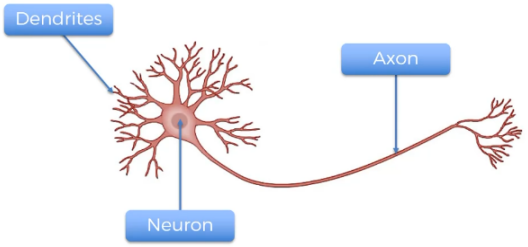
Deep Learning - Neural Networks

This whole notes prepared using super data science course of deep learning videos.

Topics

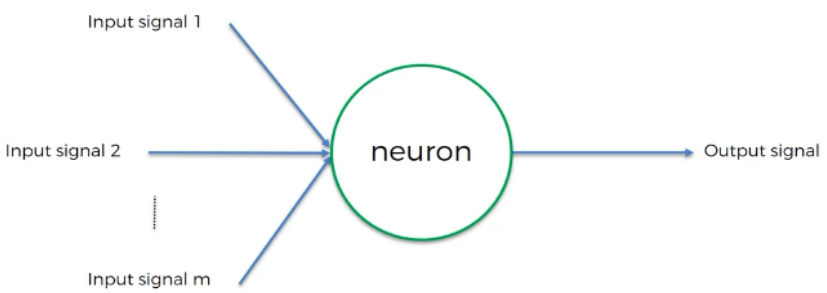
1. Neuron
2. Activation Function
3. How do Neural Networks work?
4. How do Neural Networks Learn?
5. Gradient Descent
6. Stochastic Gradient Descent
7. Back Propagation
8. Optimizers

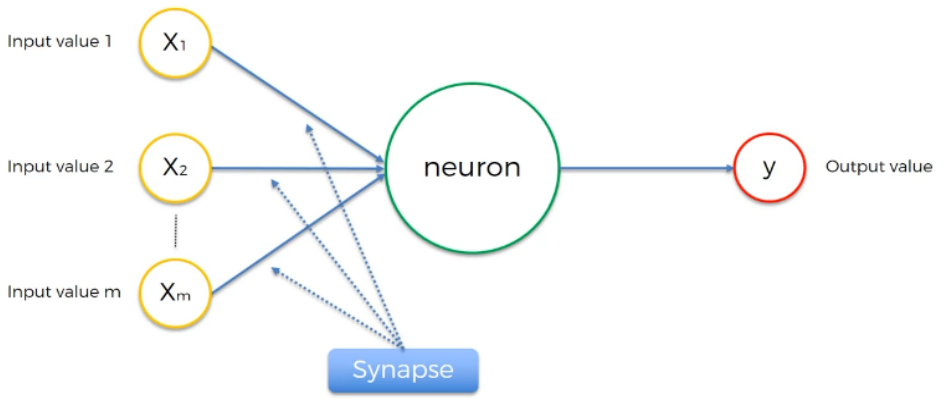
1. **Neuron:** The Neuron is a **basic building block of artificial neural networks**.



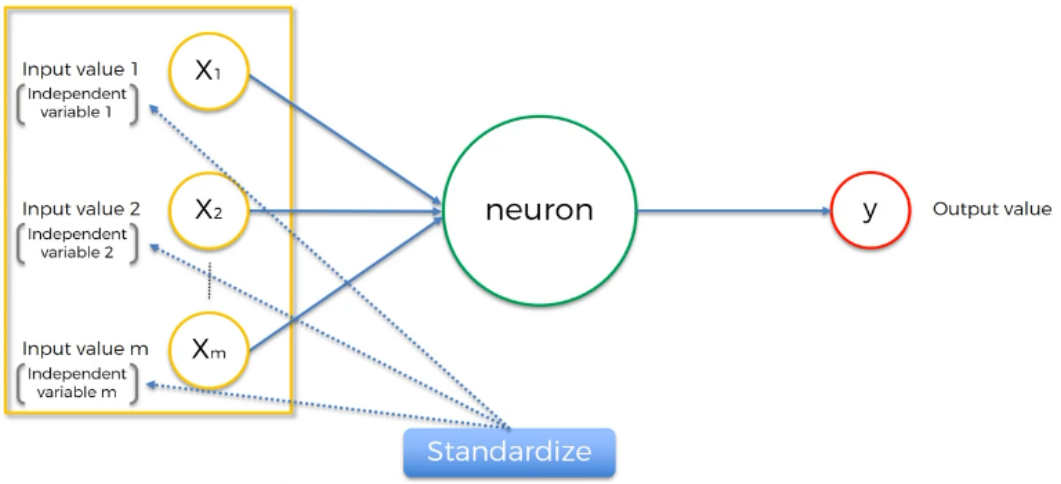
This is an actual neuron structure of human brain.
Dendrites - Receivers of the signal for the neuron
Axon - Transmitter of the signal for the neuron.

Neurons in the deep learning are as follows. The neurons are sometimes called as **Nodes**

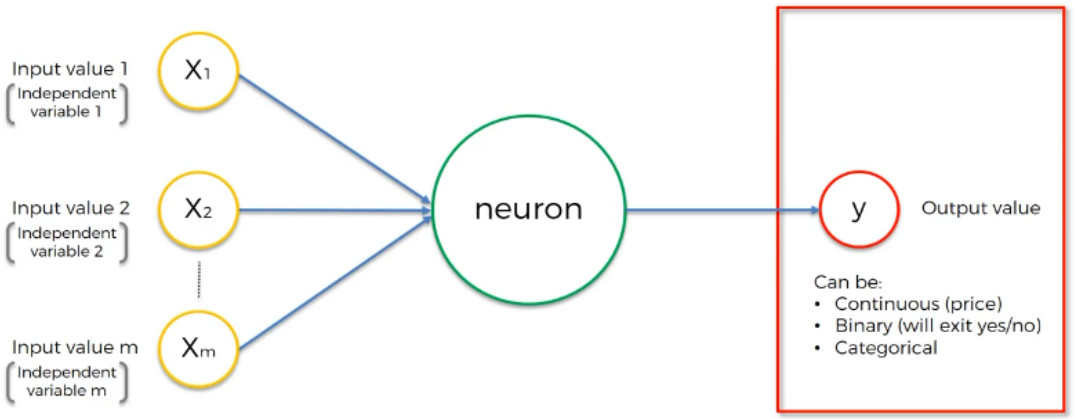




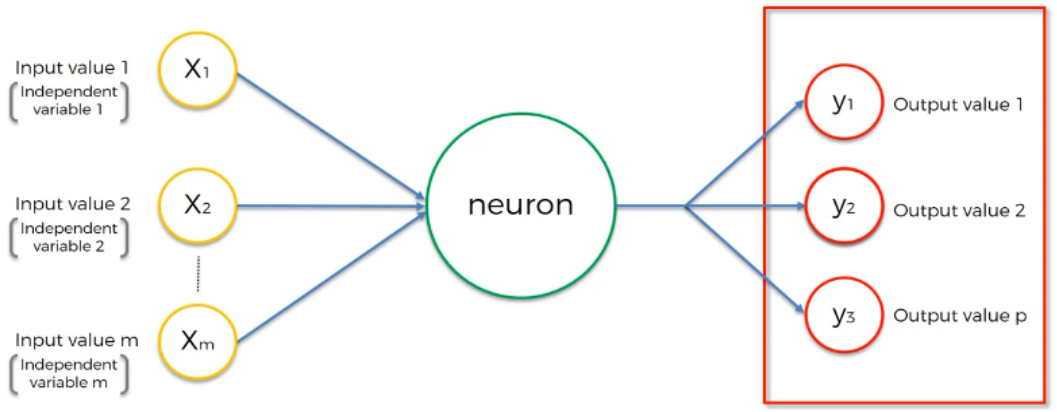
X1, X2, X3 are the input neurons.
The signal is passed through Synapses.

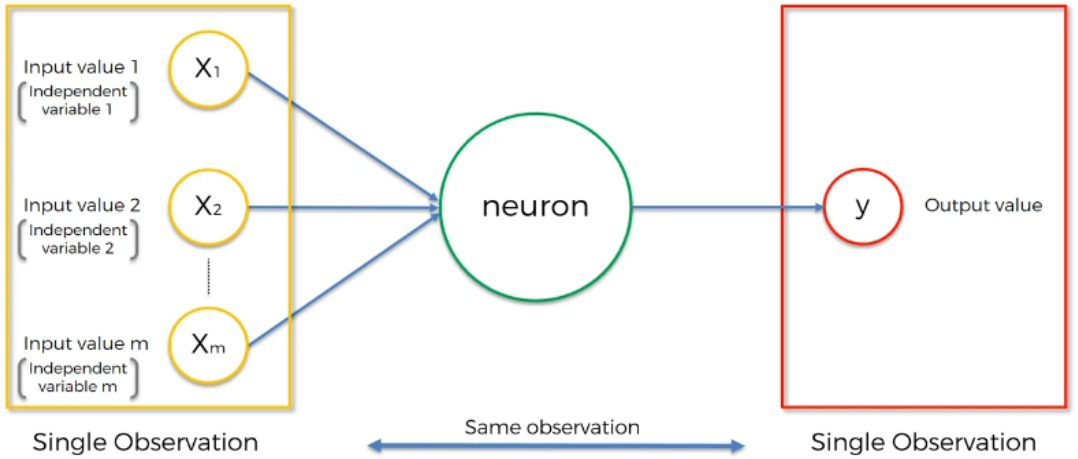


The yellow square box is the input layer. The input signals are nothing but the independent variable values for an observation/row in the dataset. The data must be standardised/normalised.



The red square box is called the output layer. In case of the categorical variable, the output will several values like below.





Remember we are dealing with one row/observation at a time.

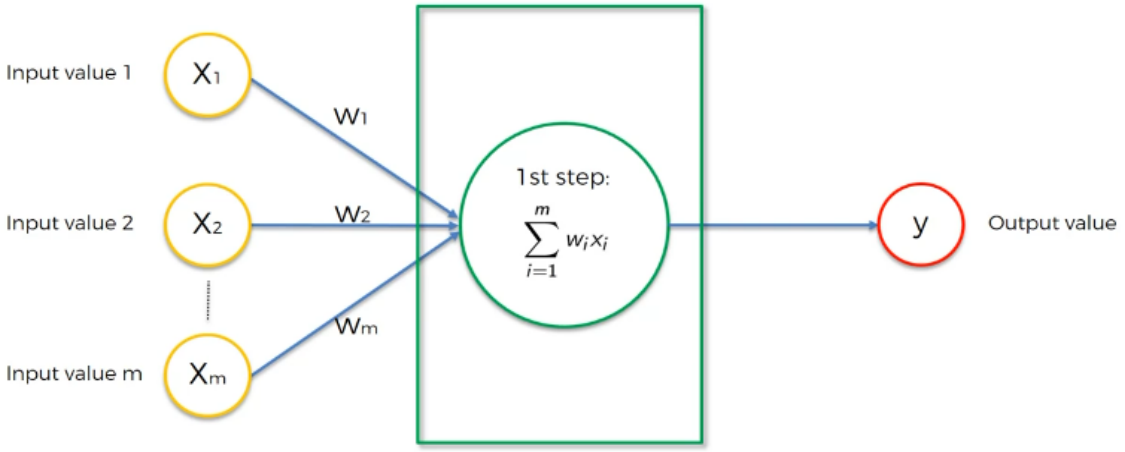
Weights are the way neural networks learn. The weights get adjusted for every single observation by learning how important a feature/signal is.

