Q1. What is **Deep learning?**

Why people started using so late although Neural Network was even available in 80s as well, then what was the reason behind that.

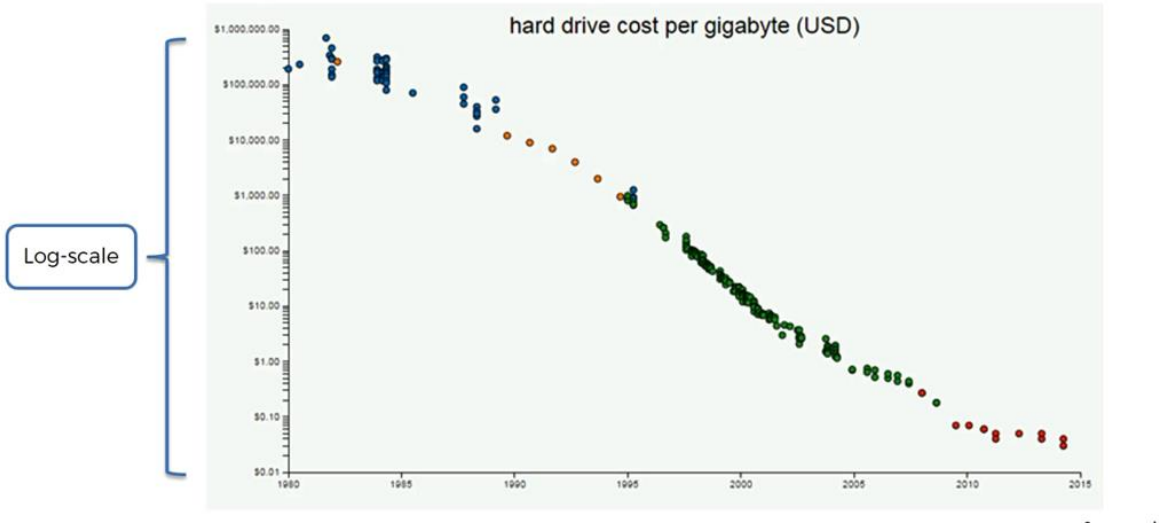
Q2. Why people were not able to use it . Was it because the NN was not evolved completely at that point of time?

No the reason behind that was,

a.) NN need lots of data

b.) NN needs huge machine to process all the data. If you look at the computers in early 80s our machines was not capable to process those kind of computation

The picture which you are able to see in pic 1 is a hard drive of 5MB. A company had to pay two and half thousand dollars of those days dollars. For renting it for one month.

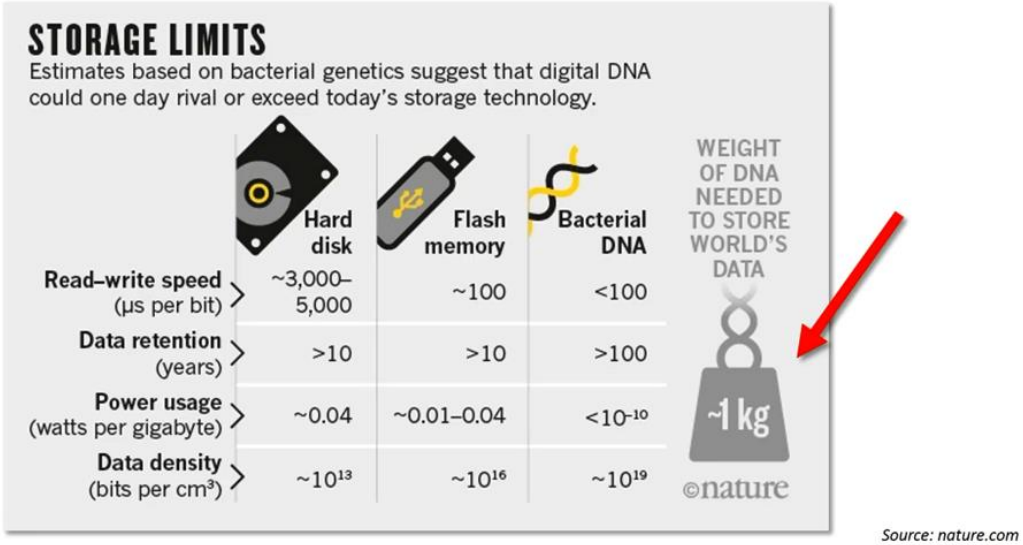


If you plot the Cost of Hard Drive with Years this is how it will look like.

3. Right now scientists are looking DNA for storage.

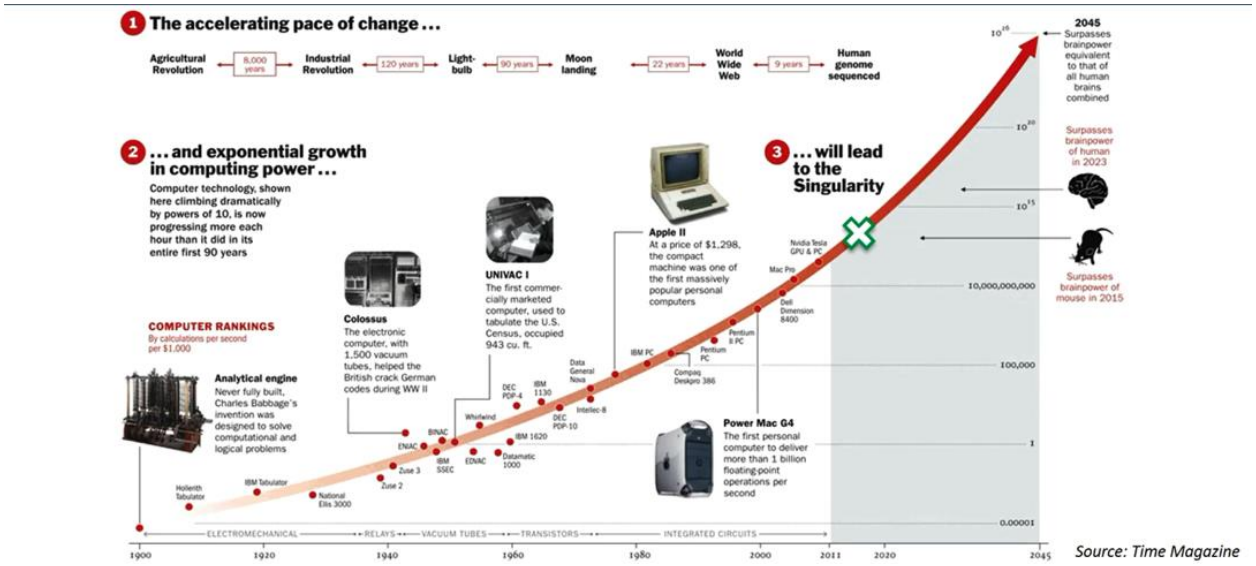
But right now it's quite expensive. It's cost 7000$ to synthesize 2MB of data and there is another 2000$ to read it.

4. if you want to analyze these stats

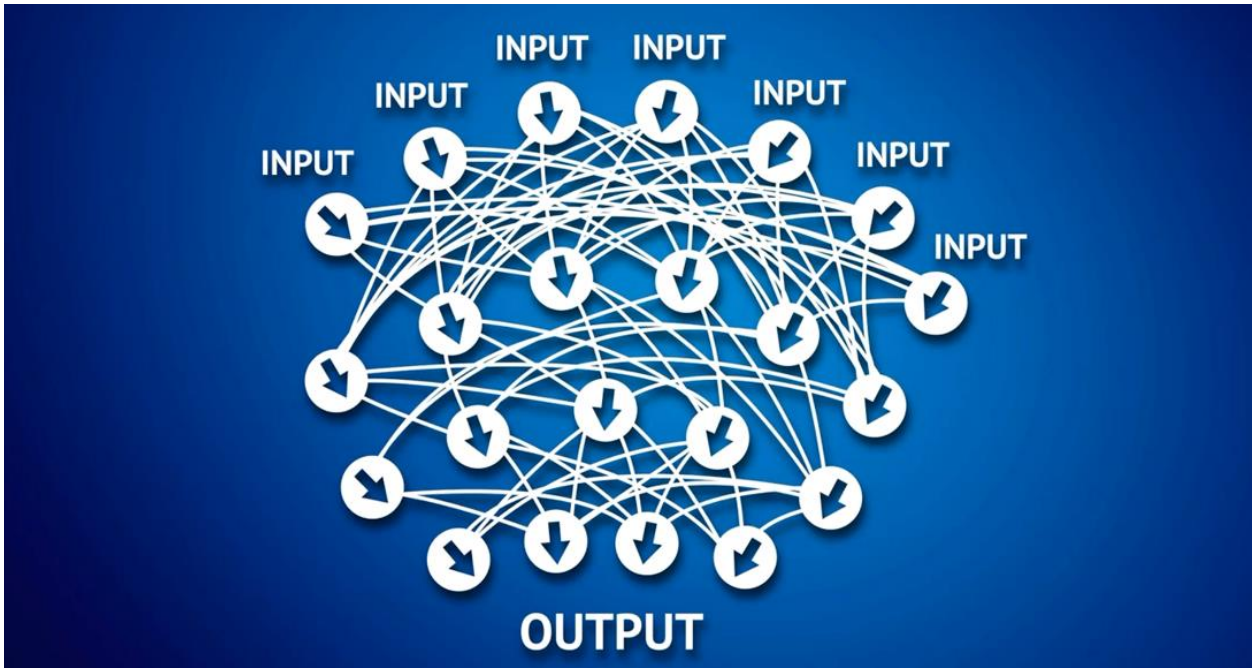


Source: nature.com

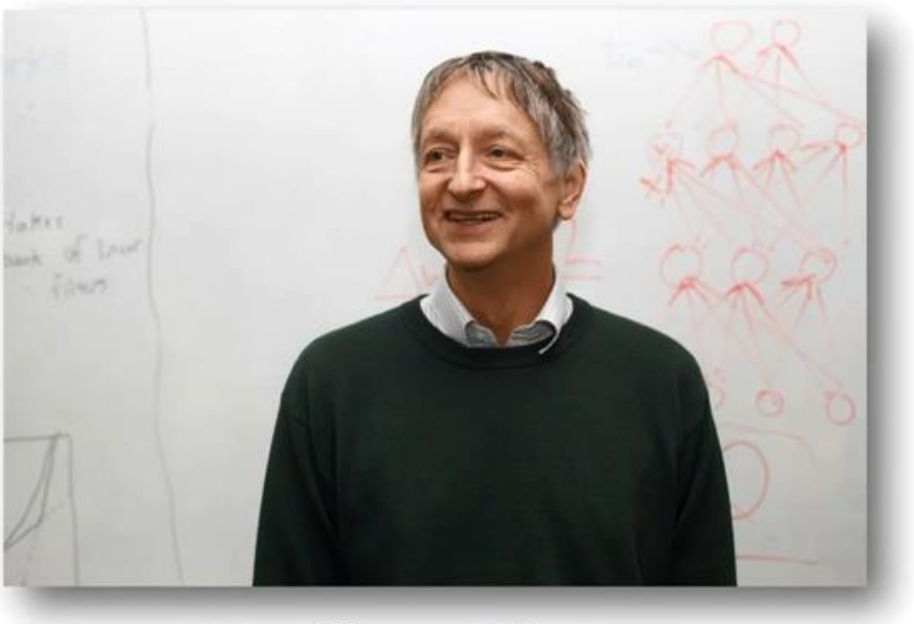
5. Moore's Law Computer Processing power again Stats if you want to analyze



6. Now our main Topic Neural Network

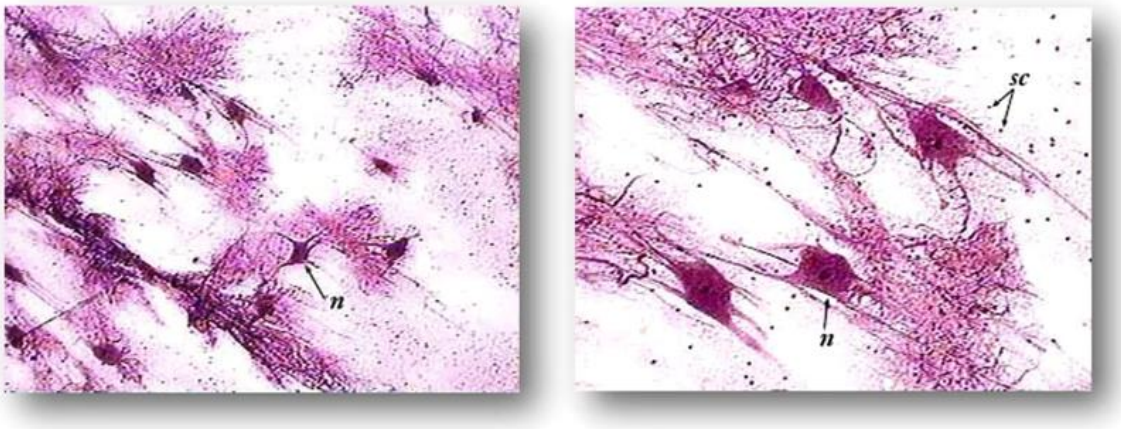


God Father of Deep Learning. He did his research in 80s. Right now he is working at google.

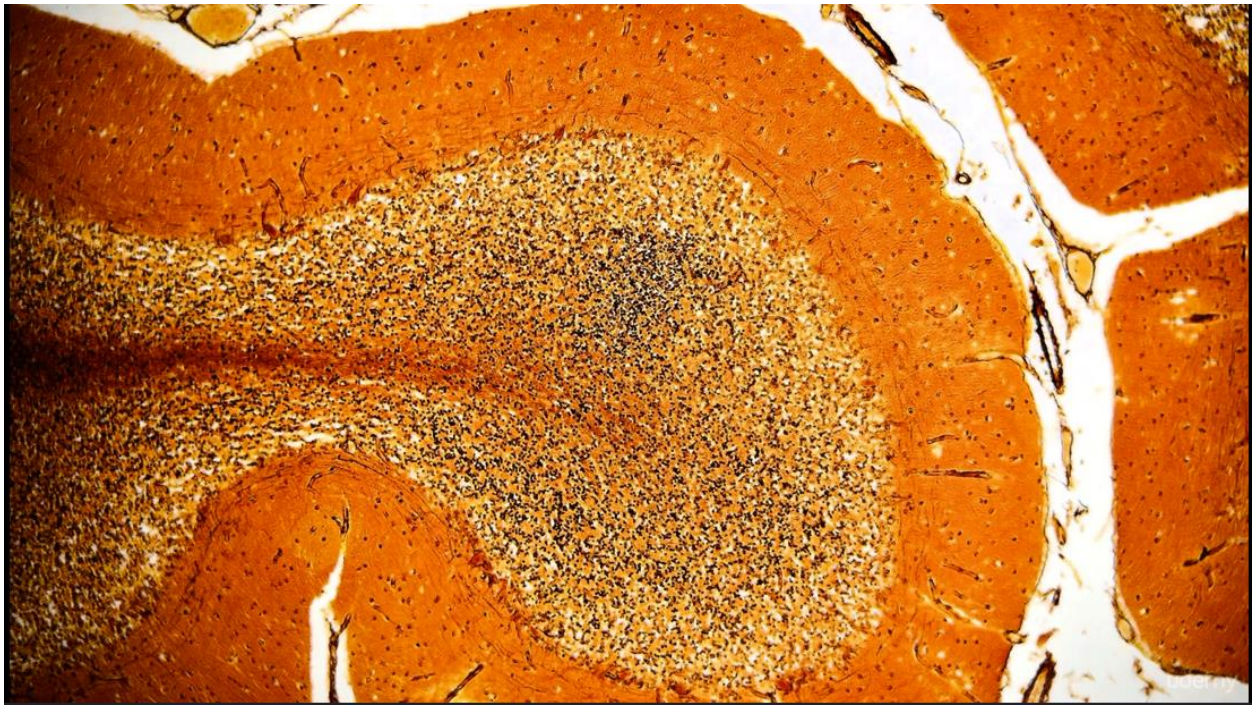


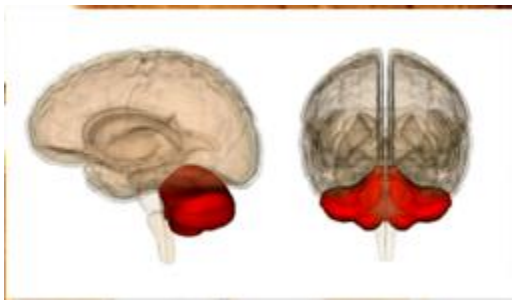
Geoffrey Hinton

How human brain looks like?