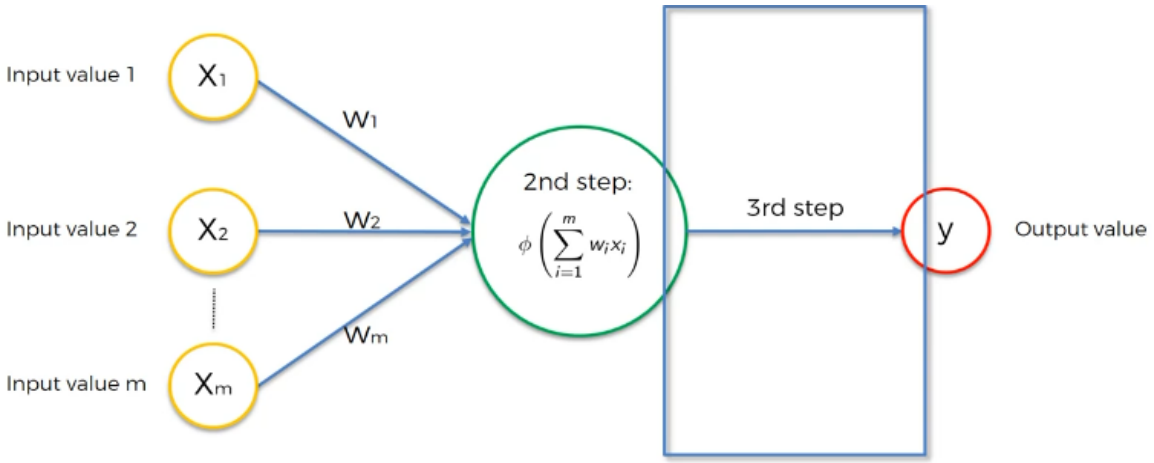
What happens inside the neuron?

Step-1: the weighted sum of the inputs are calculated

Step-2: Applies an activation function.

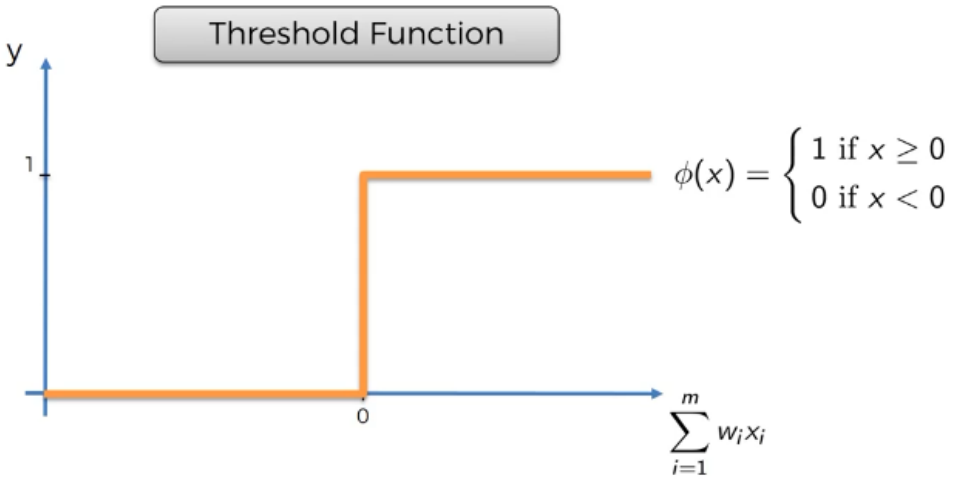
Step-3: The Neuron passes the signal to the next neuron in the next layer.



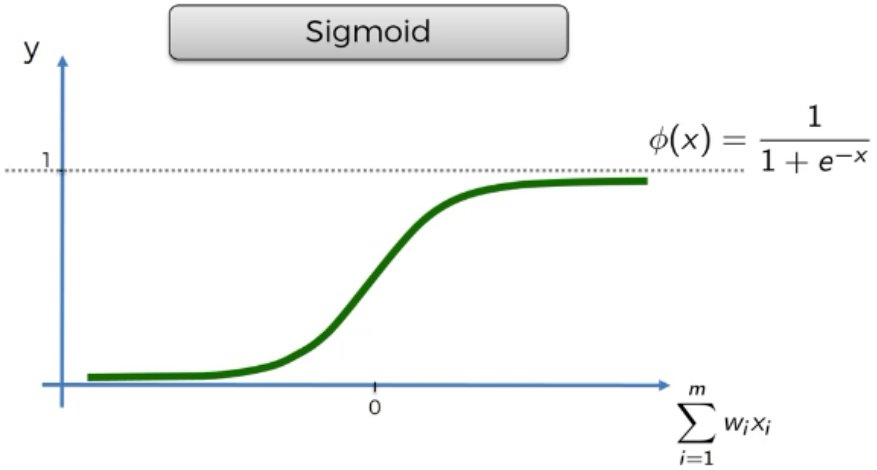


2. **<u>Activation Function</u>** : We have 4 predominant options for the activation function.

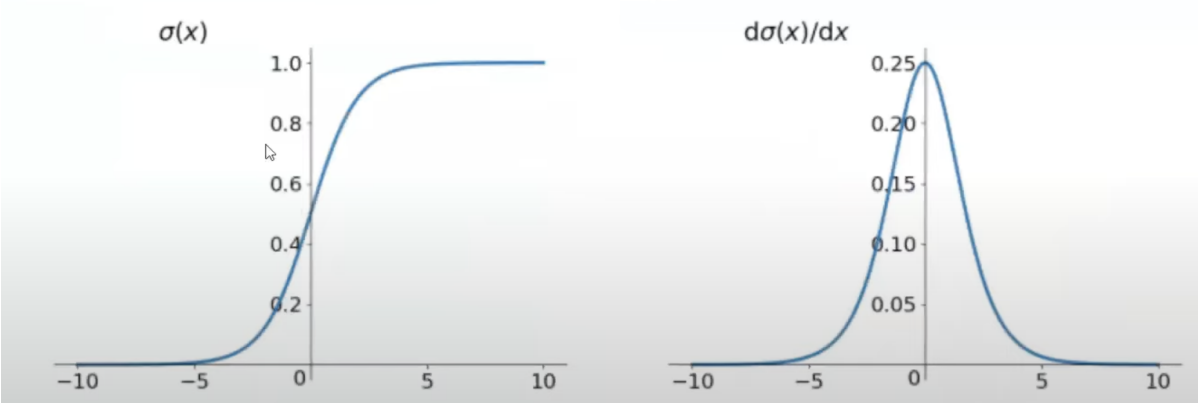
<u>A. Threshold Function:</u> It is basically like a yes/no type of function.
X-axis has the value of the weighted sum of the inputs. If the value is less than 0, then the threshold function passes a 0. if the value is higher than 0 then the threshold function passes on a 1.



<u>B. Sigmoid Function:</u> It is the function used in the Logistic Regression.
'x' is the value of the weighted sum
This function is very useful in the final output layer, especially when we are predicting probabilities.
Anything below 0 will be approximated as 0. anything above zero will be approximated to 1.



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



Disadvantages of Sigmoid activation function

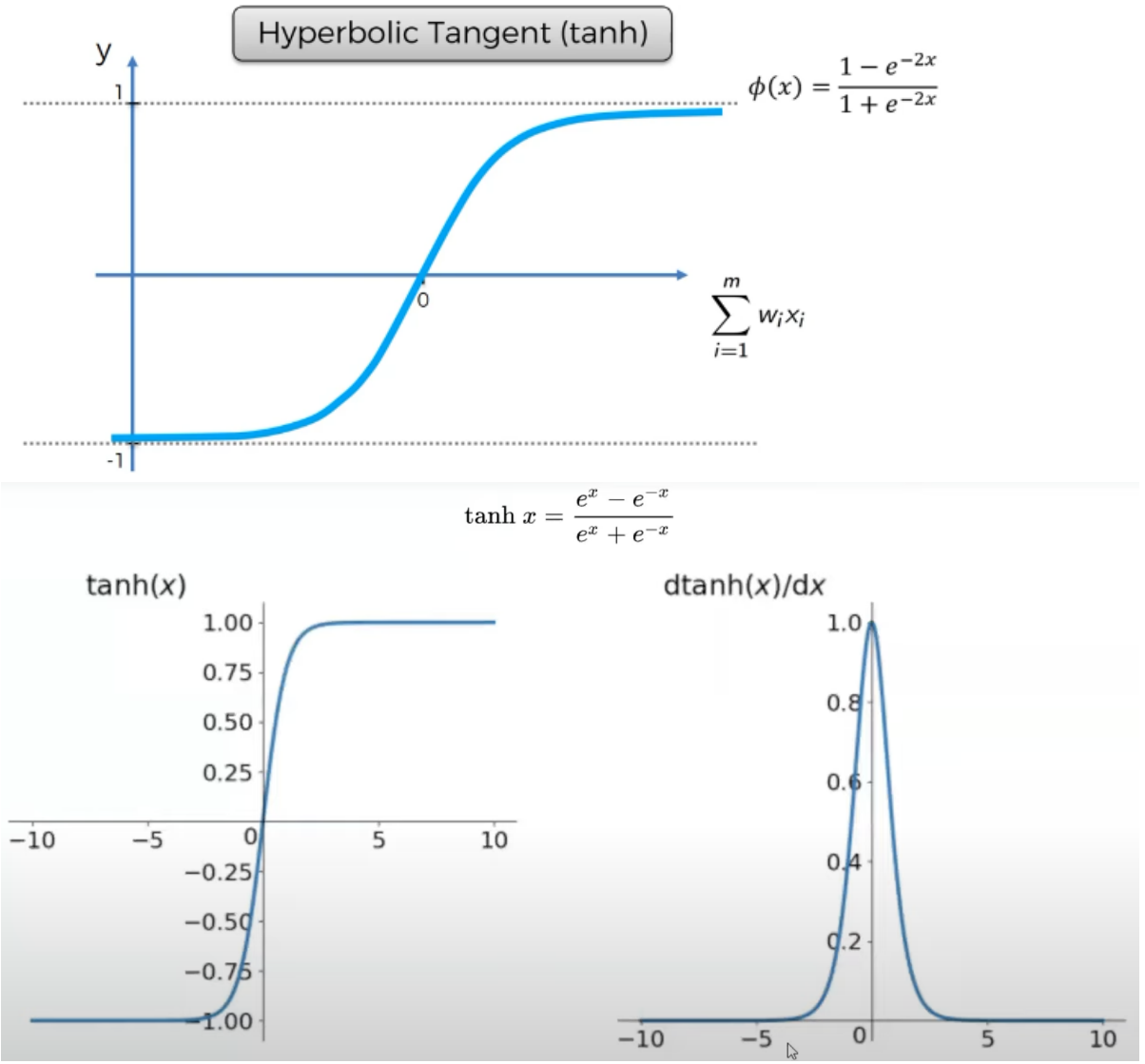
1. **Vanishing Gradient Problem**

Before the invention of the RELU activation, the only option to use as an activation function is 'Sigmoid'. The derivative of a sigmoid ranges only from 0 to 0.25. When the number of layers are increased and the during the gradient calculation the no.of derivatives increases due to the chain rule. So, in that case the derivative values decreases so much for increasing derivatives in the chain rule and the gradient slowly vanishes. This is called Vanishing Gradient Problem.
So, if we use Sigmoid as the activation function in the hidden layers of a neural network, then we encounter the Vanishing Gradient Problem. So, thats why we use RELU in the hidden layers. For clear explanation watch this video. (https://www.youtube.com/watch?v=JIWXbzRXk1I)

Notice the last graph above, the d(f(x))/d(x) (i.e. the derivative of the sigmoid function) always ranges from 0 to 0.25 only. Because of this, the gradient calculations in the case of deep neural networks, the hidden layers gradient will be almost zero, (i.e. the gradient slowly vanishes). This is called the **Vanishing Gradient Problem.** Because of which it is always recommended to avoid sigmoid function in the hidden layers.

2. The Second problem with sigmoid activation function is, the **Output of the function is NOT zero centred.** The output of the sigmoid function (not the derivative) always ranges from 0 to 1. So, if the data is being not zero centered, more computation time will be taking up. So, to reach the convergence ie. global minima, it will take time.
3. **Power operations are relatively time consuming.**

C. Hyperbolic Tangent (tanh) Function: This is similar to sigmoid function, but here the values can go below zero as well.



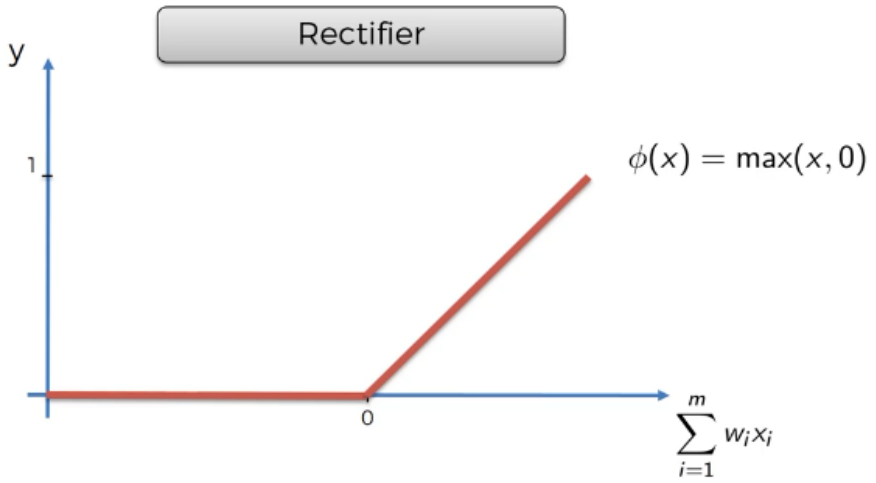
The derivative of the tanh function ranges from 0 to 1. As the output of tanh is zero centred, it is obviously better than sigmoid.

Disadvantages of tanh

Like sigmoid, the tanh also suffers from vanishing gradient problem.
This is computationally expensive function as it involves complex function derivatives.

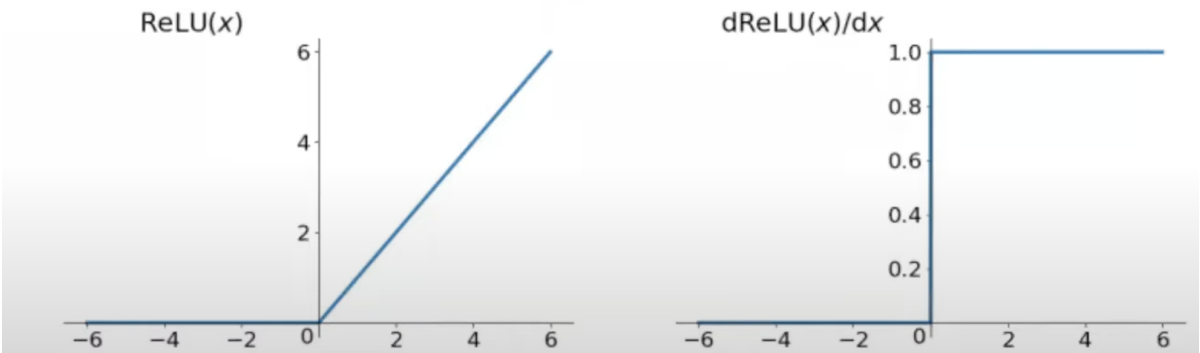
The below combination of activation functions, Rectifier in the hidden layers and the Sigmoid function in the layer is quite common.

D. Rectifier Function: This is one of the most used functions in the ANN.



ReLU function formula and curve are as follows

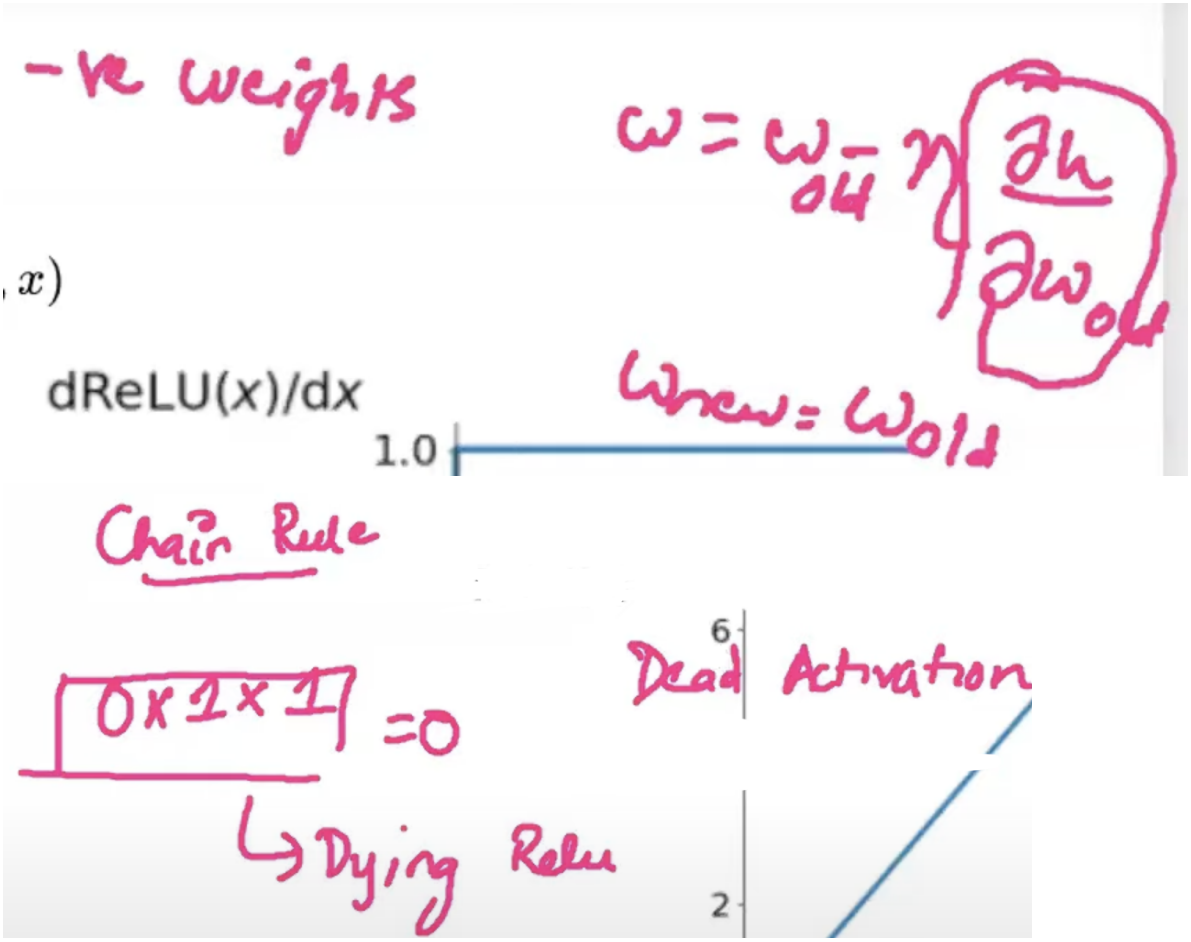
$$\text{ReLU} = \max(0, x)$$



max(0, x) gives that whenever the input is less than zero then it outputs 0. if the input is greater than 0, then the output is the same value.
When we do derivative of the ReLU function, max (0, x), if the value is zero then as the derivative of a constant is 0 we get 0. As the derivative of x wrt x is 1, we get 1. So, this way, in the ReLU, the derivatives are either 0 or 1.

<u>Problems of ReLU</u>
There is a problem here that, practically we can not calculate the derivative of a 0.

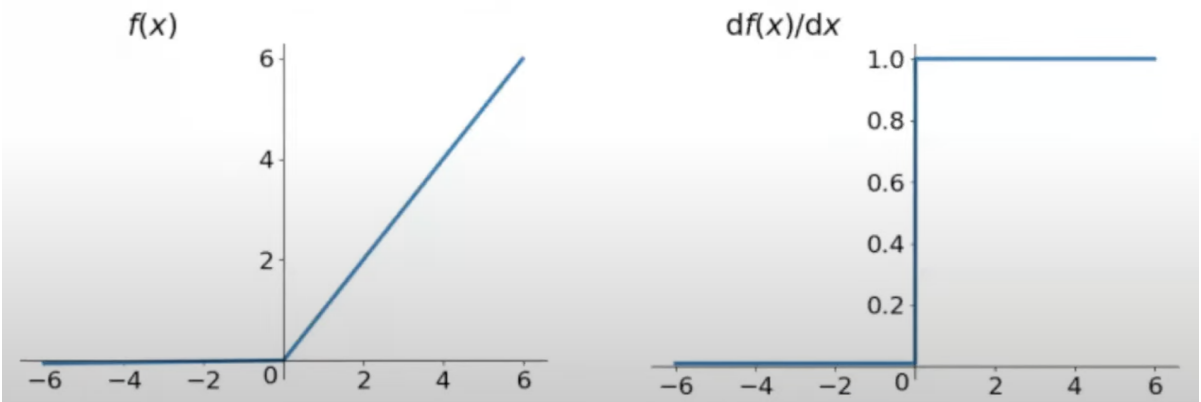
The other problem is that whenever we have a negative weight, during the back propagation, according to the chain rule the whole multiplication become zero and the the derivative of the weight becomes zero, and hence the new weight = old weight. **In this case the weight updation will not happen. This scenario is called as Dying ReLU or Dead Activation Function.**



Q: Where do we use ReLU and why?
ReLU is the most used activation function used in the hidden layer, because it solves the vanishing gradient problem.