In human brain approximately there are 100 billion neurons. These are individual neuron and each neuron is connected to 1000 of it's neighbors.



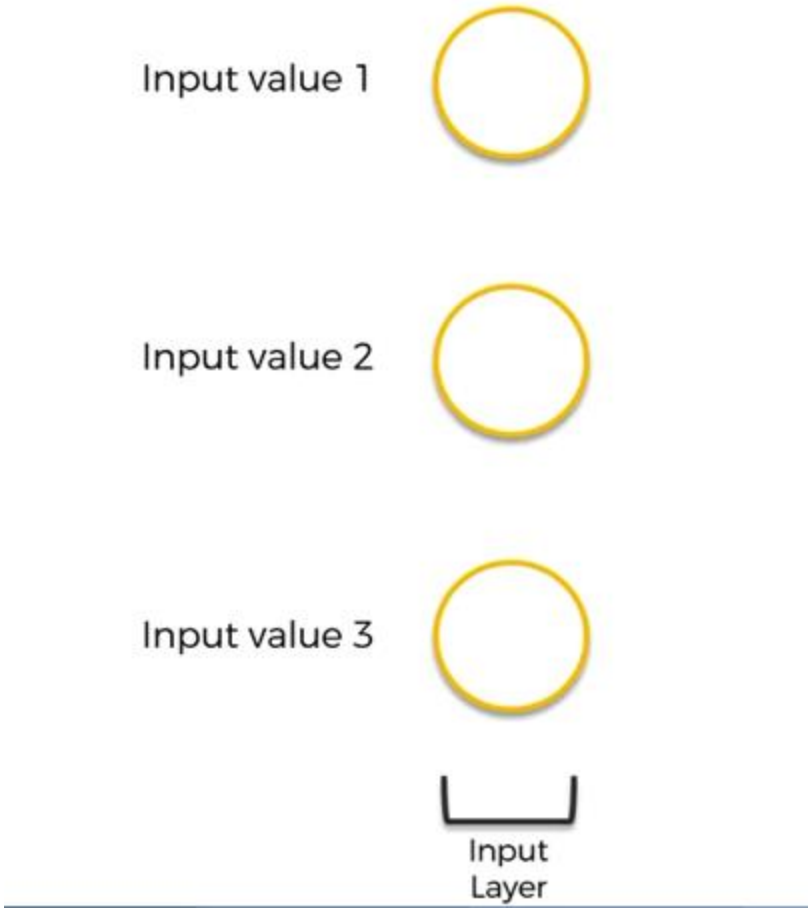


this is how actual neurons look like . This is surbalum of human body.

Now the question is how we can create such structure.

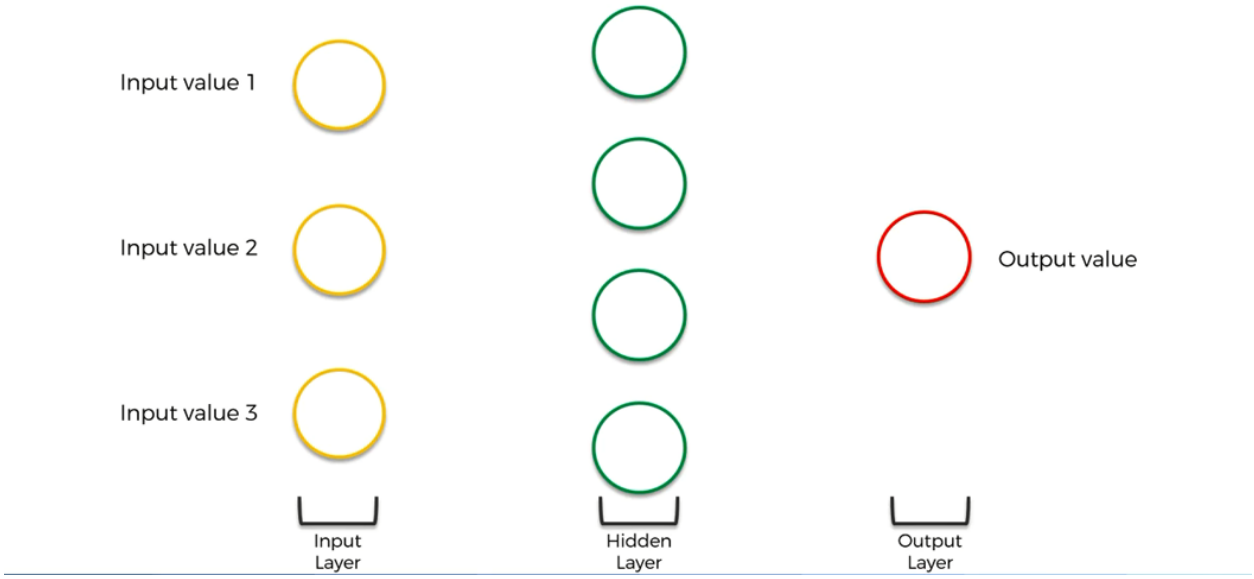
There is a concept call ANN (Artificial Neural Network / Neural Network)

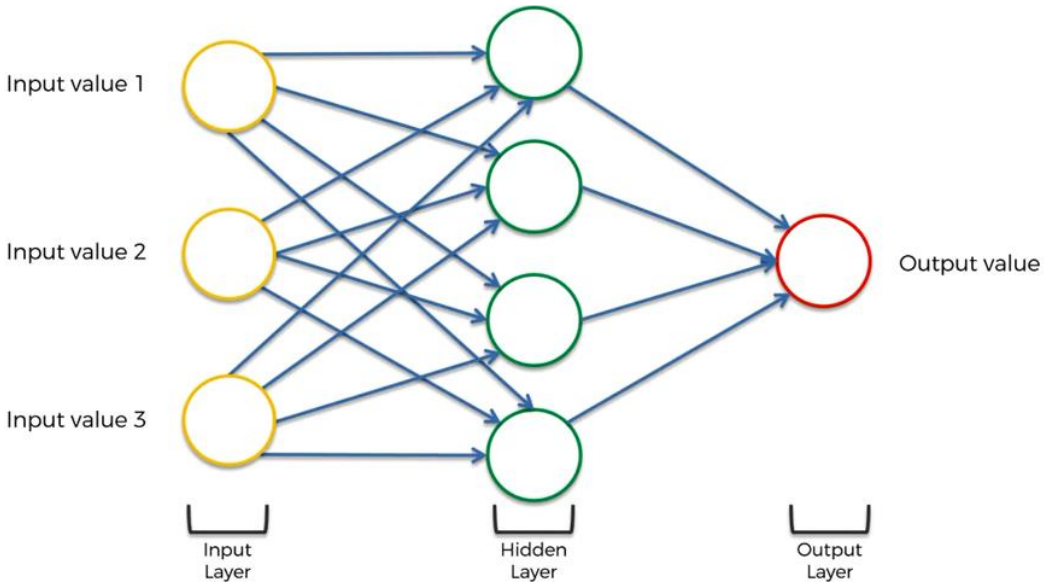
Step 1:



Step2: Then there is a Output layer

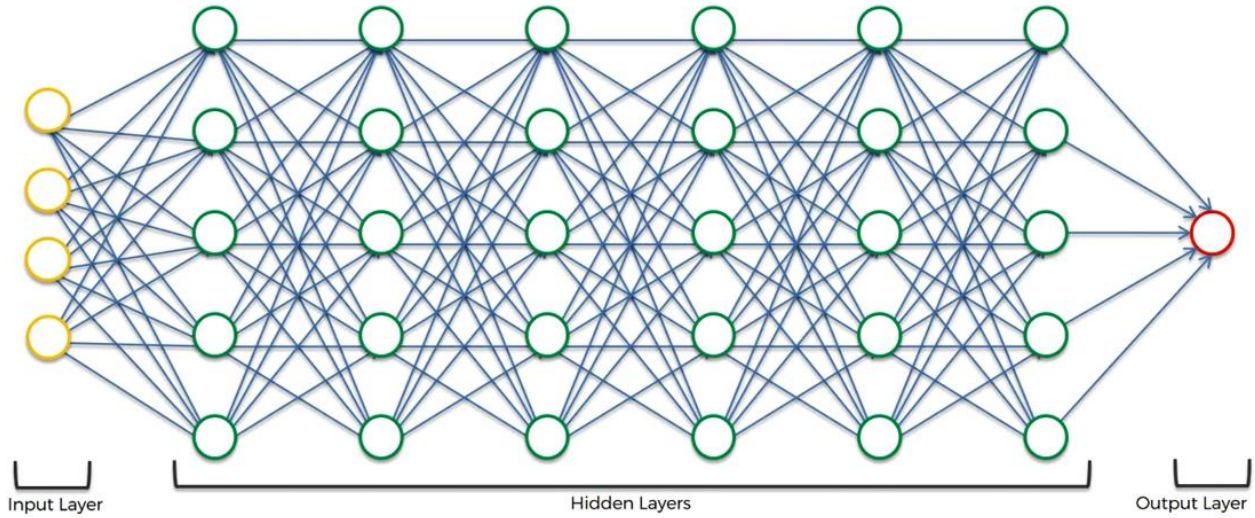
Step3 : There is a concept of Hidden Layers. Let's say you are receiving some of the input from your senses let's say eyes, nose, ears etc.... So these input is not directly going into the output .It's going to pass through billions and billions of Hidden layers. So that's the concept behind the NN we are going to model the brains. That's why we used to say if we increase the Hidden layer our accuracy will increase.





So here now you can say what is deep in this . How is it deep learning. So to answer your question this is not the actual deep learning we can call it as shallow learning.

Deep learning will come with lots and lots of hidden layer.

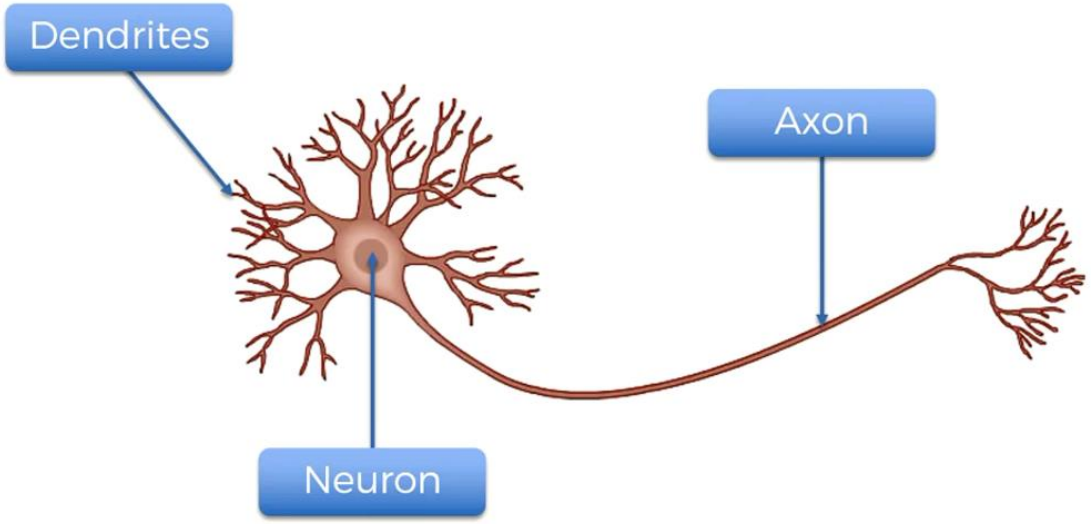


This is how our actual NN looks like.

Topics we are going to discuss :

1. The Neuron
2. The Activation function
3. How do Neural Networks work (with example) ?
4. How do Neural Networks learn?
5. Gradient Descent.
6. Stochastic Gradient Descent
7. Backpropagation

The Neuron



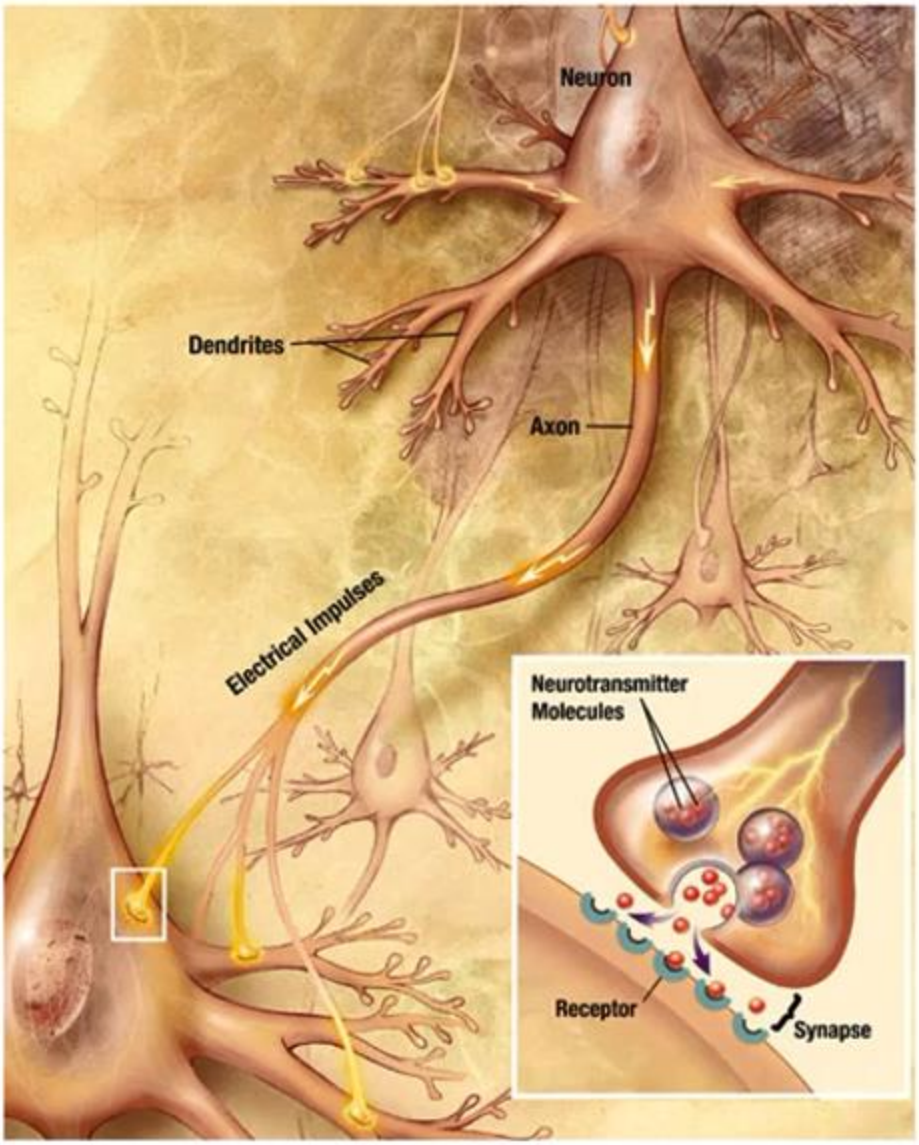
This is how one single neuron look like. If we have single neurons it's not going to do anything but if we can have lots and lots of neurons then this will Definitely do the magic for you.

a.) Dendrites --> Used to collect the information from the senses. like eyes, nose , ears. skin etc....

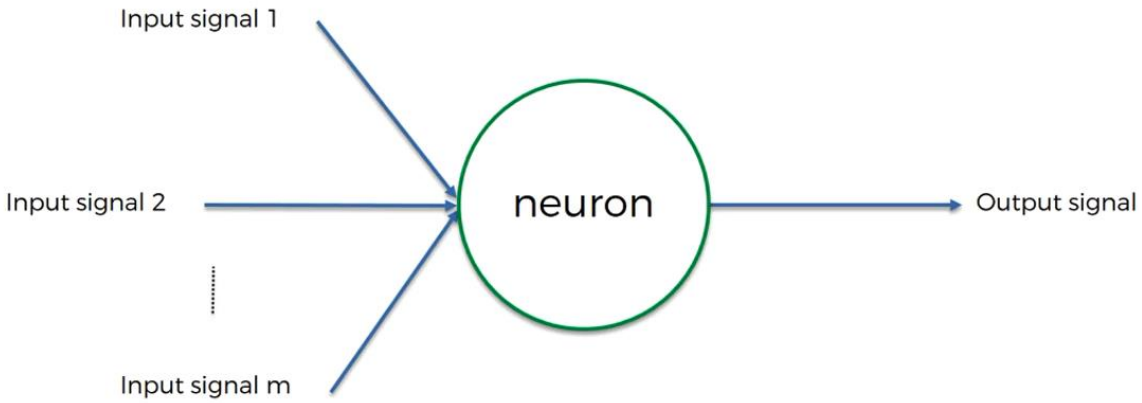
b.) Axon --> Which is used to transmit the data from one neuron to another neuron.

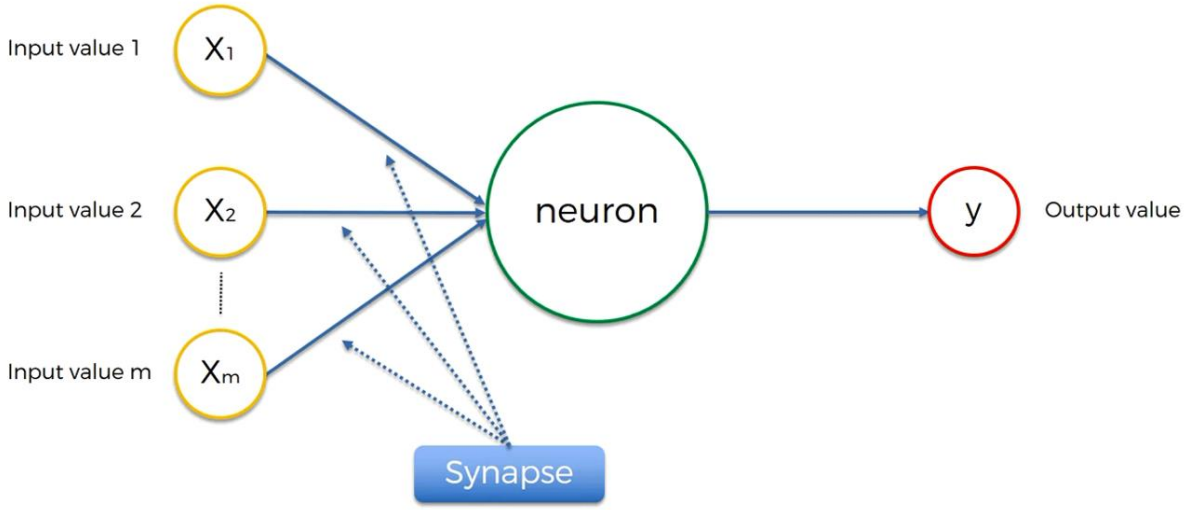
c.) Neuron --> The main body which will process the input.

If you wanted to check this is how actual neuron looks like.....

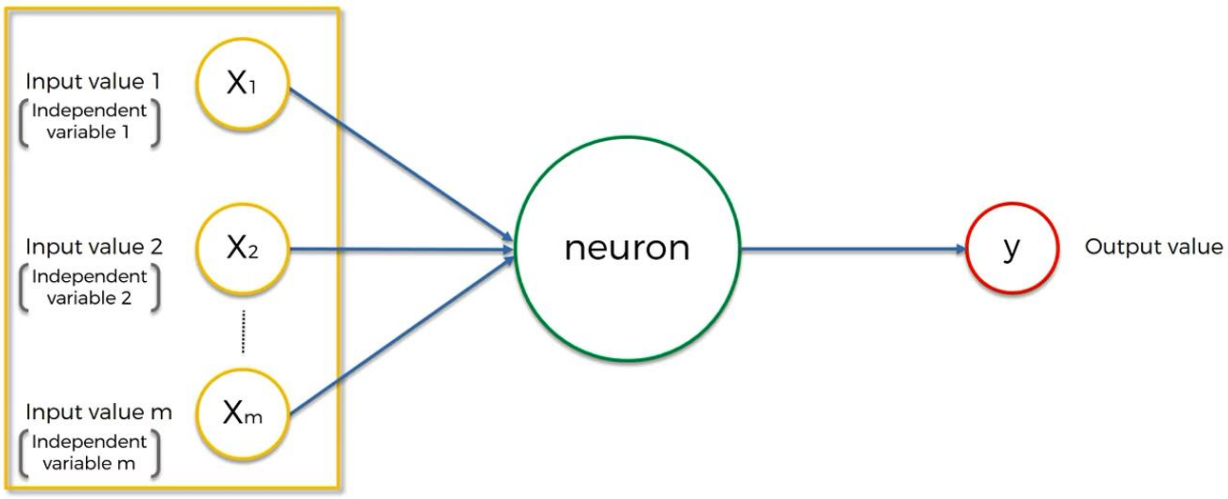


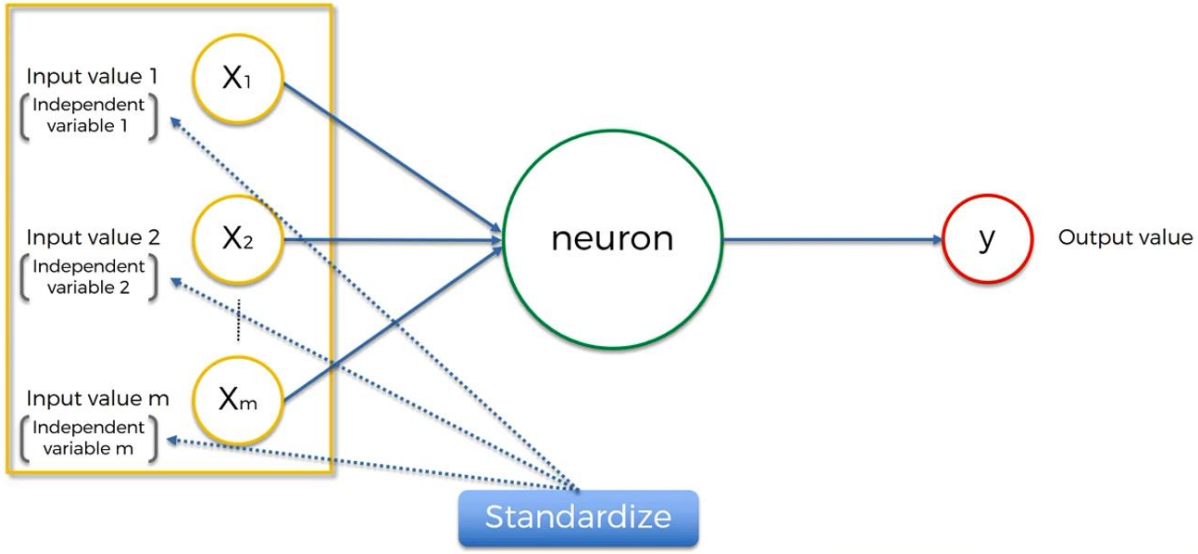
Now here is the actual structure of how our Neurons look like. Some time we also called it as Node.





We can call this input lines as Synapse



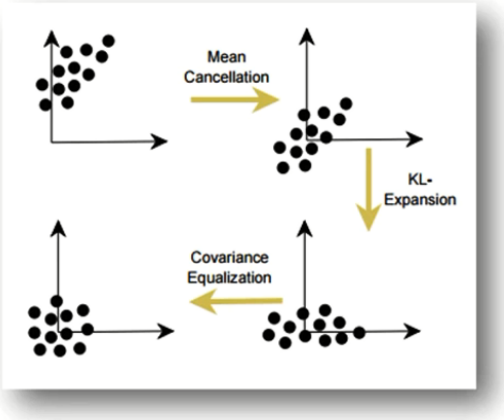


If you wanted to learn more about Standardization , Normalization, Localization here is the the link in the image you can go through it.

Additional Reading:

Efficient BackProp

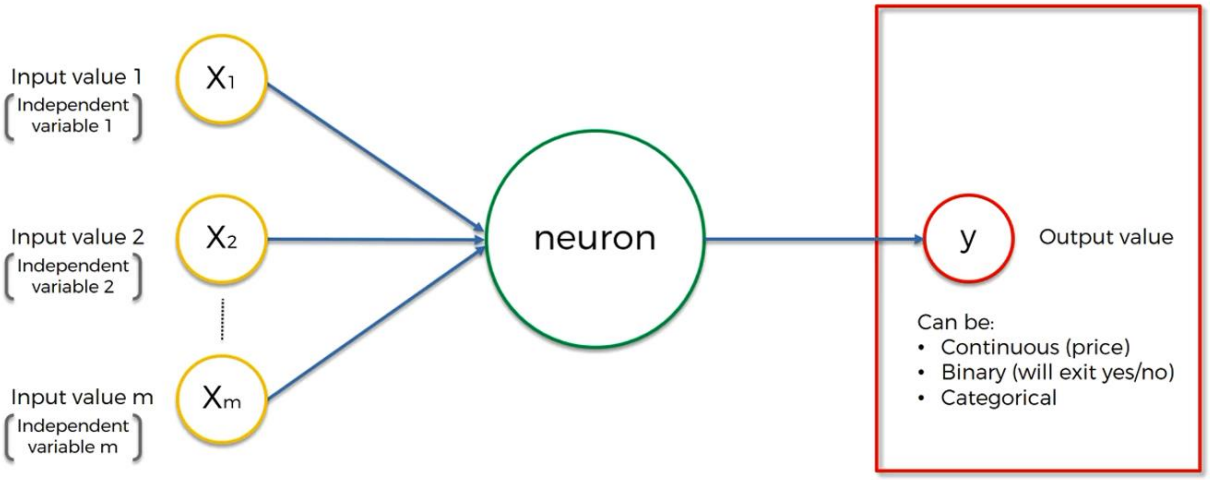
By Yann LeCun et al. (1998)



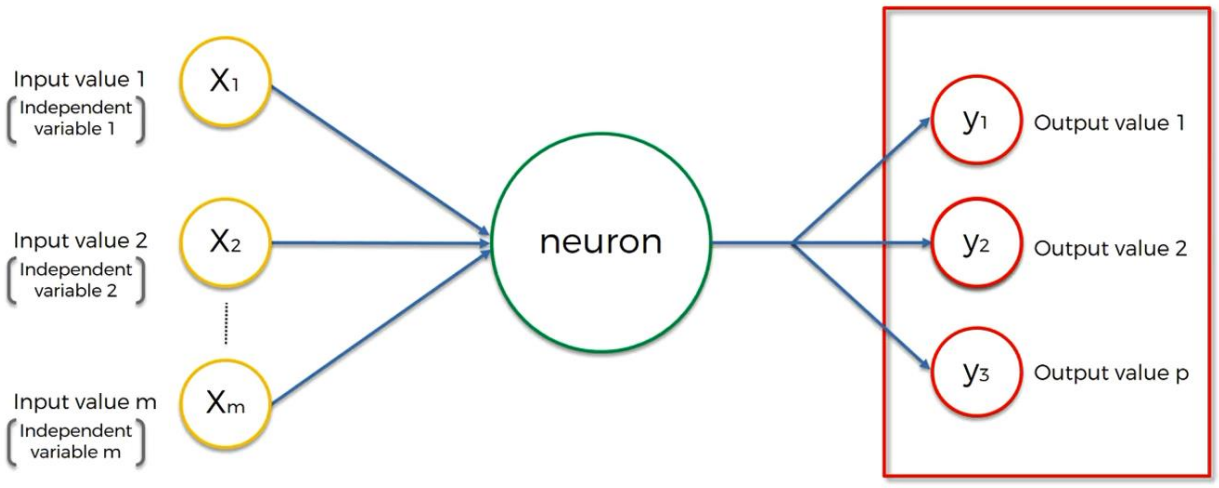
Link:

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

Now the output Value

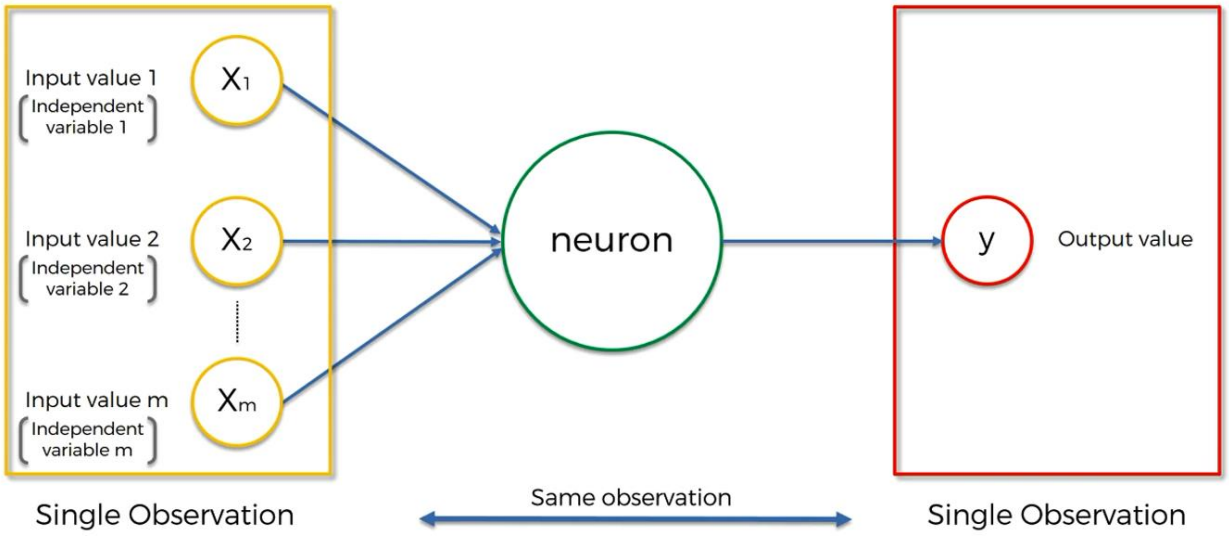


Important point to remember here is if your output is categorical variable then your output value can't be just one. It can be Multiple.

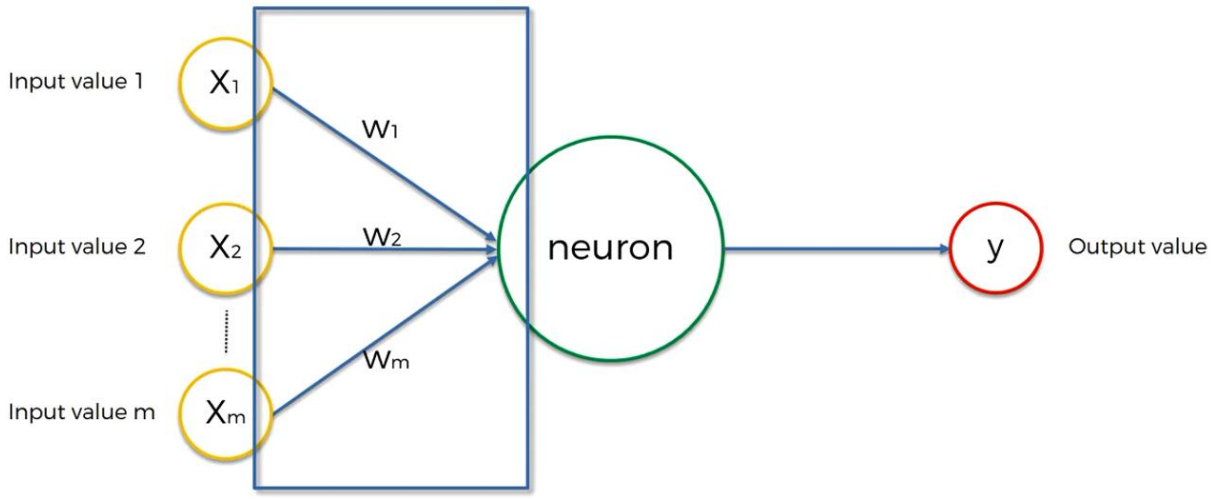


You can say these output value 1,2 3 are just dummy variables which can represent your exact Output values

Important point to remember here is you are always going to provide your one input row and your getting the output for that particular row itself.