E. Leaky ReLU

$$f(x) = \max(0.01x, x)$$



To overcome the dead activation function or dying relu, this leaky relu has come in to existence. As the derivatives of the zero is not possible, and the negative

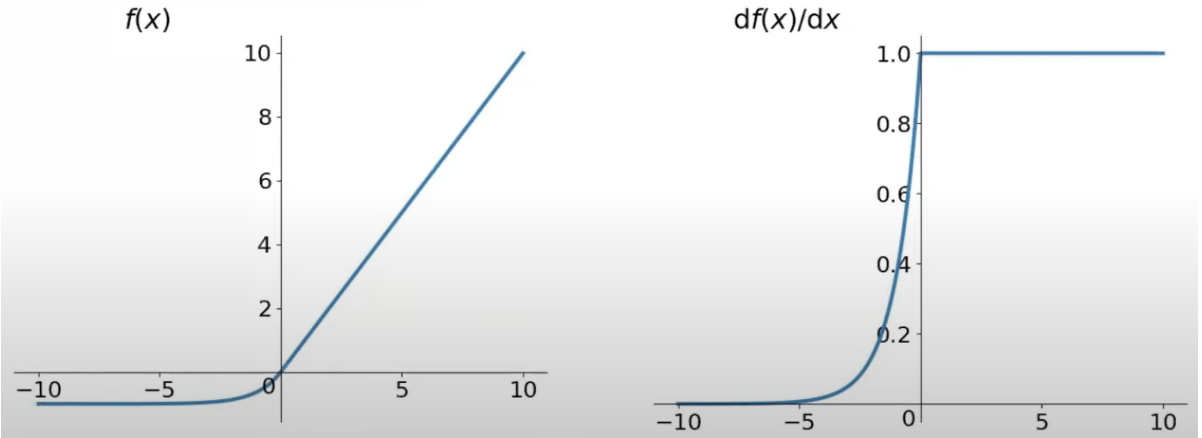
weights are becoming zero, a small constant, *0.01x* has been introduced in this function.
In theory, Leaky ReLU has all the advantages of ReLU, plus there will be no problems with Dead ReLU, but in actual operation, it has not been fully proved that Leaky ReLU is always better than ReLU.

Problems with Leaky ReLU

Leaky ReLU also will encounter vanishing gradient problem. This happens, in the deep neural networks, when the chain rule is applied, there will be many derivatives whose values are *0.01*x* , so that the whole expression will becomes very small value and when multiplied with learning rate (eta) then the value further becomes smaller and therefore, the new weight becomes approximately equal to the old weight.

F. ELU (Exponential Linear Units) function

$$f(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha(e^x - 1), & \text{otherwise} \end{cases}$$



It is almost similar to the Leaky ReLU, but here we are handling the negative values effectively here by introducing the alpha. This alpha can also be treated as a hyper parameter. We can tune this.

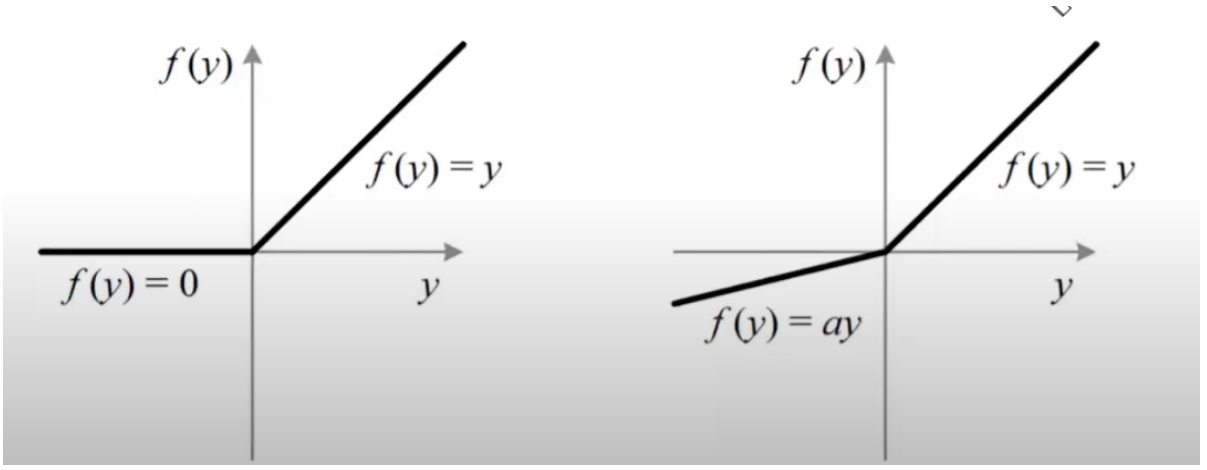
Advantage of ELU

- Here we will never encounter the situation of finding derivative of a zero. So, this major problem is overcomes here.
- No Dead ReLU issues
- The mean of the output is close to 0, zero-centered

Disadvantage of ELU

ELU is computationally expensive as it involves derivative calcuation of e power x function.

G. PRelu (Parametric ReLU)



$$f(y_i) = \begin{cases} y_i, & \text{if } y_i > 0 \\ a_i y_i, & \text{if } y_i \leq 0 \end{cases}$$

Mathematical definition of PReLU

Above, y_i is any input on the ith channel and a_i is the negative slope which is a learnable parameter.

- if a_i=0, f becomes ReLU
- if a_i>0, f becomes leaky ReLU
- if a_i is a learnable parameter, f becomes PReLU

ai can also be called as alpha. Y i can be treated as x.
PReLU is also an improved version of ReLU.

H. Softmax

Sigmoid only gives two outputs as probabilities, whose sum will be equal to 1. So, in cases where we have more than 2 classes (multi-class classification), we use softmax activation function, which gives probabilities of each class and all those probabilities summation will be equal to 1.

$$S(x_j) = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}}, j = 1, 2, \ldots, K$$

If we have values of *x (i.e. (weights*inputvalue) + bias)* as below at different nodes in the output layer

$$[40, 30, 10, 5]$$

then the sigmoid function applied at those nodes will be as follows.

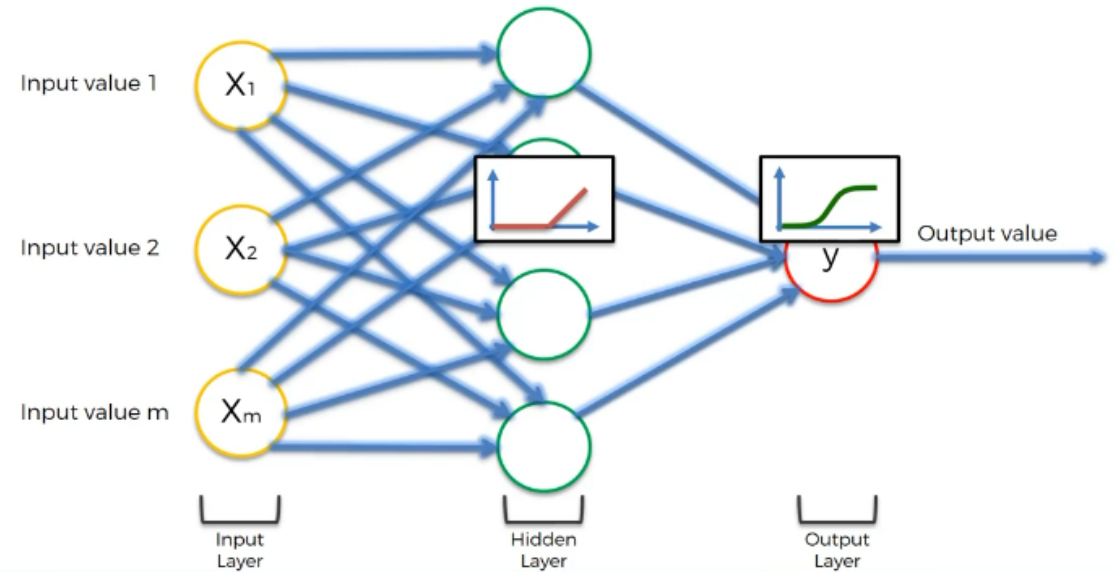
$$\boxed{\frac{e^{40}}{e^{40}+e^{30}+e^{10}+e^{5}}}$$

$$\frac{e^{30}}{e^{40}+e^{30}+e^{10}+e^{5}} \qquad \frac{e^{10}}{e^{40}+e^{30}+e^{10}+e^{5}}$$

So, in this way we get different probabilities for each class for an observation.
Sigmoid and Softmax are both always used at the output layer.

Other Activation Functions

Swish, Softplus

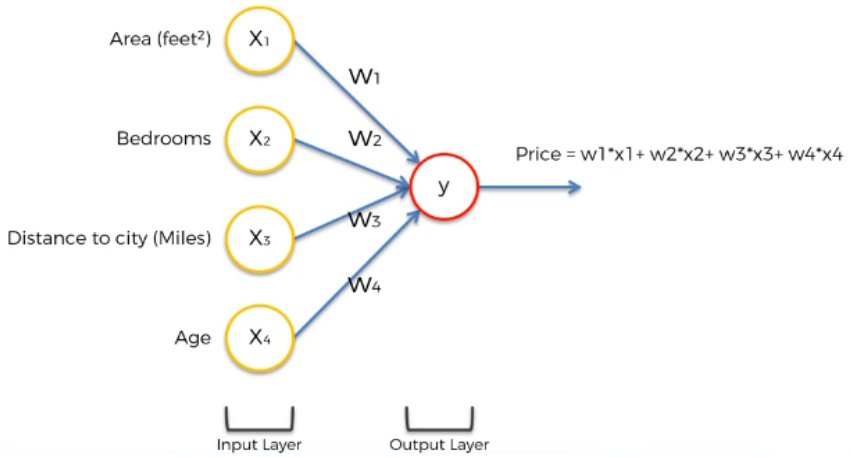


Question: Why is the RELU most used function in the hidden layers? What are the advantages of RELU?
Q: What are the problems with Sigmoid activation function?

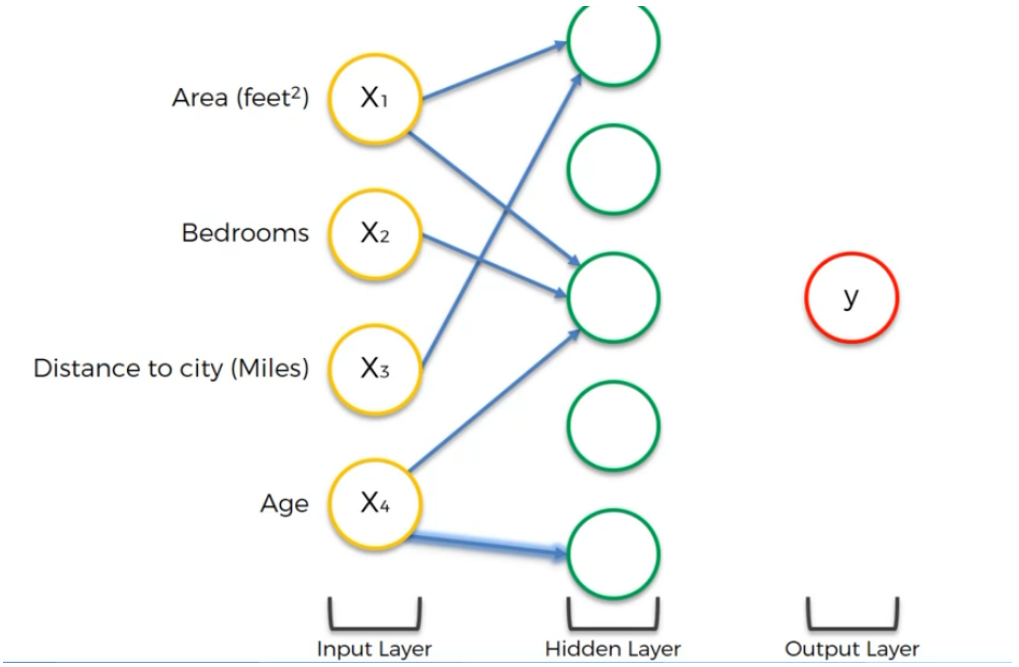
Q: What are the problems with RELU? What is Leaky RELU? Why is Leaky RELU useful over RELU?
Q: What is the problem with Tanh activation function?
Q: When do we use sigmoid activation function?
Q: When do we use softmax activation function?
Q: What activation do you use in case of a multi class classification?