Now let's talk about Weights. Weights is the value on which your neural network is going to learn by adjusting them throughout the process. Will talk about the weights again down the road, but for now try to understand this is how it's going to work

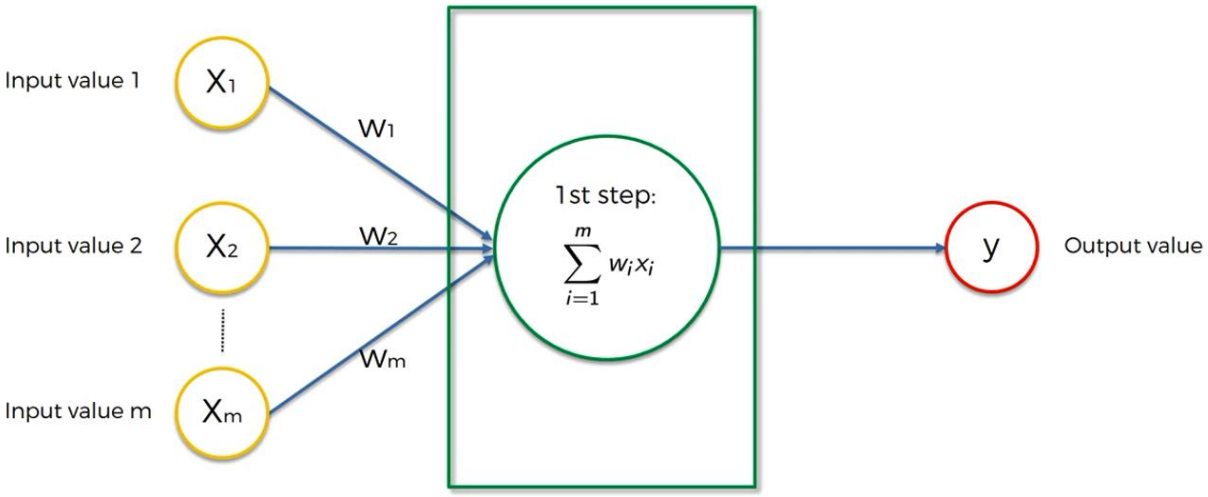


Now the next part is Neurons the signal is going into the neurons. What is happening inside neurons.

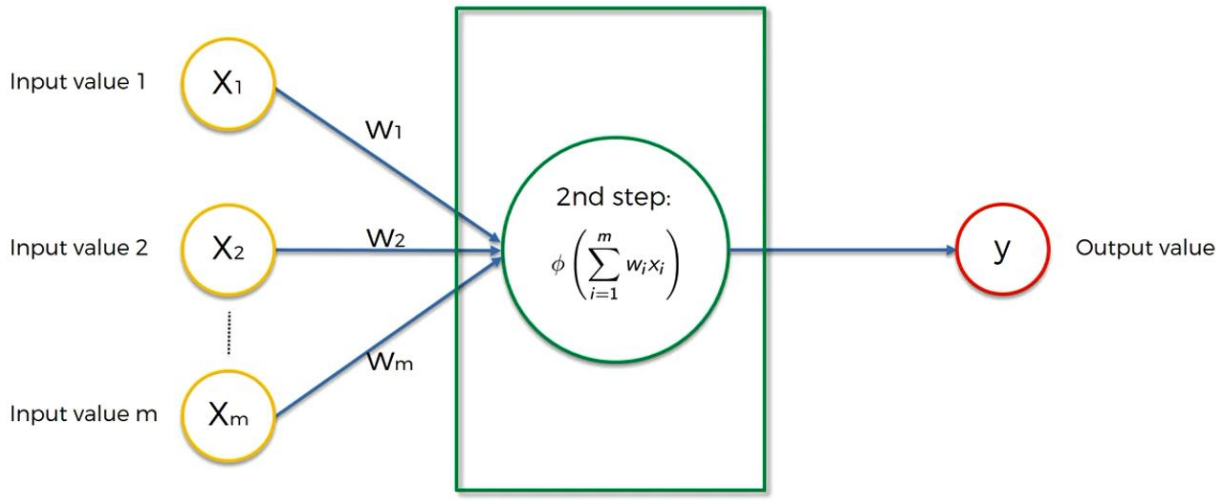
So few steps are happening there .

1. All the input values and weights are going to added up there.

`

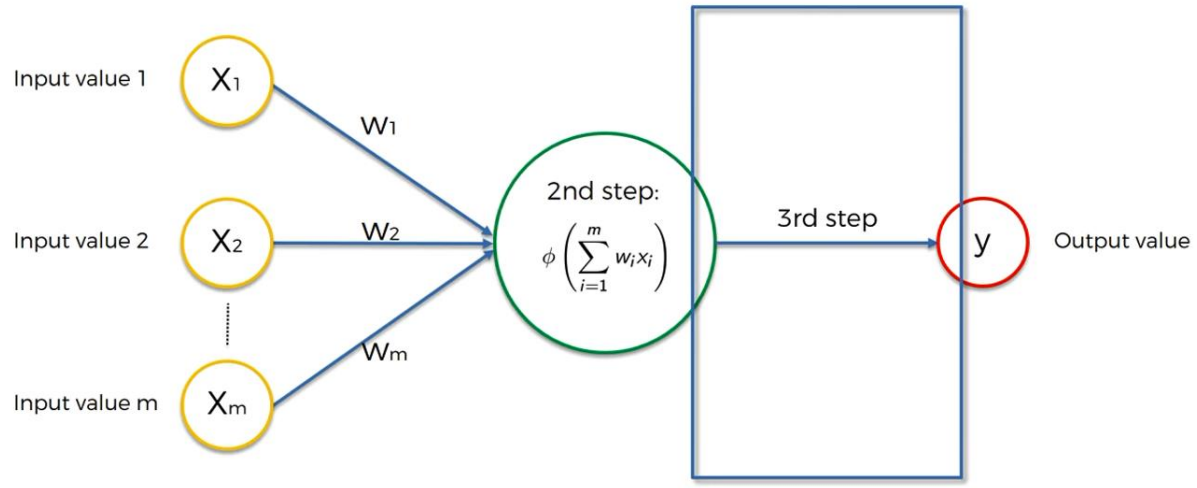


2. Now in the second step it's applied the activation function



Don't worry will talk more about this activation function down the road but for now just try to understand that, on the basis of the output of the activation function the neuron will decide either it's going to pass this input to the next layer or not.

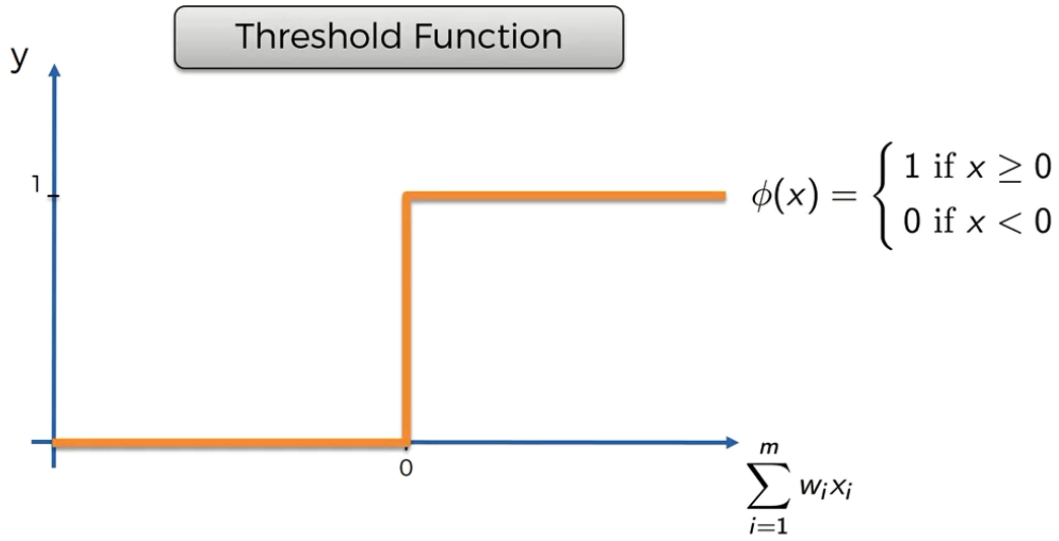
3. Now in the third step the output of that particular neuron is going to pass to the next neuron



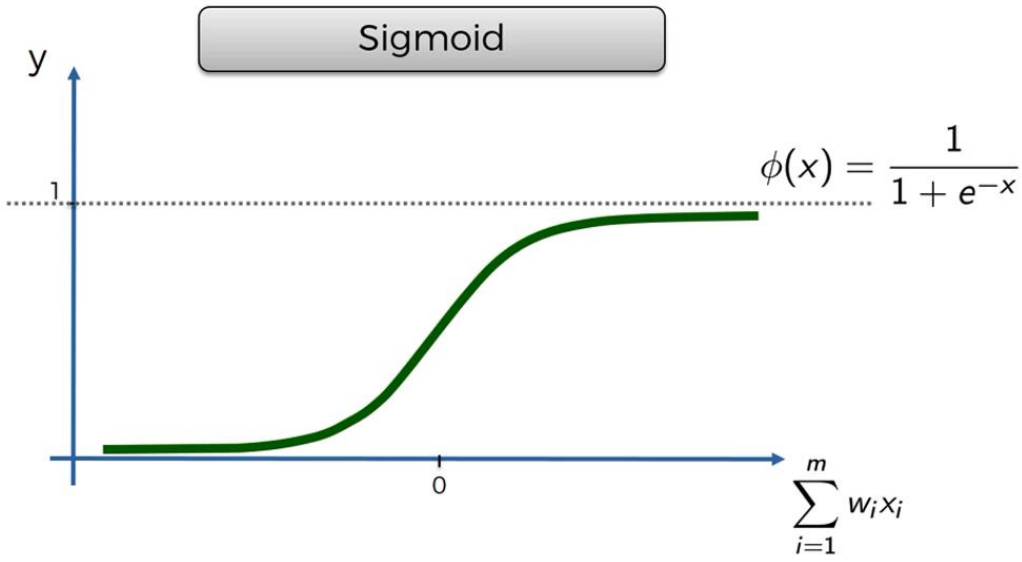
The Activation Function

1. What are all the most famous Activation Functions. There are many activation functions are available but for now we are going to talk about the most popular for activation function.

a. Threshold Function --> So on the axis X we have weighted some of inputs and weights and on y axis we just have values between 0 to 1. This basically a yes/ no kind of function. Basically the way it works is if the value is < 0 then the Threshold function passes on 0 or if the value is >= 0 then it will passes on 1

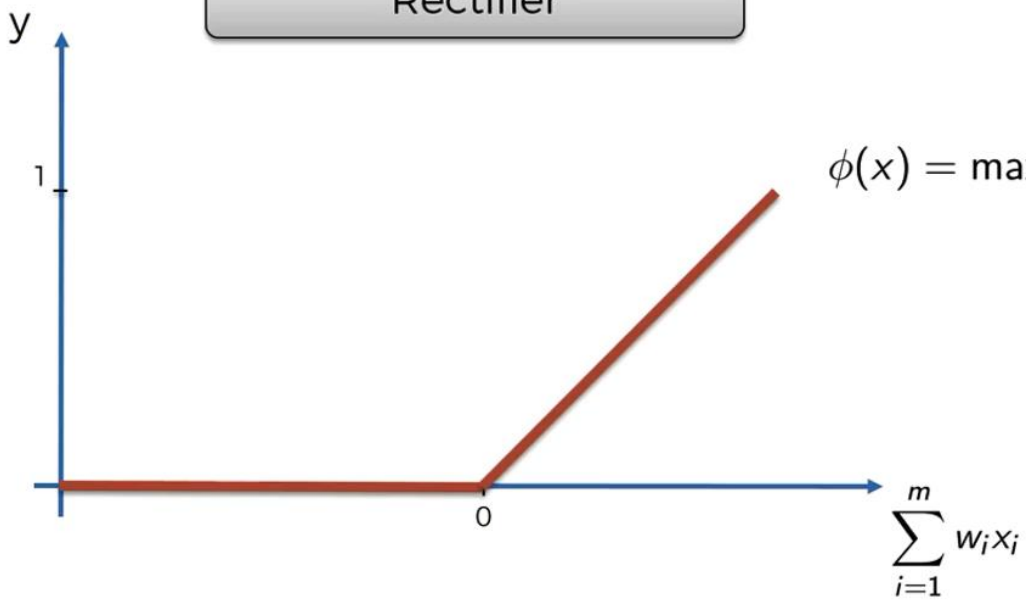


2. Sigmoid Function --> In the equation the value of X = the value of weighted sum. This is the function we also use in Logistic Regression. This sigmoid function is very useful in the output layer specially when you are trying to predict probability's



3. Rectifier Function --> It's one of the most popular function in ANN. Will talk about it later down the road.

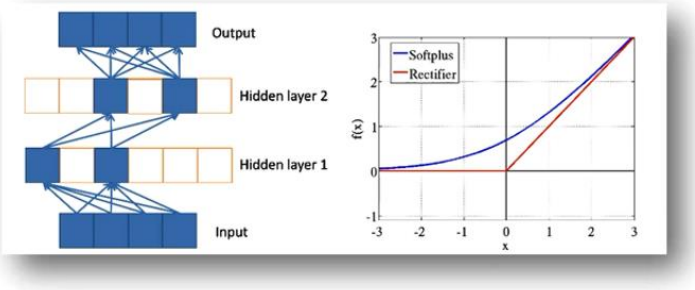
Rectifier



Additional Reading:

Deep sparse rectifier neural networks

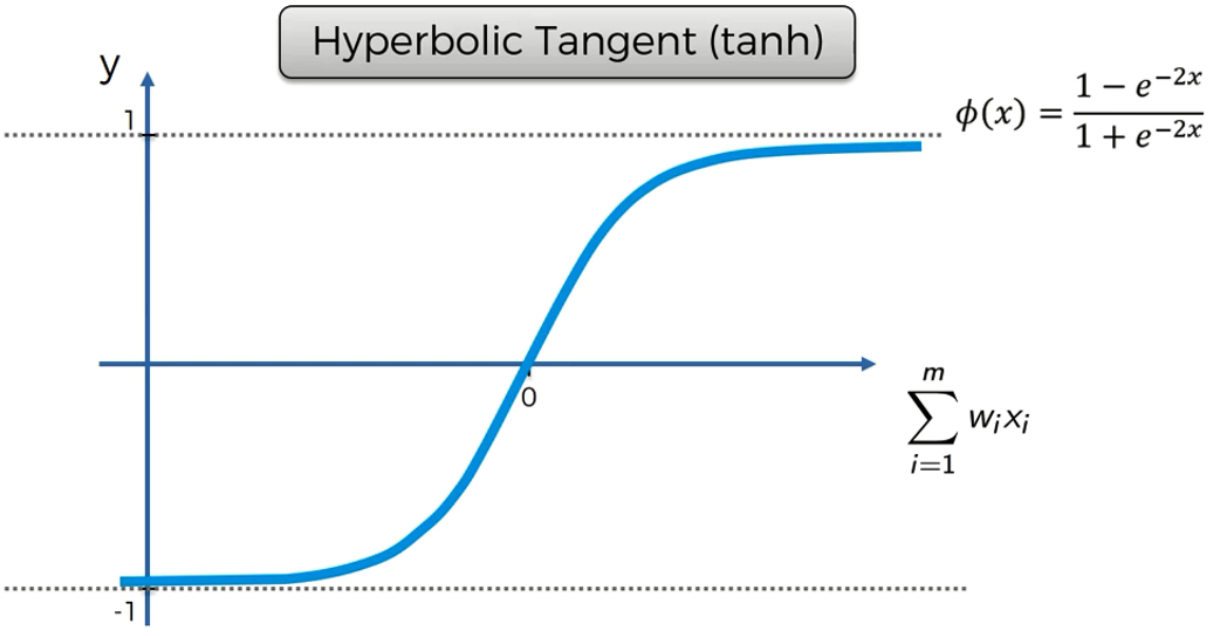
By Xavier Glorot et al. (2011)