3. **<u>How do Neural Networks work?</u>**

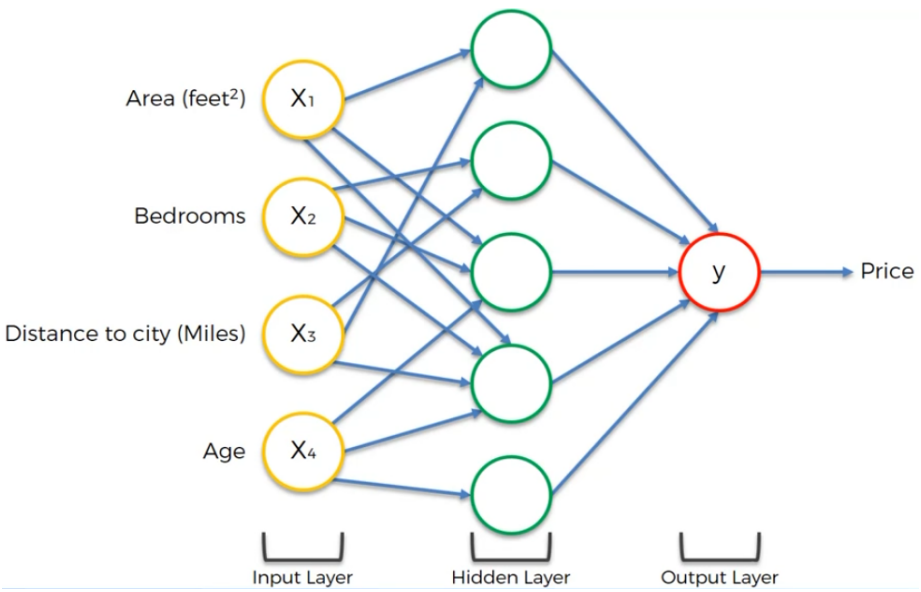
Consider an example of predicting house prices.



Above is the basic neural network without any hidden layers. It just has input and output layers and the output layer will have the activation function. The output can be represented as a weighted sum of the features. But, the real power of the neural networks lies in the hidden layers, as they can learn so many permutation combinations of the features.

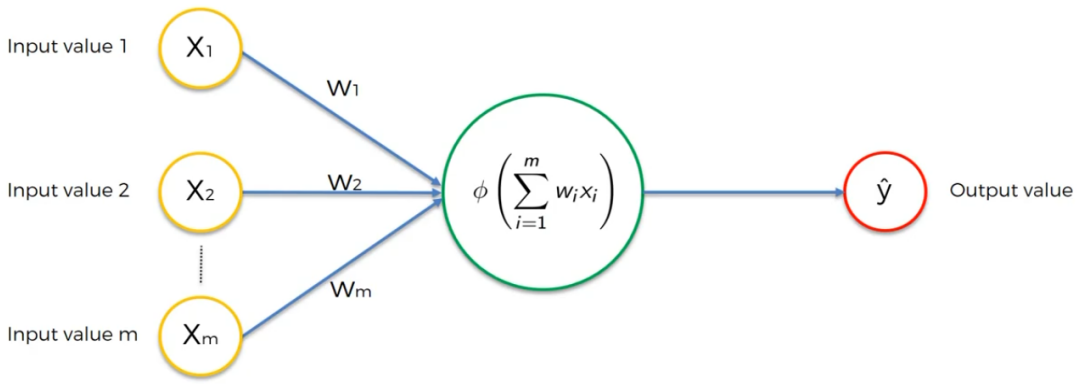


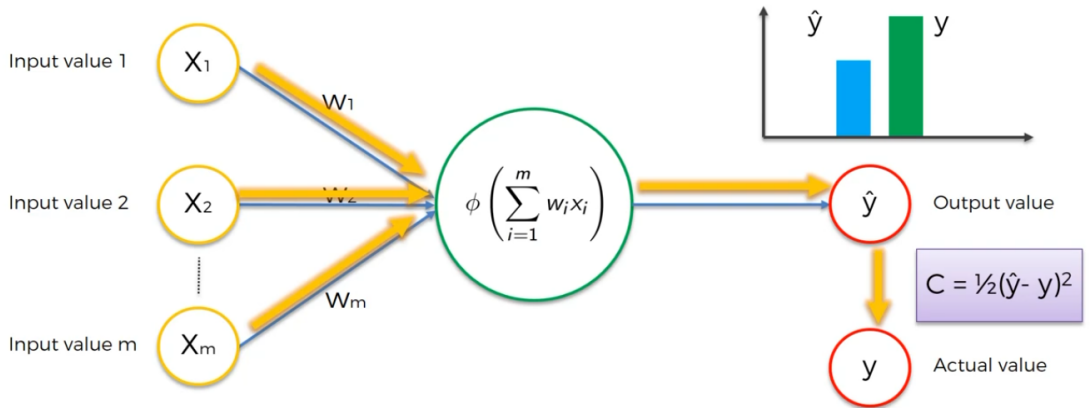
In the above example, the first neuron in the hidden layer, is learning only the patterns involving the Area, Distance to City. For rest of the features the weights of those synopsis will be zero. The middle (3rd) neuron is learning only the patterns involving Area, Bedrooms and Age variables. In the real estate, when some properties more than hundred years, will be considered as a historic property and it will have huge value. So, the last neuron is learning those kind of patters, where the focus is only with Age. Each neuron individually can not learn multiple patterns in the data. But together, with a network of multiple neurons the whole neural network is much powerful in learning the whole dataset patterns and will be able to predict the price of a house.



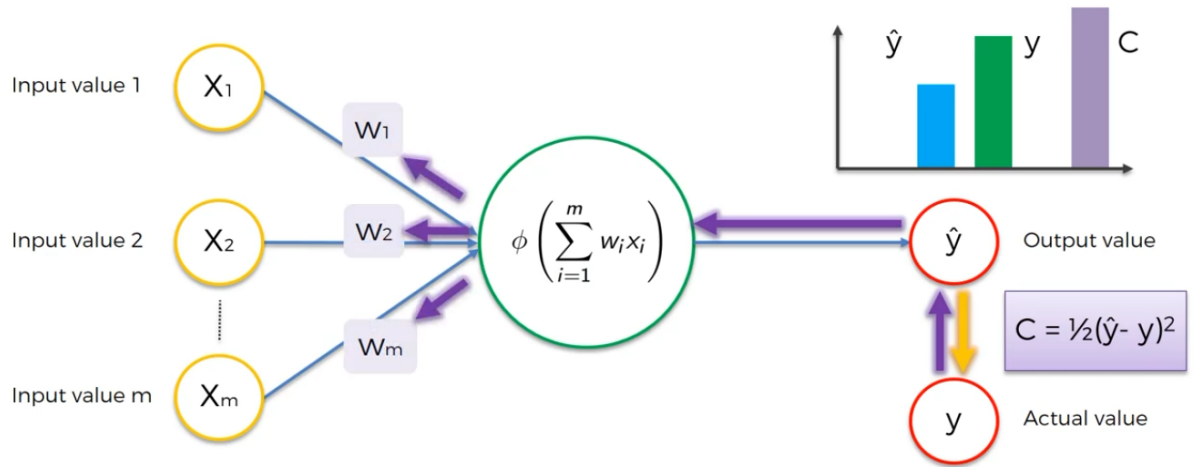
4. **How do Neural Networks Learn?**

Below is a single layered feed-forward neural network. Also called as Perceptron.

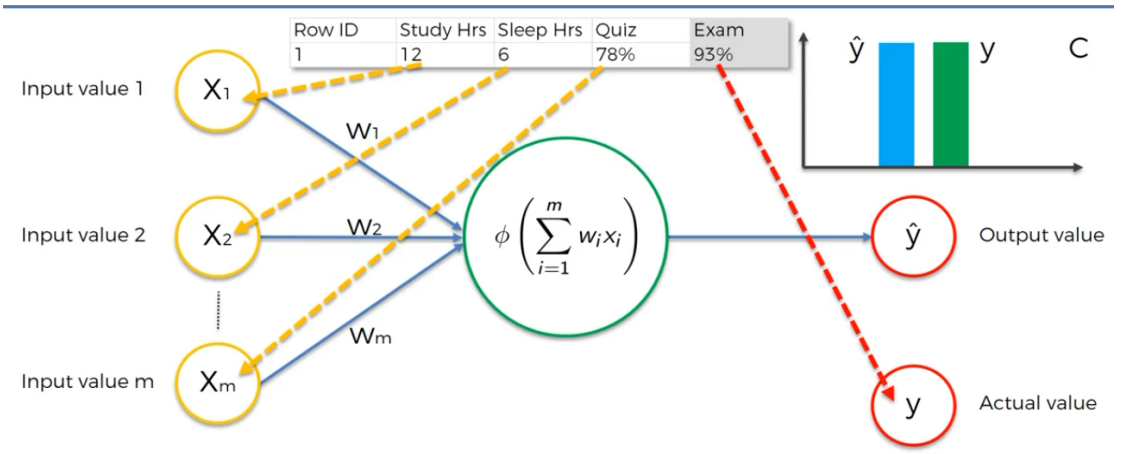




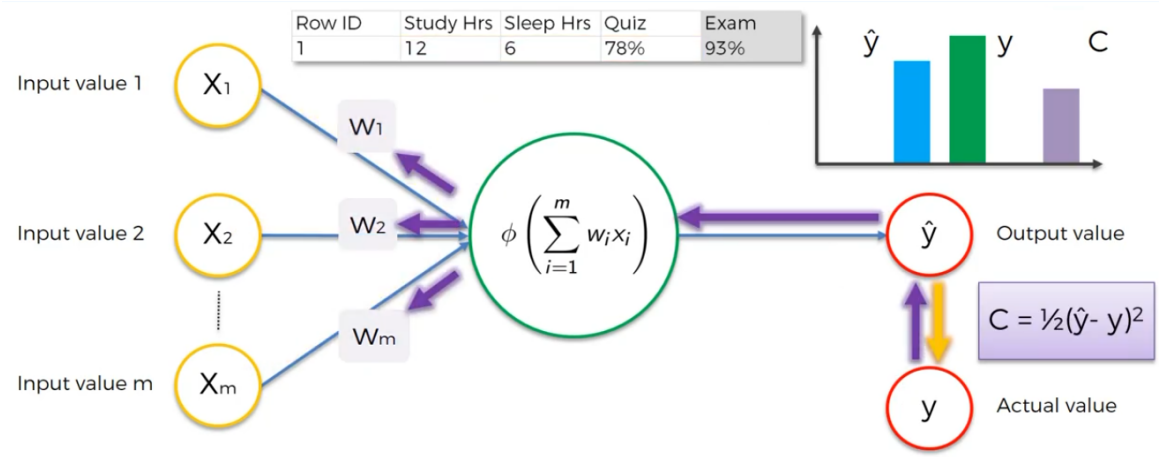
First an observation is passed and the output is calculated by keeping the weights at a default values. The calculated output (y^) is compared with actual output (y) and a cost function is calculated. There are different no.of cost functions. But here a simple cost function is used. Cost function basically gives us the error made in the prediction. Our goal is to minimize the cost function.