Link:

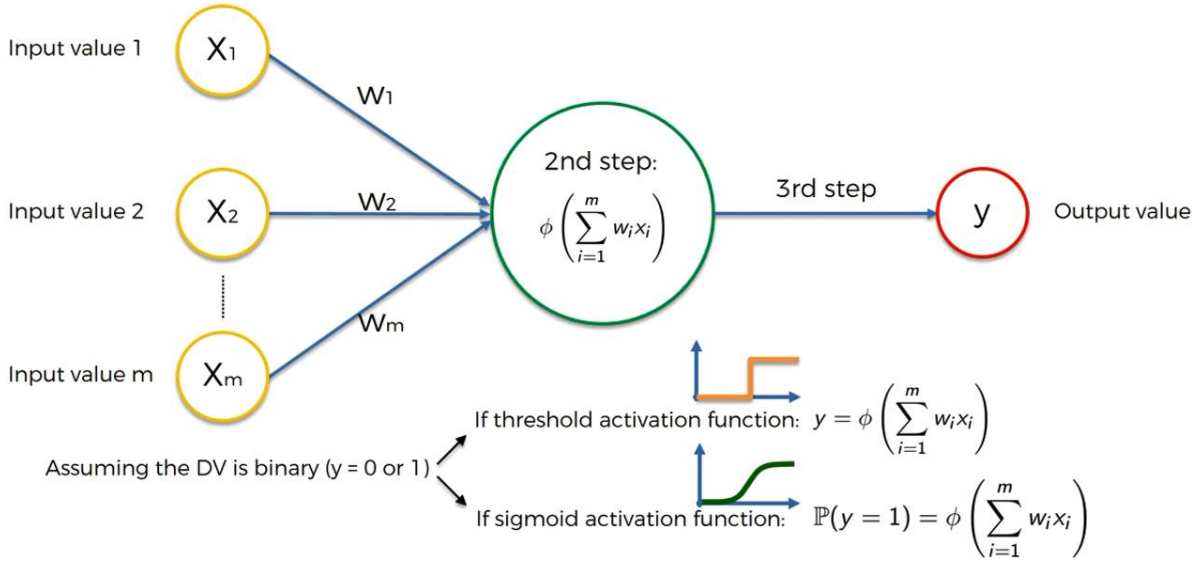
http://jmlr.org/proceedings/papers/v15/glorot11a/glorot11a.pdf

4. Hyperbolic tangent (tanh) --> It's very similar to the sigmoid function. But here tanh goes below 0 as well if you remember from the Sigmoid function it never goes below 0.

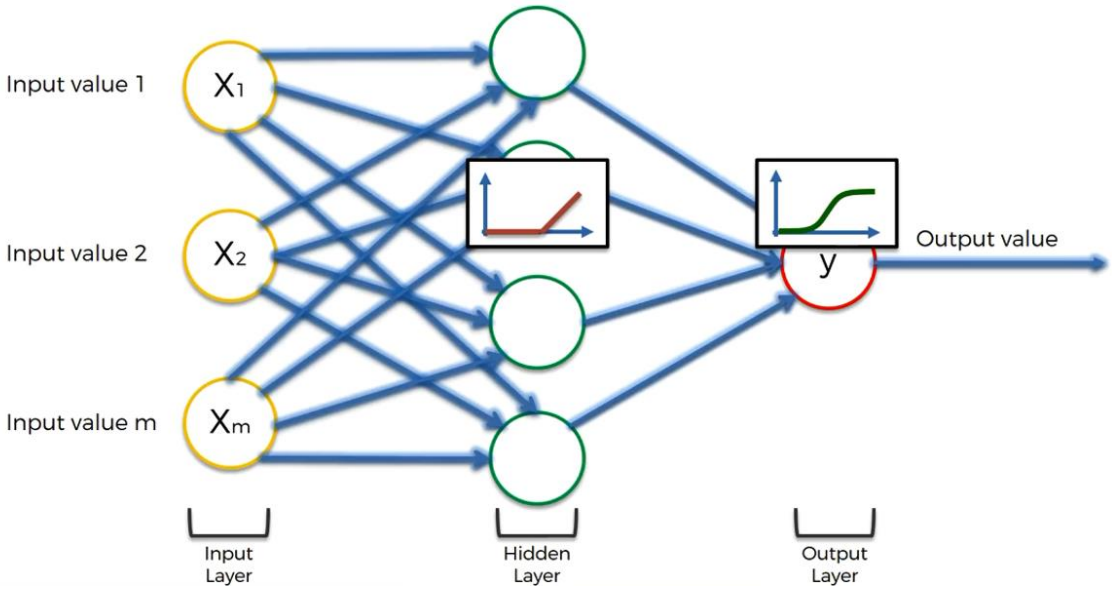


Now let's try to wrap our discussion about NN steps.

Let's in your dataset if your Y value is having binary output, then what all the function which we can use on the basis of the activation functions we have discussed so far.

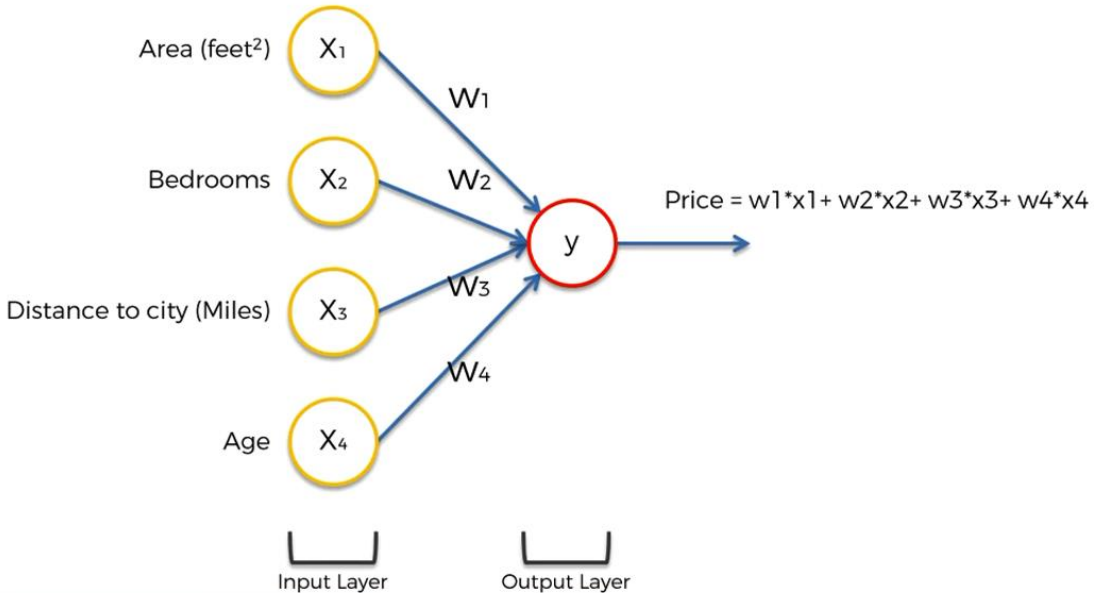


Let's say we have applied the function in this way.



How do NNs work ?

Now we will try to understand this by going through one of the example. In the example what we are going to do is we are going to predict the property value on the basis of different input parameters.

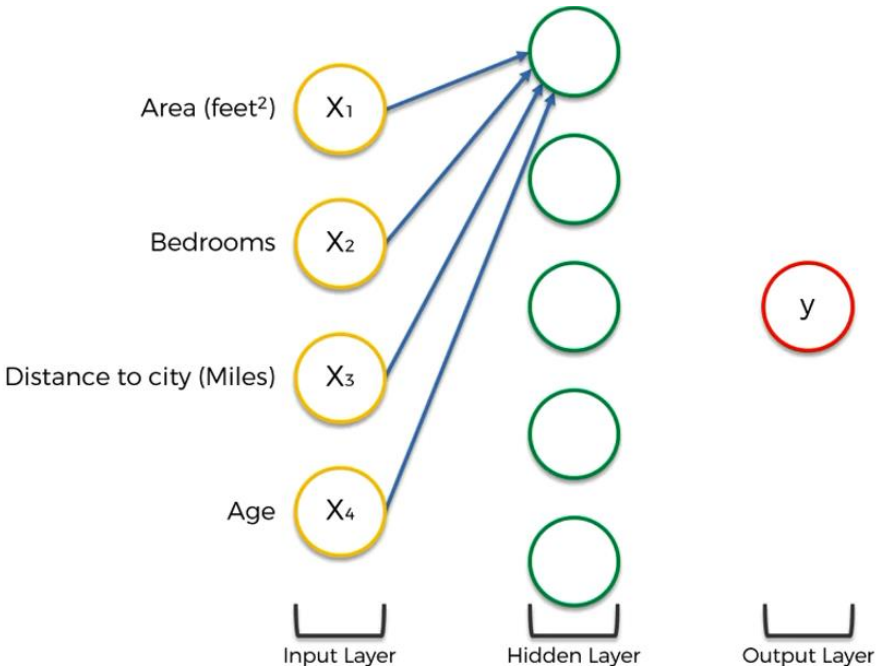


So, In the above example let's say we have four input parameters. We can have more parameters in real case scenario but for simplicity purpose let's try to understand with these parameters only.

Let's try to solve this example with just input layer and output layer there is no hidden layer in between, so in this case the way we are going to calculate our Y value is the same as shown above. And here you can use any activation function which will try to give you the output in continuous manner.

Now let's try to understand how the hidden layer is going to give you the power

Step1 : Let's consider this one input only at a time.

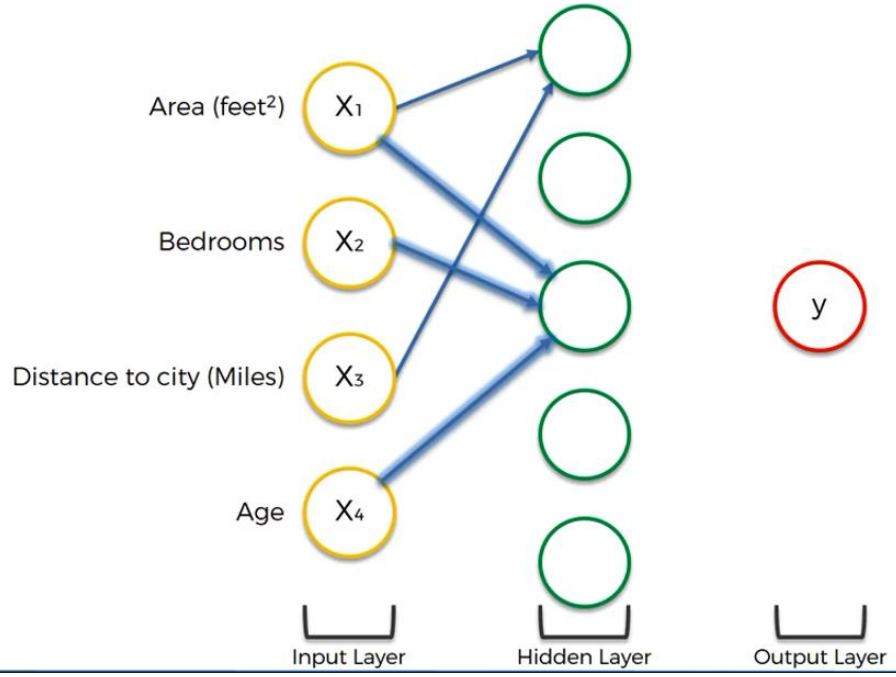


Now as per our discussion so far we already know that all the inputs from input layer should have some weight . Now let's agree that some weights have zero values and some have non zero values. Basically not every single input is important for every neuron sometimes all the inputs is not useful.

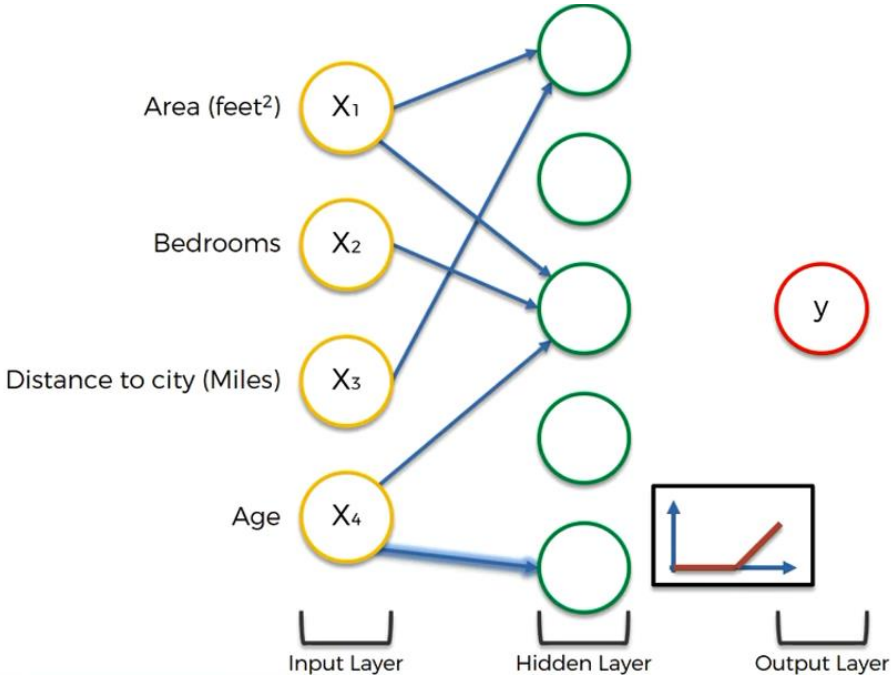
Now let's try to understand it in such a way if someone is looking for some specific properties in their property requirement. Let's say a customer is looking for any property which is not so far from the city and the area of the property should be high. And hypothetically we are assuming that this particular neuron is looking for only those properties in the input so in this case other two parameters for this particular neuron is not important.

so in this hypothetical situation our network will look like something

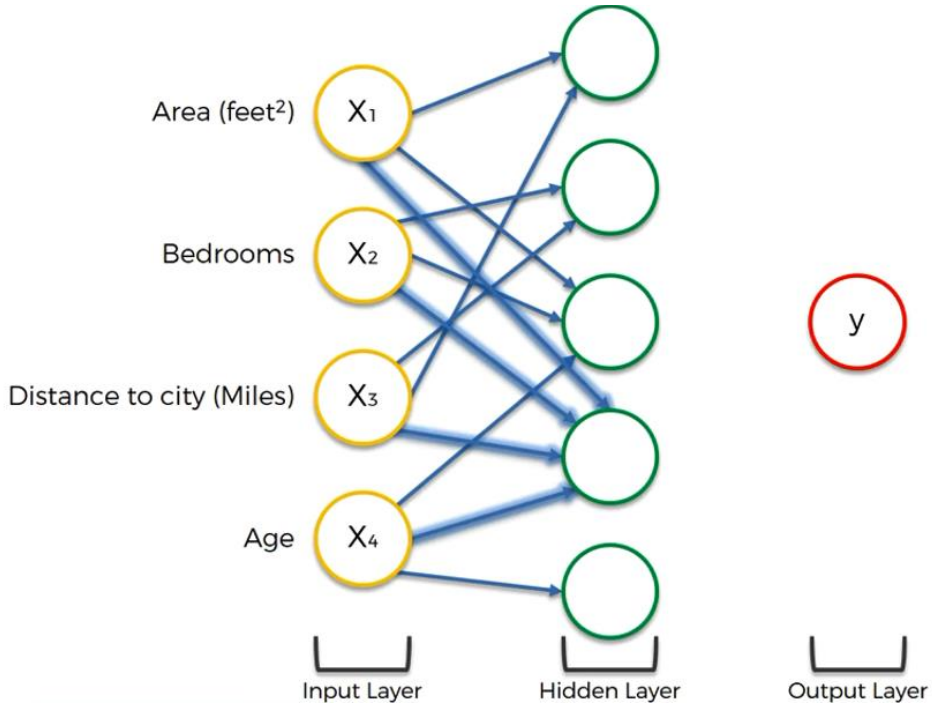
Now let's assume new hypothetical situation for this neuron as well

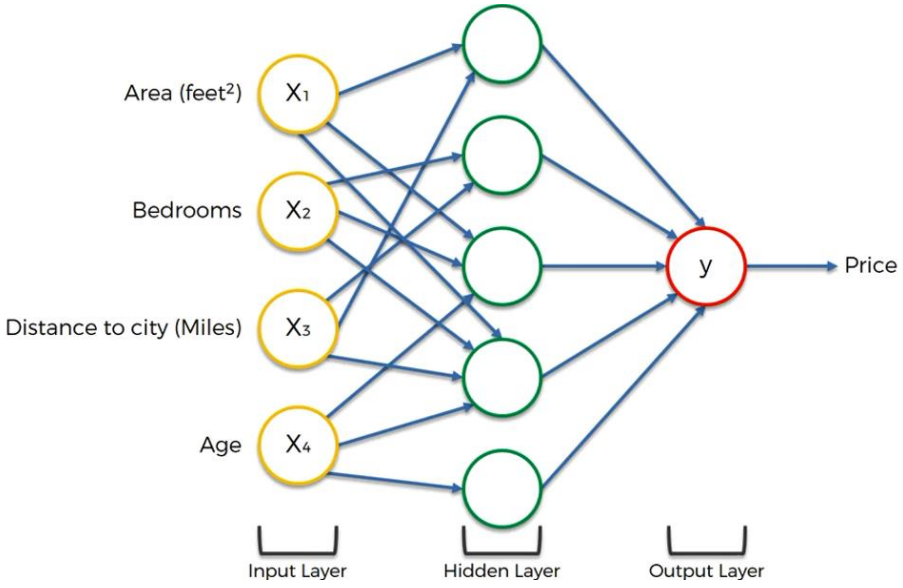


Now let's assume new hypothetical situation for this neuron as well



Now let's assume new hypothetical situation for this neuron as well



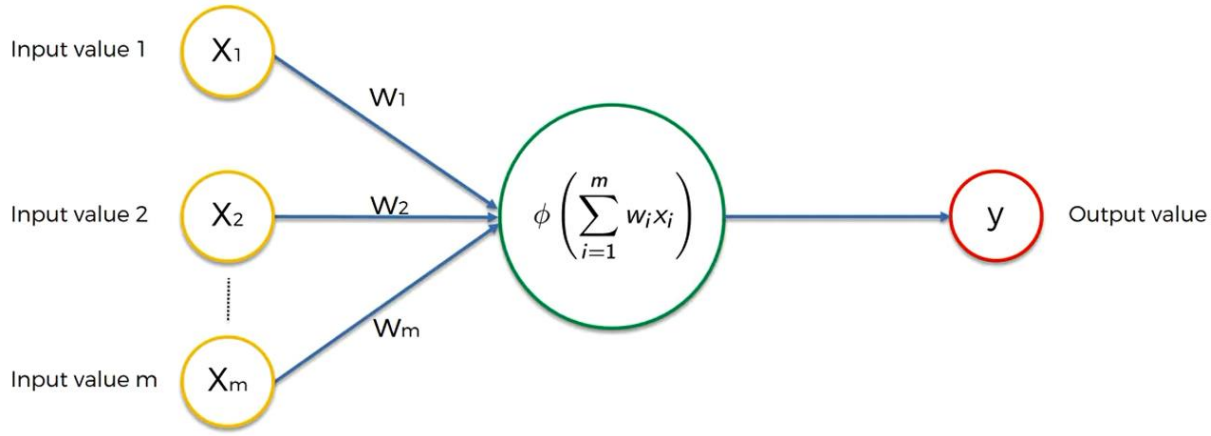


How do NN learns?

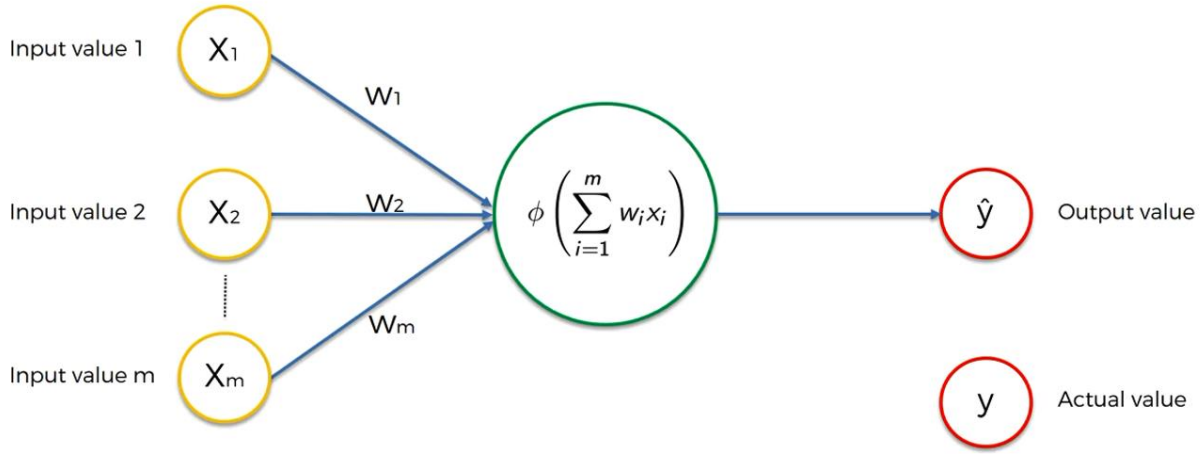
There are two fundamentally different approaches to getting a program to do what you want to do.

1. Hard coded coding.

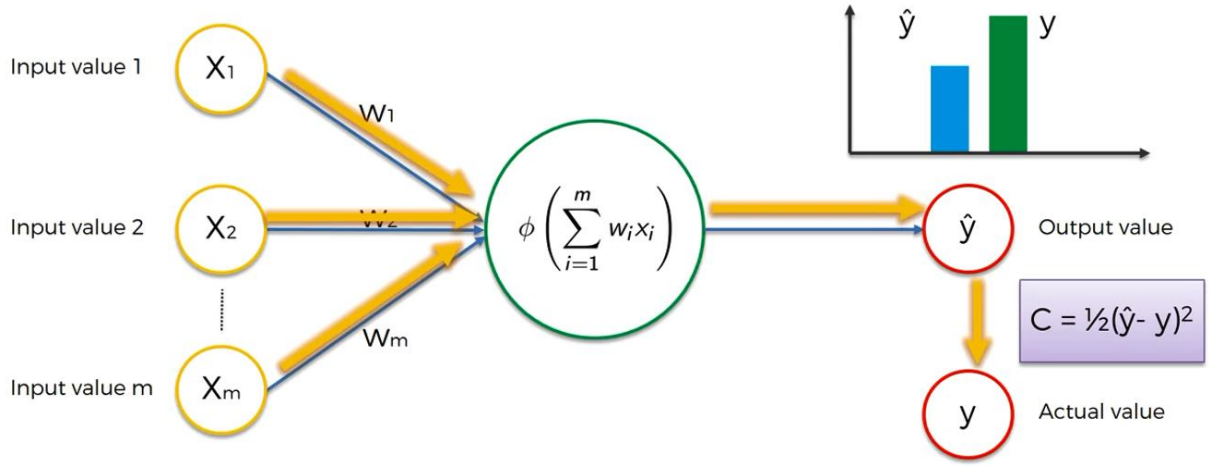
2. NN will try to learn all the output on it's own.

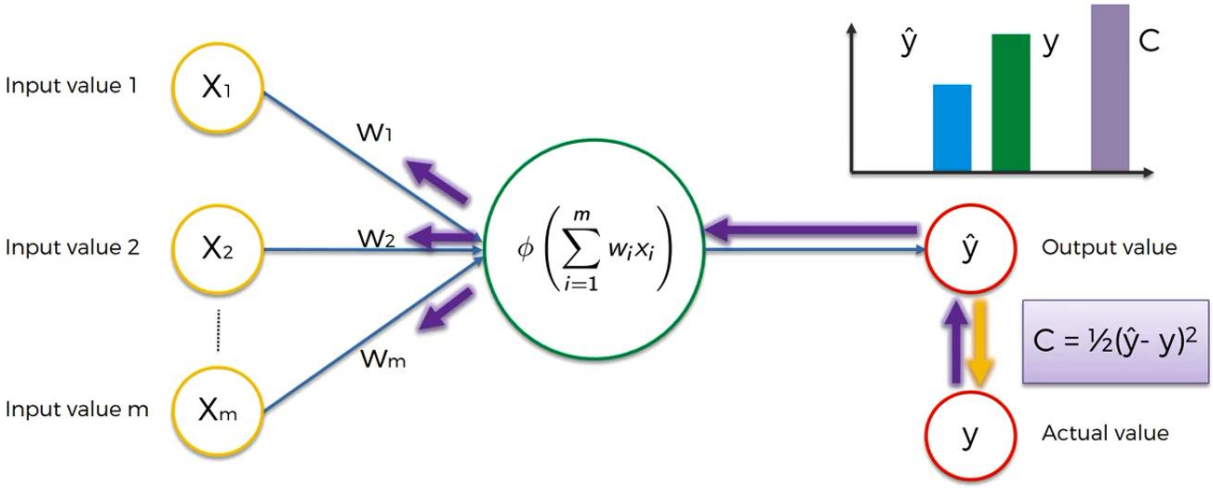


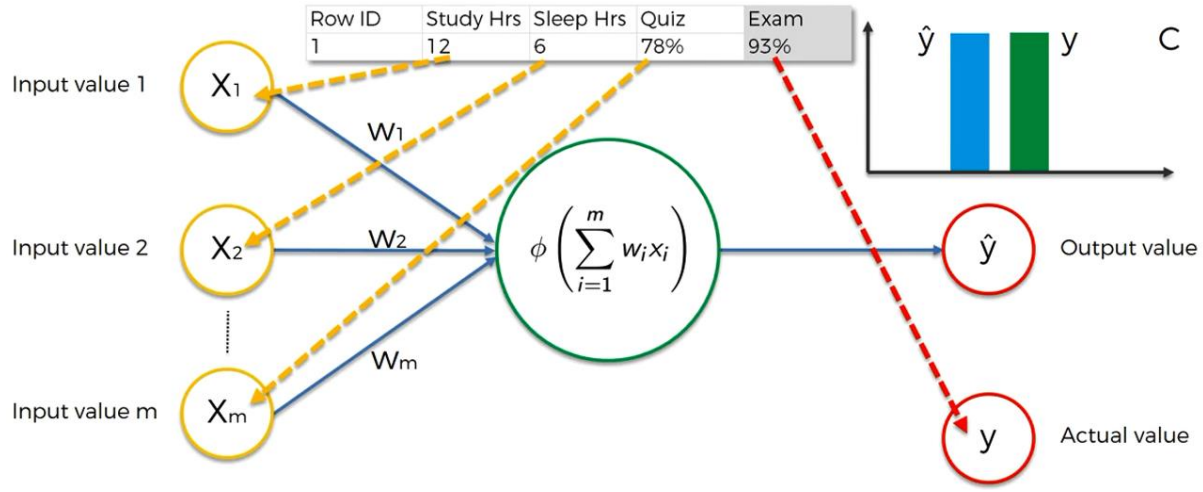
Here we have a very basic Neural Network with one layer this is called single network feed forward network,, we also called it as Perceptron. The preceptron was first invented in 1957 by Frank Rosen Blood. And his whole idea was to create something that can actually learn and adjust itself.

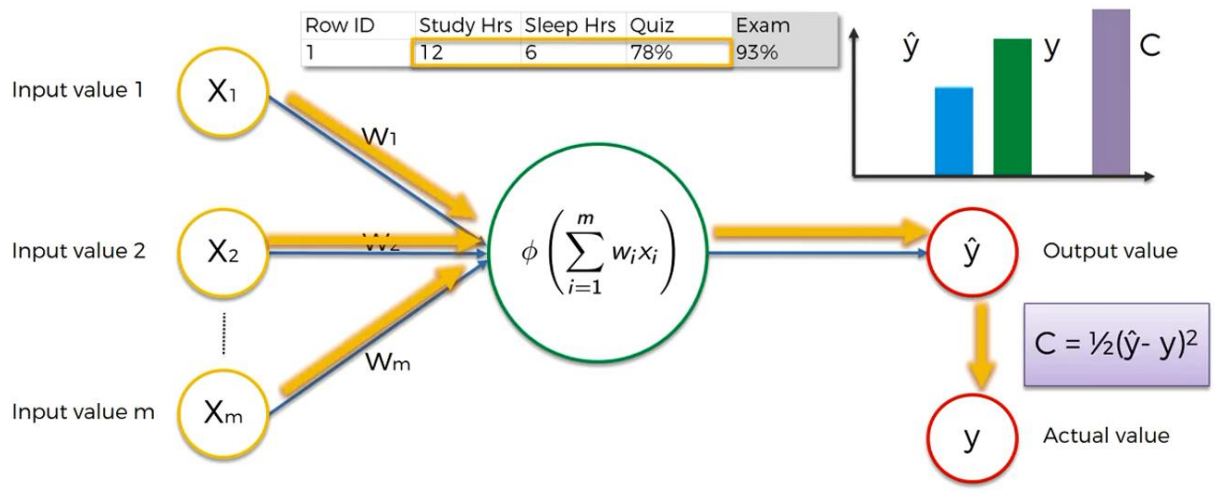


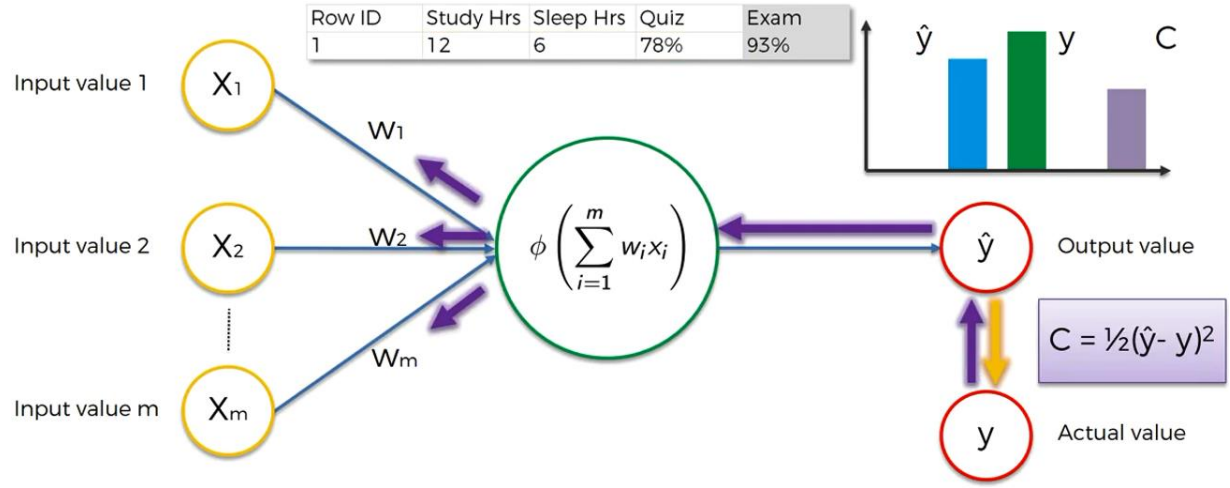
Now let's try to understand this how NNs actually learns. Let's say we have three inputs and one hidden layer and we applied some activation function inside the hidden layer neuron. and we got the some predicted value called Y hat.. let's plot the graph for predicted one and the actual one. Now we can see the difference between the actual one and the predicted one. Now we are going to create one function called as cost function. There are many different cost functions are available that we can use but for now just try to understand it with the basic one