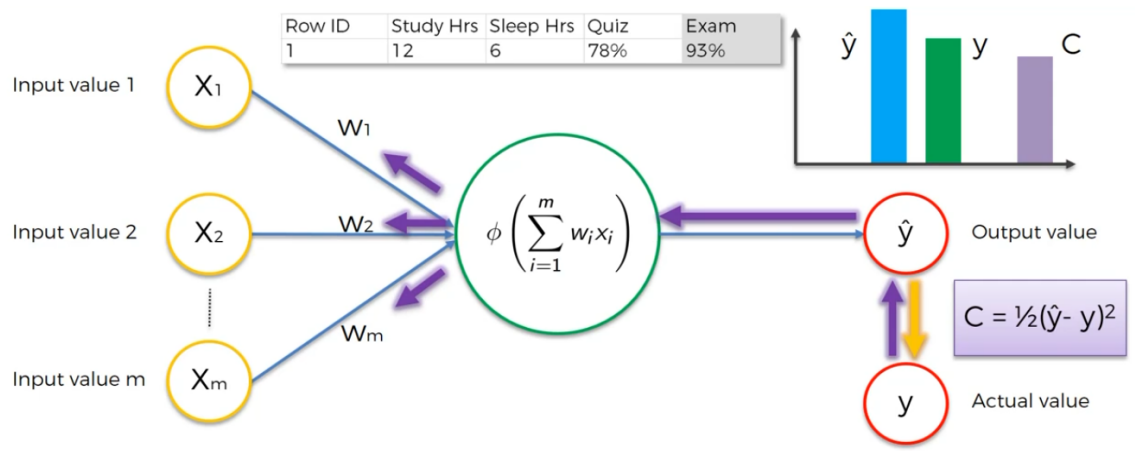
Now, the information about the cost function is feed back (Purple arrows) in to the neural network and the weights are adjusted accordingly.
The only control we have on this network is adjusting the weights. So, by adjusting weights we aim to reduce the cost function.



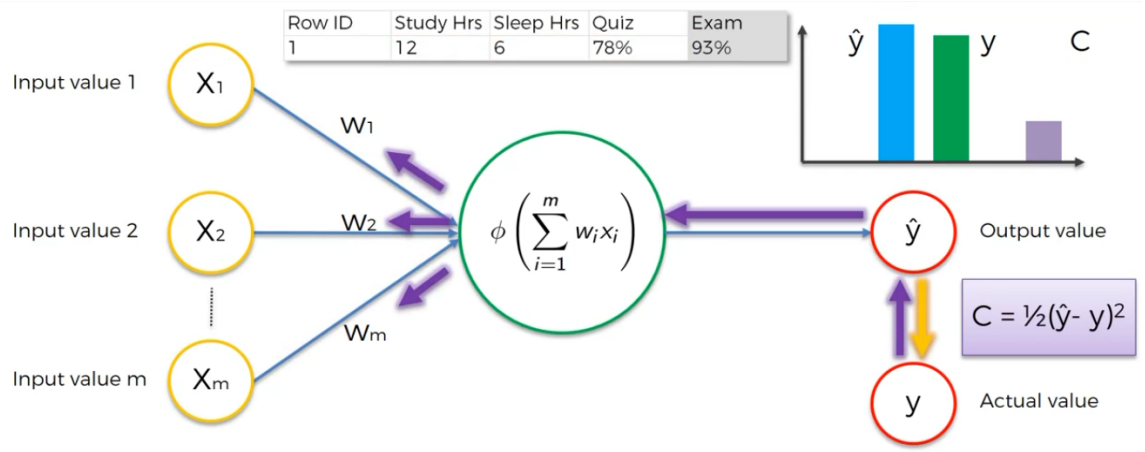
Remember that we are working on only one observation the whole time as part of this example to understand the neural network learning mechanism.
Now, with the adjusted weights, we try to predict the output one more time.



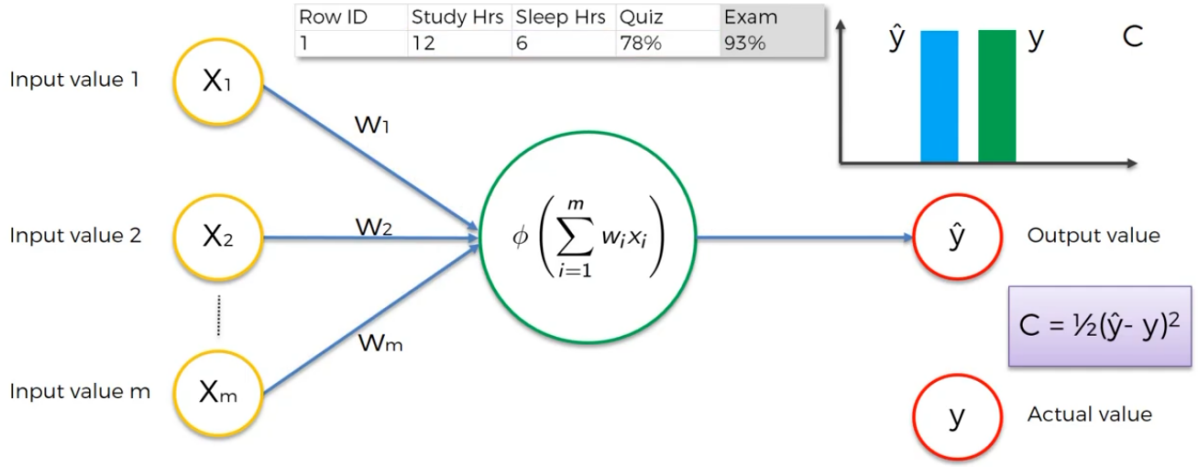
This time, the cost function is reduced as shown in the top right corner graph. The same info is passed to the feedback loop and the weights are again adjusted.



Again the prediction is made using the new weights, this time, the y^ is higher than the actual value. So, again the cost function is calculated and the info is fed back to the neural network and the weights are adjusted again.



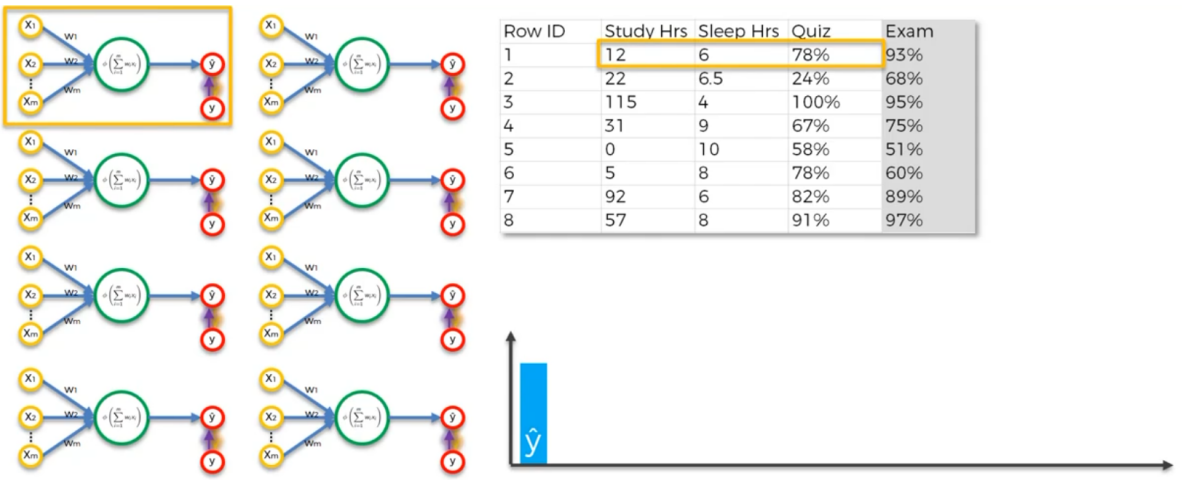
The cycle is repeated and the cost function is further reduced.



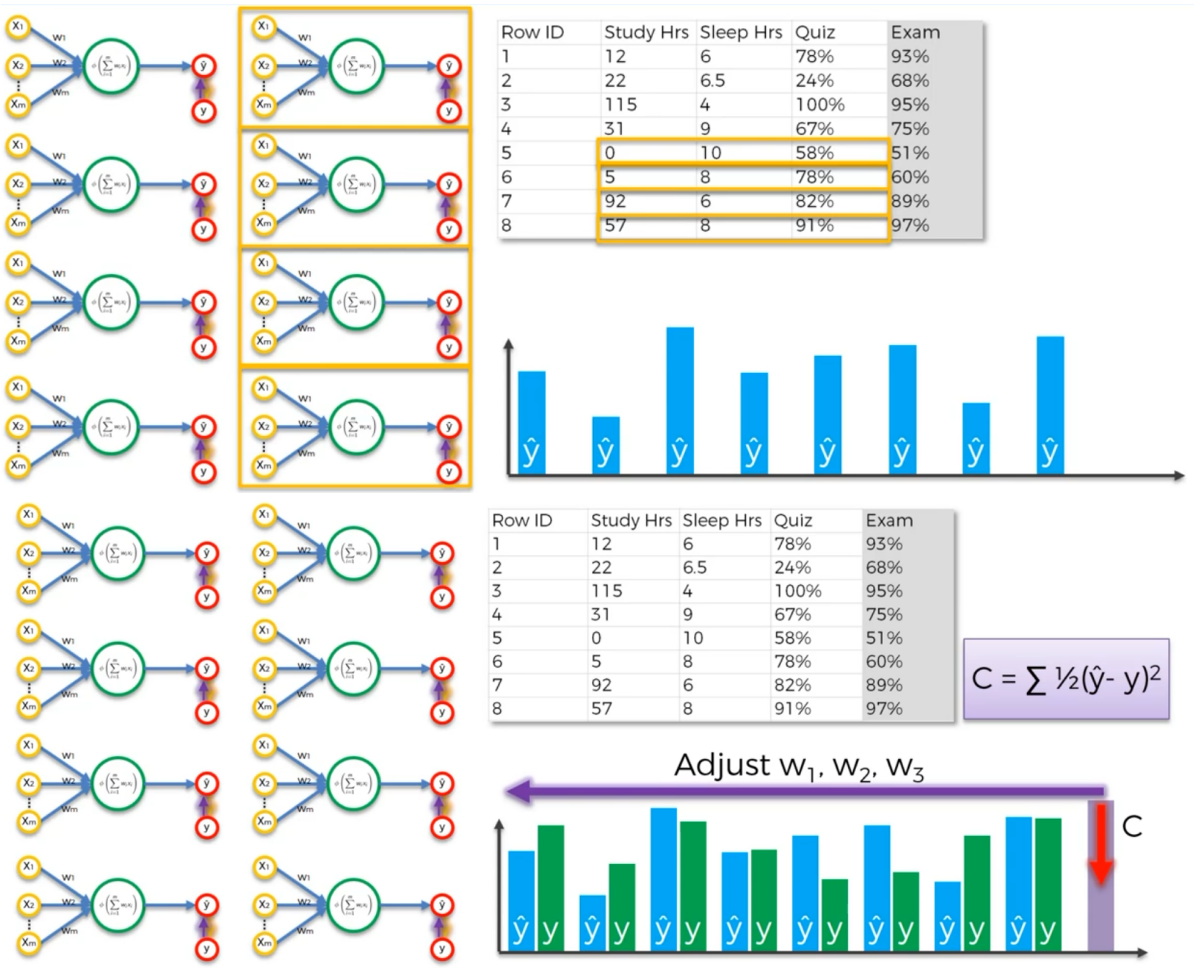
By repeating this cycle, we get the cost function to a minimum value. Ideally the cost function will never be zero.