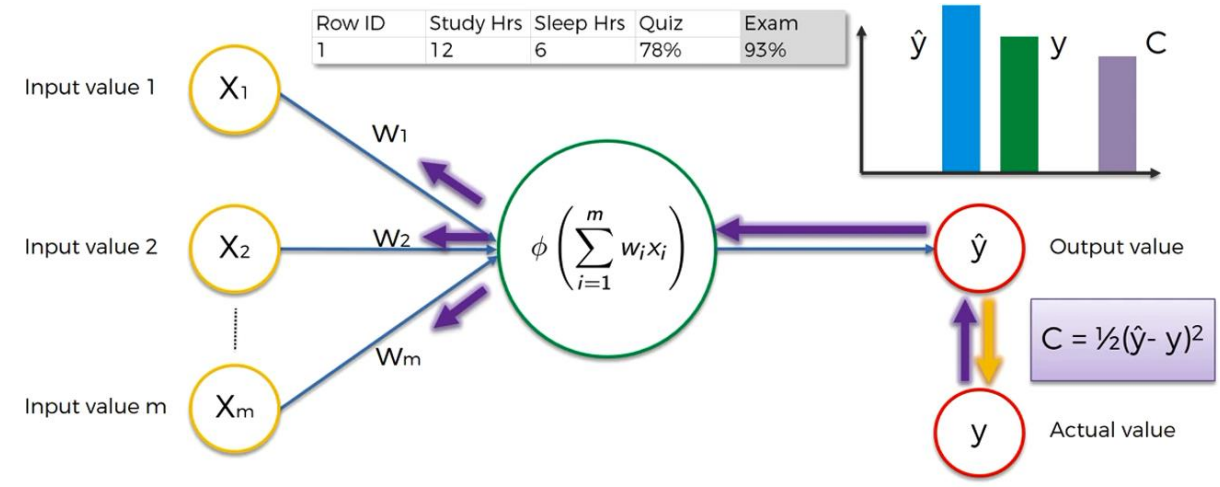

Input value 1
X1
W1
Input value 2
X2
Wm
Input value m
Xm
$\phi\left(\sum_{i=1}^{m} w_i x_i\right)$
ŷ
y
ŷ
Output value
C = ½(ŷ- y)2
y
Actual value

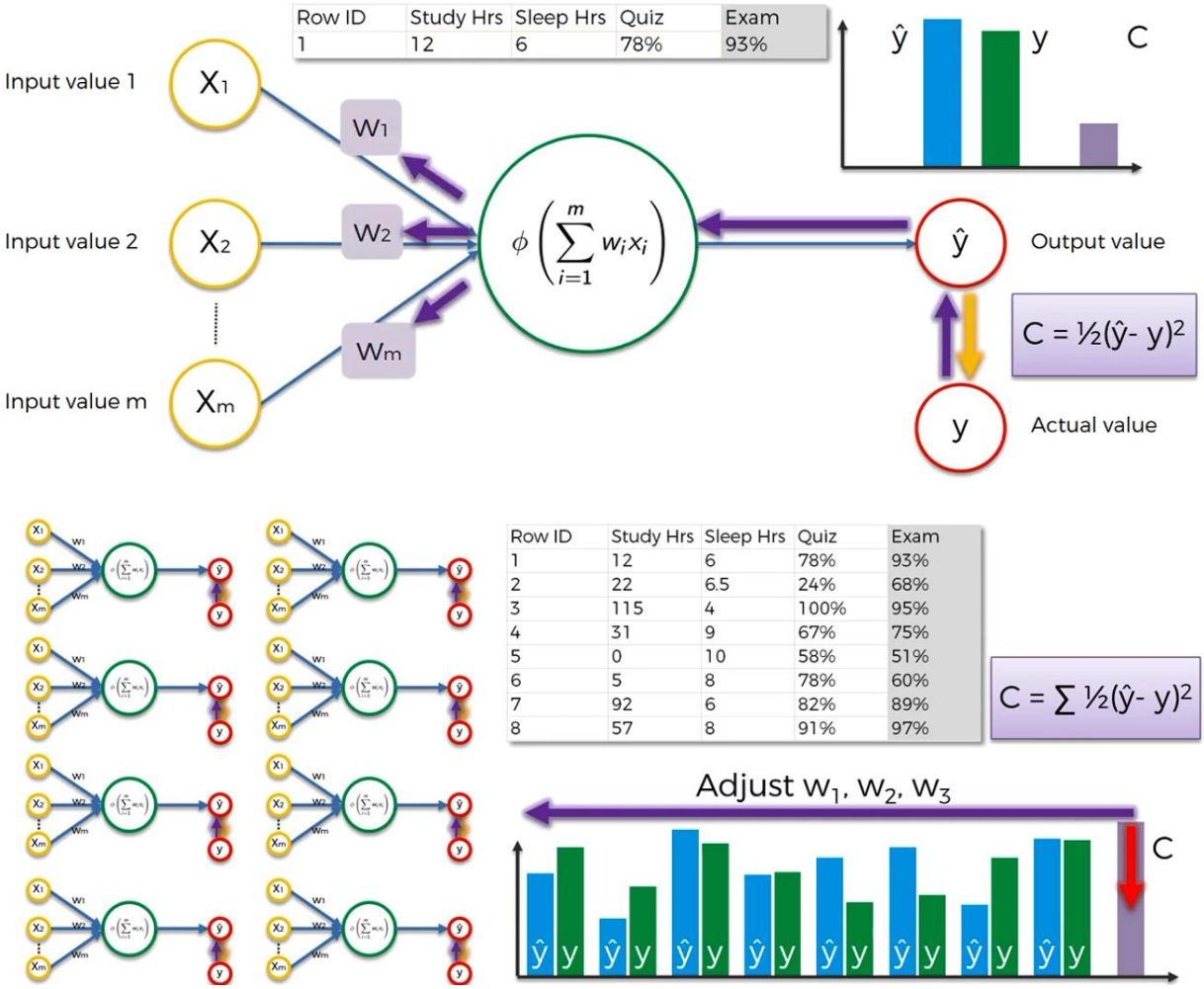

Input value 1
X1
W1
Input value 2
X2
W2
Wm
Input value m
Xm
$\phi\left(\sum_{i=1}^{m} w_i x_i\right)$
ŷ
y
C
ŷ
Output value
C = ½(ŷ- y)2
y
Actual value


Row ID
Study Hrs
Sleep Hrs
Quiz
Exam
1
12
6
78%
93%
Input value 1
X1
W1
Input value 2
X2
W2
Wm
Input value m
Xm
$\phi\left(\sum_{i=1}^{m} w_i x_i\right)$
ŷ
y
C
ŷ
Output value
y
Actual value


Row ID | Study Hrs | Sleep Hrs | Quiz | Exam
1 | 12 | 6 | 78% | 93%
ŷ
y
C
Input value 1
X1
W1
Input value 2
X2
Wm
Input value m
Xm
$\phi\left(\sum_{i=1}^{m} w_i x_i\right)$
ŷ
Output value
C = ½(ŷ- y)2
y
Actual value


Row ID | Study Hrs | Sleep Hrs | Quiz | Exam
1 | 12 | 6 | 78% | 93%
ŷ
y
C
Input value 1
X1
W1
Input value 2
X2
W2
Input value m
Xm
Wm
$\phi\left(\sum_{i=1}^{m} w_i x_i\right)$
ŷ
Output value
C = ½(ŷ- y)2
y
Actual value


Row ID | Study Hrs | Sleep Hrs | Quiz | Exam
1 | 12 | 6 | 78% | 93%
ŷ
y
C
Input value 1
X1
W1
Input value 2
X2
W2
Input value m
Xm
Wm
$\phi\left(\sum_{i=1}^{m} w_i x_i\right)$
ŷ
Output value
C = ½(ŷ- y)2
y
Actual value



all this process is called as BackPropagation

Additional Reading about Cost Functions

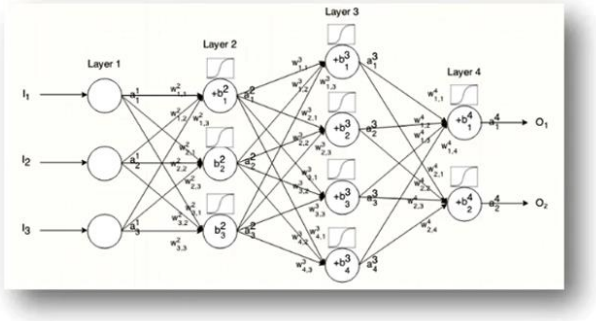
Additional Reading:

A list of cost functions used in neural networks, alongside applications

CrossValidated (2015)

Link:

http://stats.stackexchange.com/questions/154879/a-list-of-cost-functions-used-in-neural-networks-alongside-applications

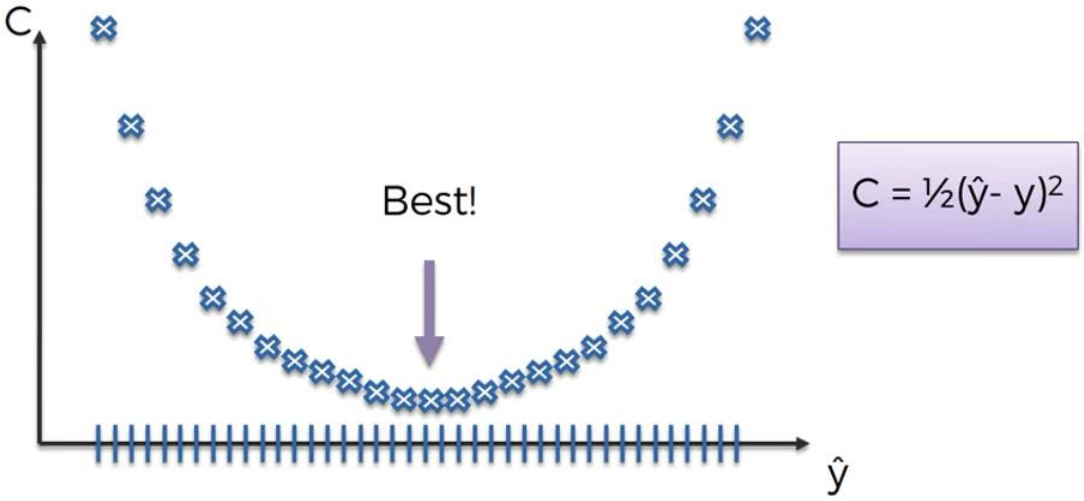


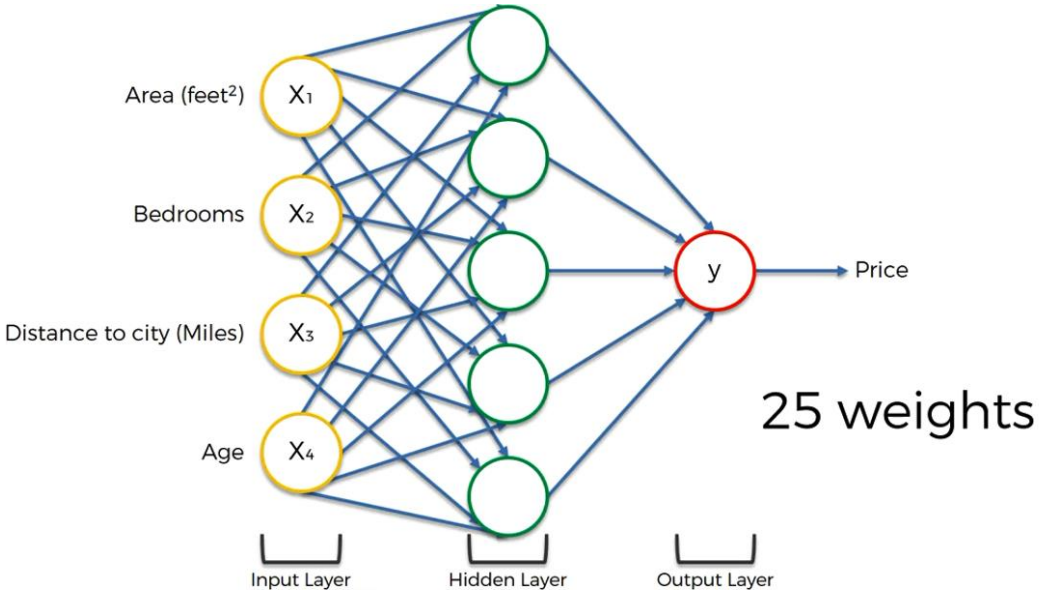
Gradient Descent

It's a cost function which is been used to minimize the cost/loss.

One approach is Brut Force approach --> In this approach what we do is we collect some weights let's say 1000 weights and their corresponding losses and then plot a graph and we can choose which one is best.

But if we use this approach then it might work with one or two column but in NN we have multiple columns so this approach is not going to work. We face the problem which is called as Curse of Dimensionality.