Till now, we trained the neural network only on one record. Lets work on a dataset with multiple observations as below.

Note: Just for illustration purpose the same perceptron is copied 8 times. But, actually all the records are passed through one perceptron only.



First the 1st record is passed through the perceptron and the output y^ is calculated. The same process is repeated and y^ for all the rows is plotted in the graph.

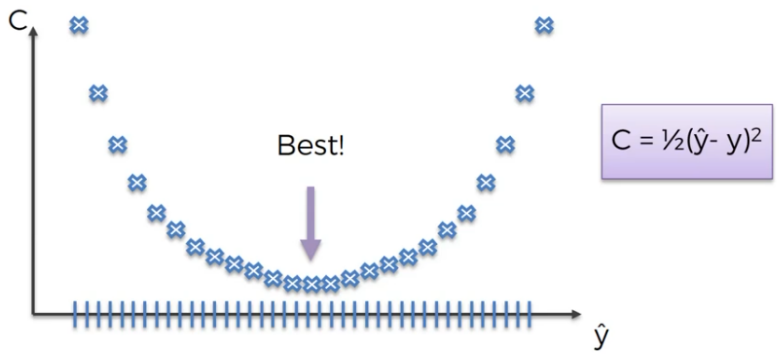


Now, the cost function is calculated for the whole dataset using the actual values. The weights are adjusted and then predictions are made again and cost function is calculated again and the cycle repeats.

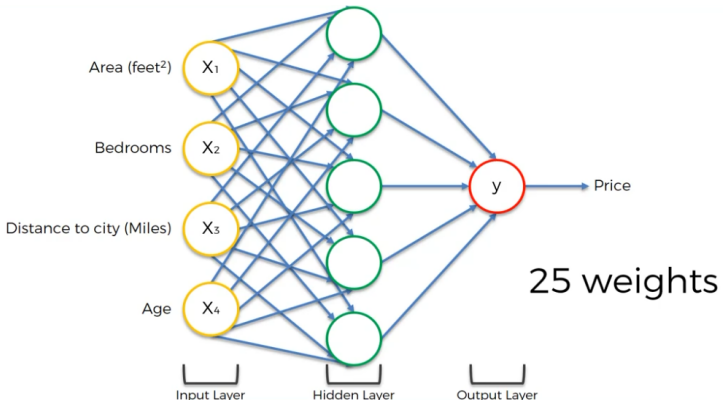
Note: The weights will be same while processing all the rows. There won't be different weights for different rows.
This whole process is called Back Propagation.

5. **<u>Gradient Descent</u>**

When we try to minimize the cost function we have the option of Brute force (just trying out different random values).
When we plot the cost function values against different y^ values. The best y value to get the minimum cost function occurs at the bottom of the curve. This brute force method will not work when we have many no.of synopsis and weights. We might face the curse of Dimensionality.



Consider the below example, in which we have only 4 input variables and 1 hidden layer with 5 neurons, we have a total of 25 weights.



For example if we have 1000 possible values for each of the 25 weights, then we will have

$1{,}000 \times 1{,}000 \times \ldots \times 1{,}000 = 1{,}000^{25} = 10^{75}$ combinations

Sunway TaihuLight: World's fastest Super Computer

93 PFLOPS

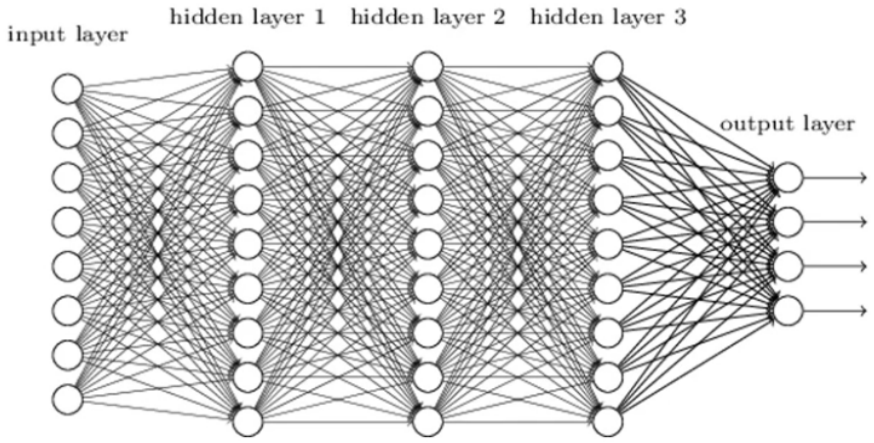
93×10^{15}

$10^{75} / (93 \times 10^{15})$

$= 1.08 \times 10^{58}$ seconds

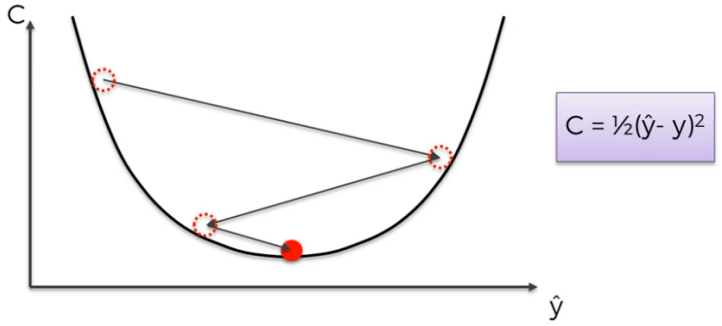
$= 3.42 \times 10^{50}$ years

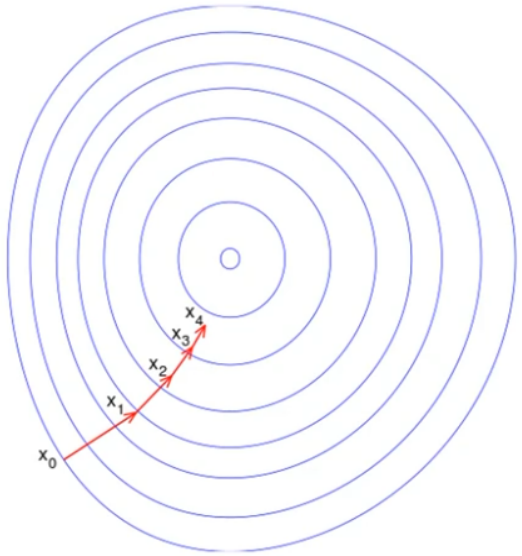
So, even with the world's fastest computer it would take 3.42 x 10^50 years, which is impossible. The above example is a very simple one. What if the neural network looks something like below. So, it is impossible to minimize the cost function with best optimal weights. We need an algorithm to solve this problem. That solution is Gradient Descent.

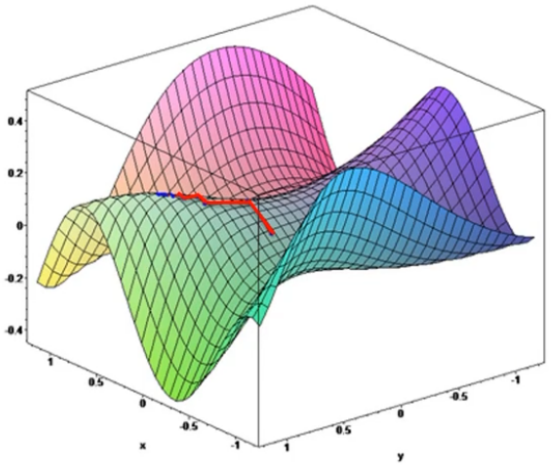


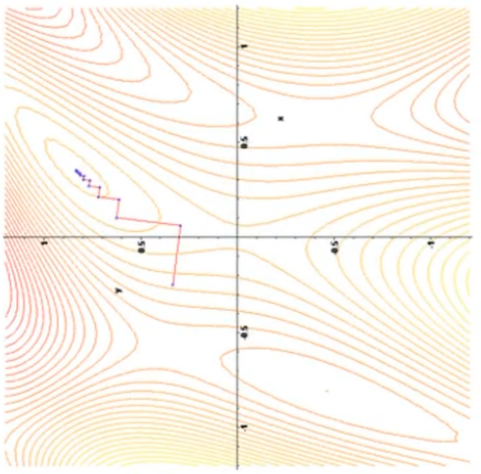
Working

We start at some random place. At that point, calculate the gradient/slope (differentiate the cost funct). If the slope is negative, then we are going downhill (right). At the second step again calculate the gradient. If the slope is positive then go left to go downhill. Repeat this until you find the optimal cost function value. Below is the gradient descent applied on 1-D and 2-D and 3-D (last on is 3D projected to a 2D)space.



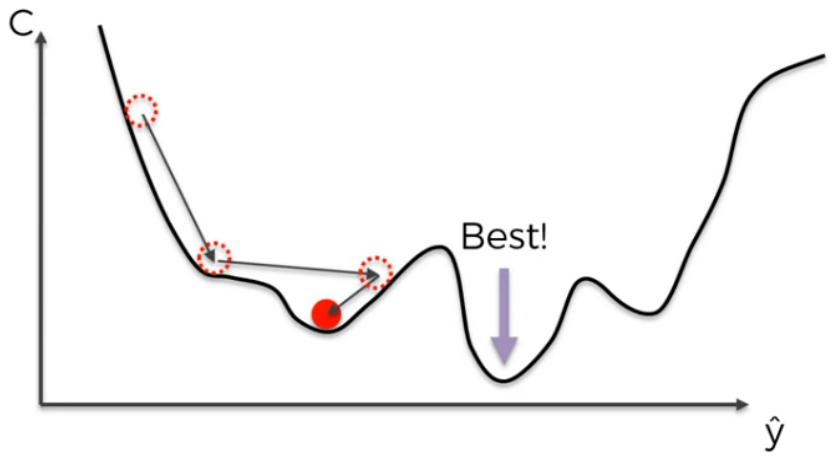




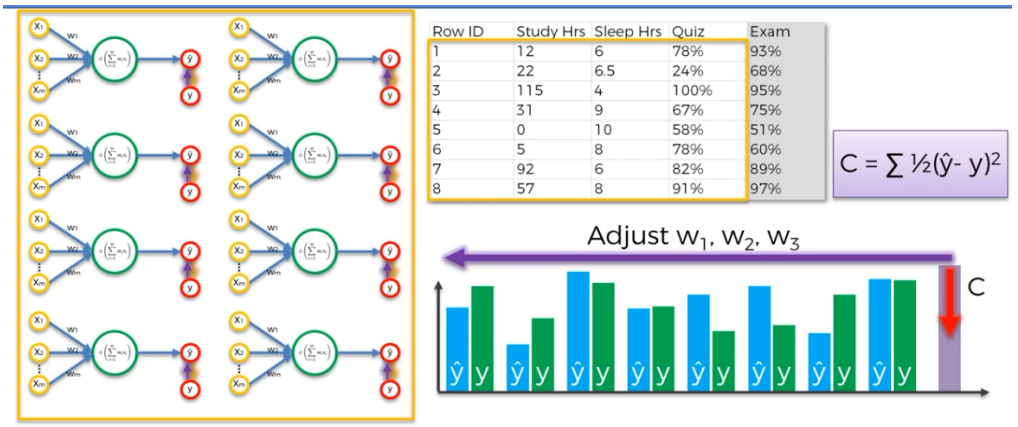


6. **<u>Stochastic Gradient Descent</u>**

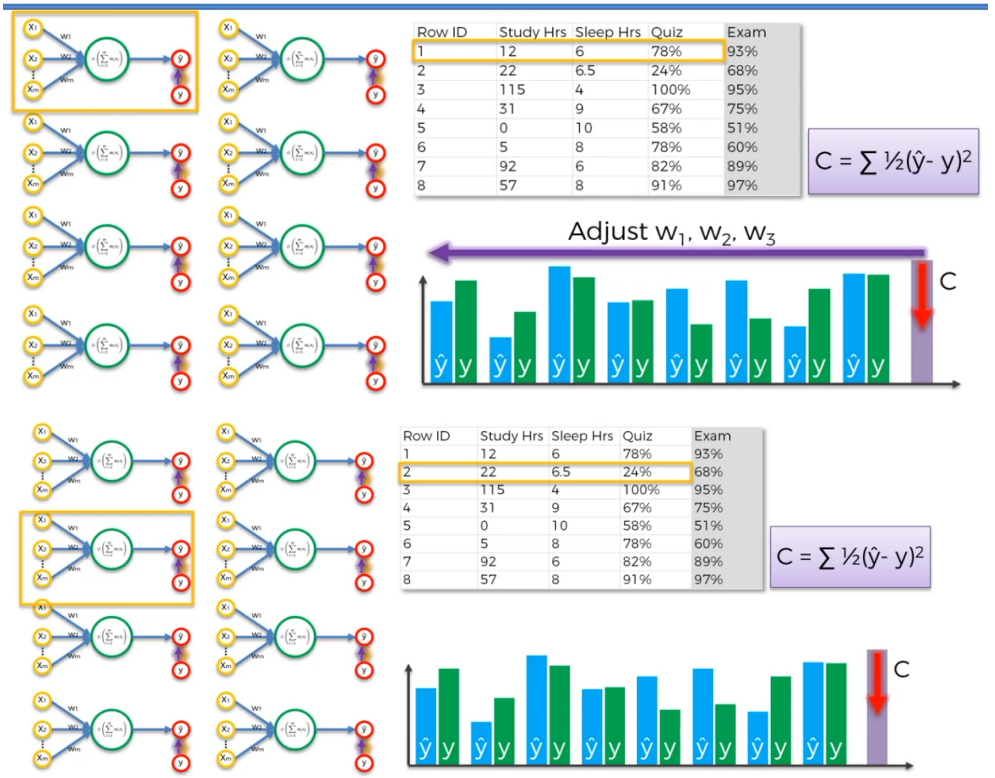
The Gradient Descent works only with the convex cost function. It means it should convex in one direction or it should have only one global minimum. Gradient descent finds the local minima as shown in the below figure. So, in cases like this we need Stochastic Gradient Descent algorithm.



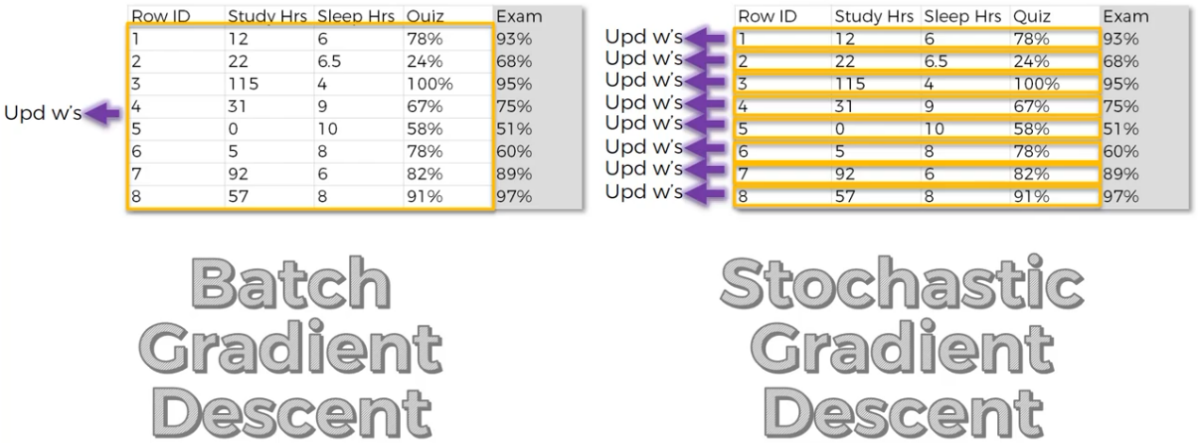
Unlike the gradient descent the stochastic gradient descent doesn't require the cost function to be convex.



The above example uses gradient descent to optimise the cost function. As the whole dataset is taken as a batch, it is called Batch Gradient Descent.



Here, in the Stochastic gradient descent we adjust the weights for every observation. This is the main difference between gradient descent and stochastic gradient descent.



As the stochastic gradient descent is updating weights after passing every observation, it has much more fluctuations and it is most likely to avoid the local minima.

The other advantage of Stochastic Gradient is, it is much faster than the Batch Gradient Descent. The reason for this is, it doesn't need to load the whole data in to memory and run and wait until all those rows are run all together. As it is running one record at a time, it is a much lighter algorithm. So, it is faster.

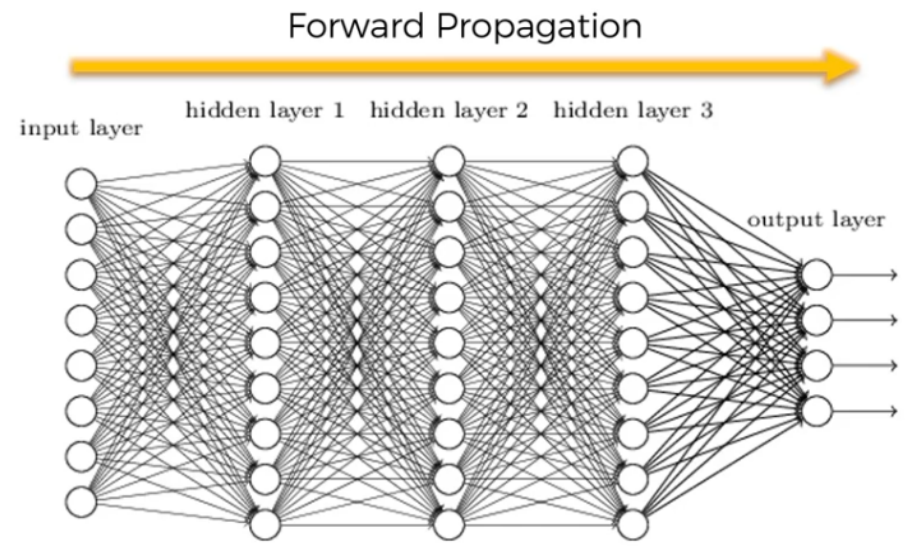
The advantage of Batch Gradient Descent is that, it is a deterministic algorithm, where as Stochastic is a random. With the same initial weights if you run the Batch Gradient Descent you will end up with same results. But Stochastic GD is random and doesn't give you same results.

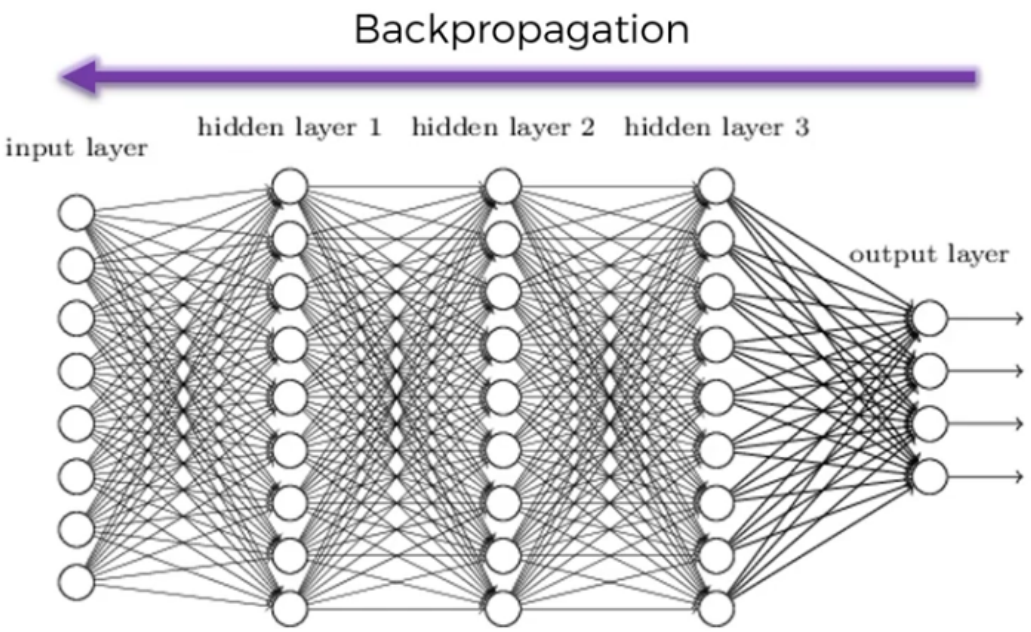
Mini Batch Gradient Descent is a combination of batch and stochastic. Means we divide the data in to mini batches and run it.

A neural network in 13 lines of python by andrew trask

7. **Back Propagation**

We know that there is a process called Forward Propagation, where information is entered in to the input layer and then is propagated forward and the output is predicted. The output is compared with the actual values and the errors are calculated. Then the errors are back propagated through the network in the opposite network. That allows us to train the network by adjusting the weights.





Back Propagation is an advanced and an intense mathematical algorithm which adjusts all the weights at the same time. Because of the way the back propagation algorithm is designed, we basically knows which part of the error, each of your weights in the neural networks are responsible for.

Chain Rule

Chain rule is the basic fundamental concept through which the whole back propagation algorithm works. It uses the chain rule of differentiation. To understand more about it, watch the following video. (https://www.youtube.com/watch?v=CRB266Eyjkg)

https://www.youtube.com/watch?v=CRB...

Vanishing Gradient Problem

Before the invention of the RELU activation, the only option to use as an activation function is 'Sigmoid'. The derivative of a sigmoid ranges only from 0 to 0.25. When the number of layers are increased and the during the gradient calculation the no.of derivatives increases due to the chain rule. So, in that case the derivative values decreases so much for increasing derivatives in the chain rule and the gradient slowly vanishes. This is called Vanishing Gradient Problem.
So, if we use Sigmoid as the activation function in the hidden layers of a neural network, then we encounter the Vanishing Gradient Problem. So, thats why we use RELU in the hidden layers. For clear explanation watch this video. (https://www.youtube.com/watch?v=JIWXbzRXk1I)

https://www.youtube.com/watch?v=JIW...

Training the ANN with Stochastic Gradient Descent

STEP 1: Randomly initialise the weights to 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 the 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 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).

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

Question: Do we need feature scaling in ANN and in general deep learning?
Yes, as the neural networks are highly compute intensive computations, we need to apply feature scaling to ease off these calculations. And more over we don't want one column dominating other because of the scale mismatching.

8. **Optimizers**