How we can brut force the neural network of this size?

in above example we have 25 weights and let's say we have 1000 combination, Which we want to test out for every weight then we might end up having

1000 x 1000 x .... x 1000 = 100 pow 75 == 10 pow 25 different combinations

PFLOPS = Peta Float Operations per second

1,000 x 1,000 x … x 1,000 = $1{,}000^{25} = 10^{75}$ combinations

Sunway TaihuLight: World's fastest Super Computer

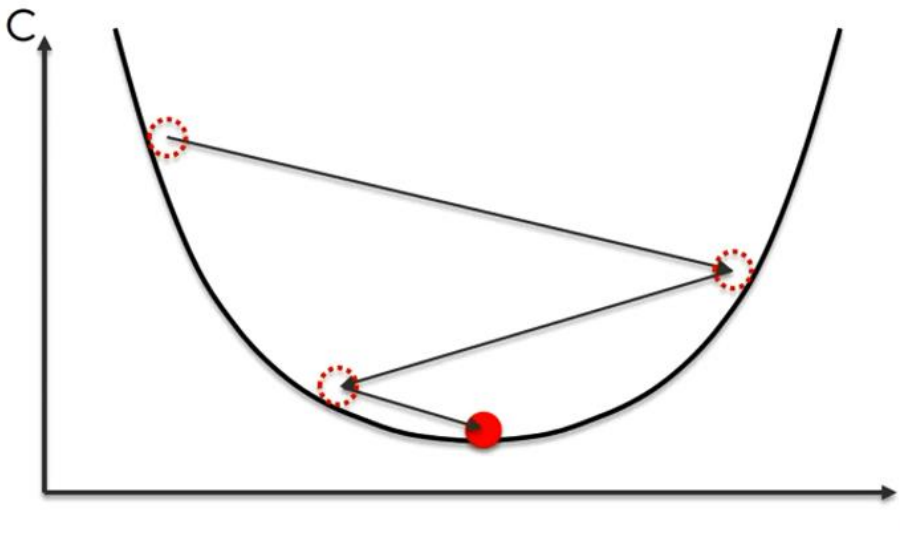
93 PFLOPS

93×10^{15}

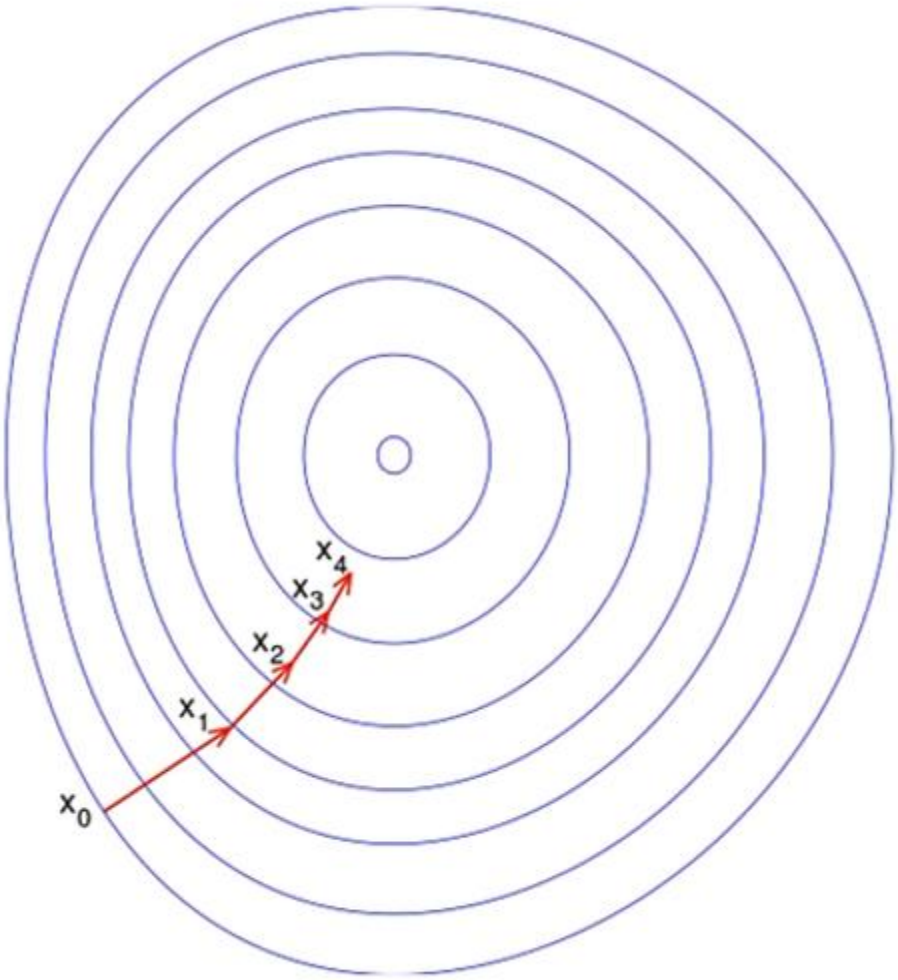
$10^{75} / (93 \times 10^{15})$

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

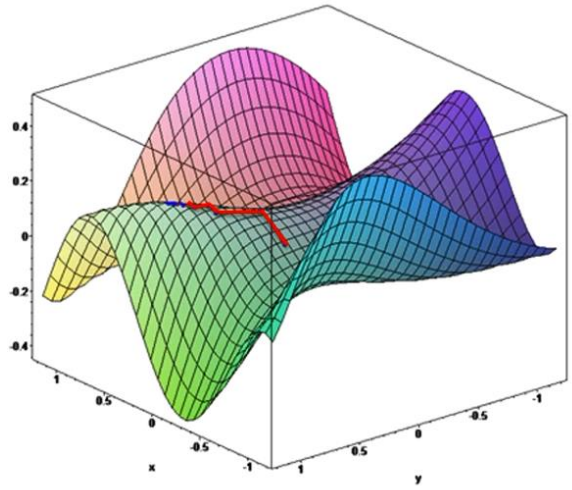
$= 3.42 \times 10^{50}$ years

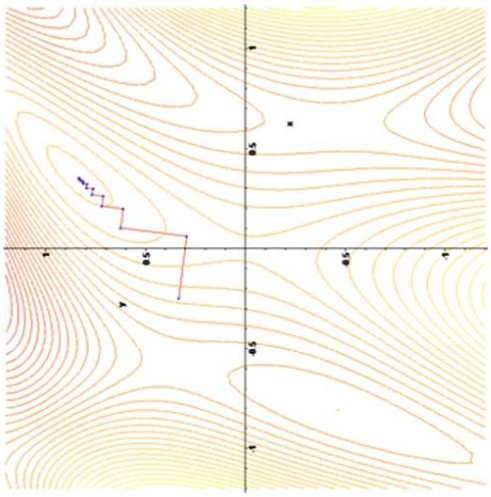


$$C = \tfrac{1}{2}(\hat{y} - y)^2$$



Gradient Descent in 2-d





Gradient Descent in 1-d In 2-d

Let's start by using GD to find the intercept and once we are comfortable with this will try to explore both.

Let's talk about simple regression

y = mx + c m = slope , c = intercept

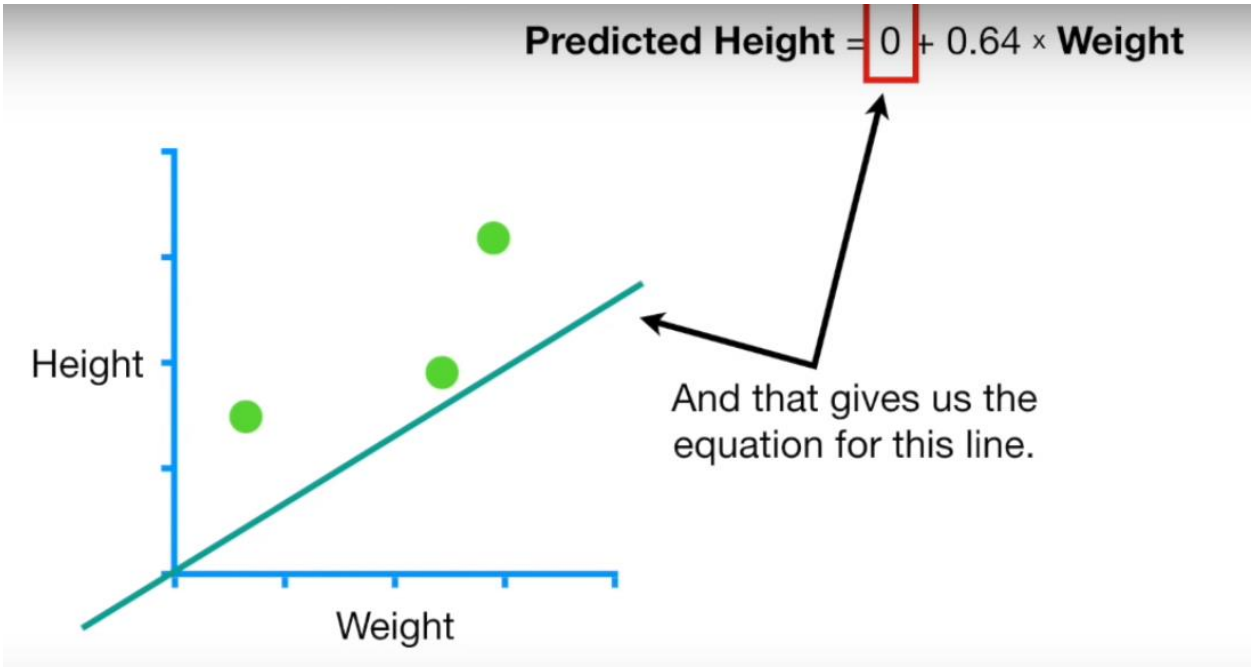
First thing we do is pick a random value for the intercept. This is just a random guess.

let's rewrite the equation

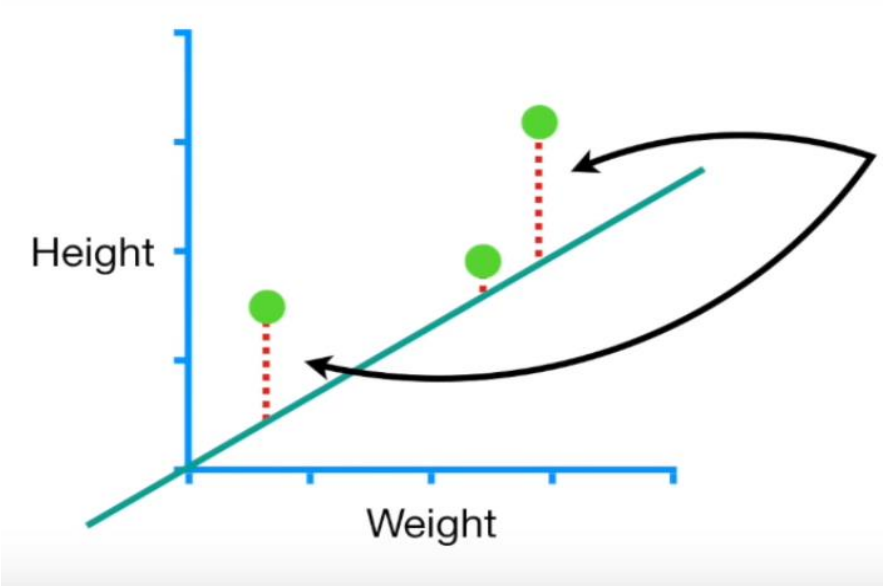
y = 0.64*x + c ==>

In this case we are using 0 for intercept but you can use any number, just to start with

y = .64*x + 0 .....



In this example we will evaluate how well this line fits the data with the **sum of the squared Residuals (it's a type of loss function)**



Let's say the first data point represents the person with (0.5, 1.4).

Now we can get the predicted height by **Predicted Height = 0 + 0.64 * 0.5 = 0.32**

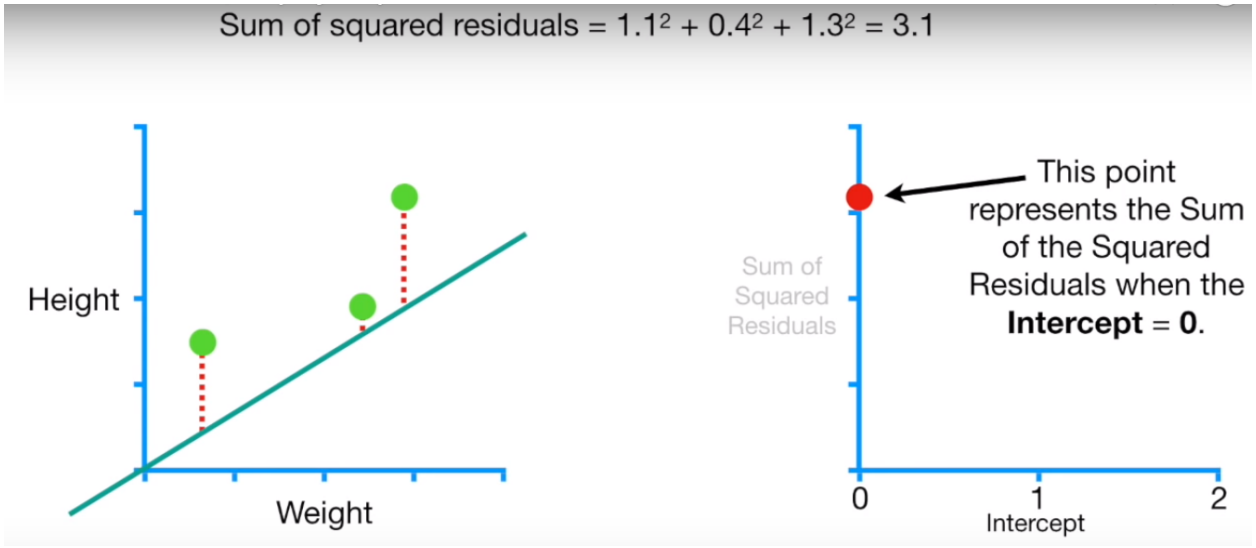
The residual is the difference between the **observed height** and the **predicted height**

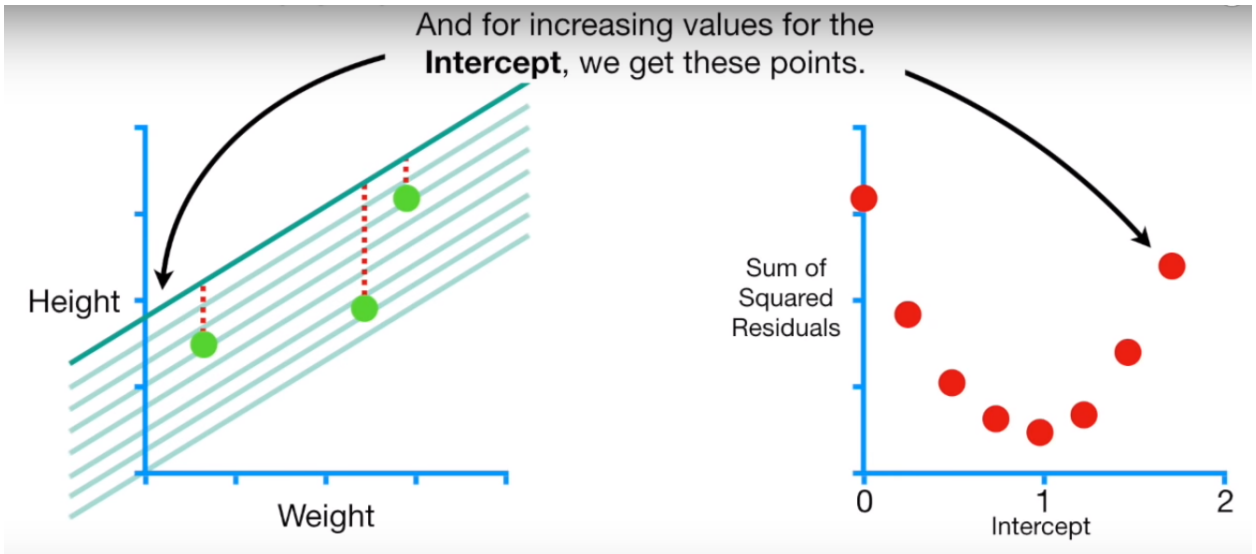
Residual = Observer Height - Predicted Height = 1.4 - 0.32 = 1.1

Sum of the squared residuals r2 = 1.1 pow2 + .4 pow2 + 1.3 pow2 = 3.1

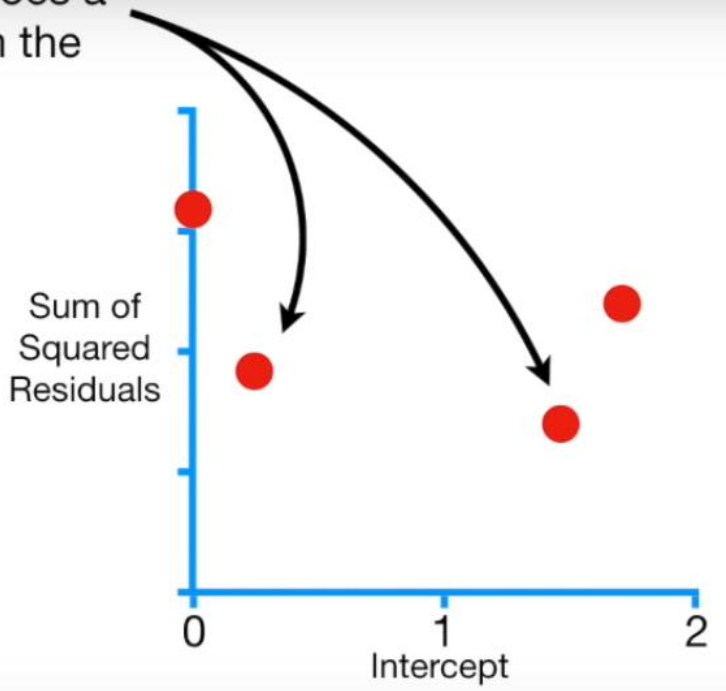
second data point represents the person with (2.3, 1.9).

third data point represents the person with (2.9, 1.2).

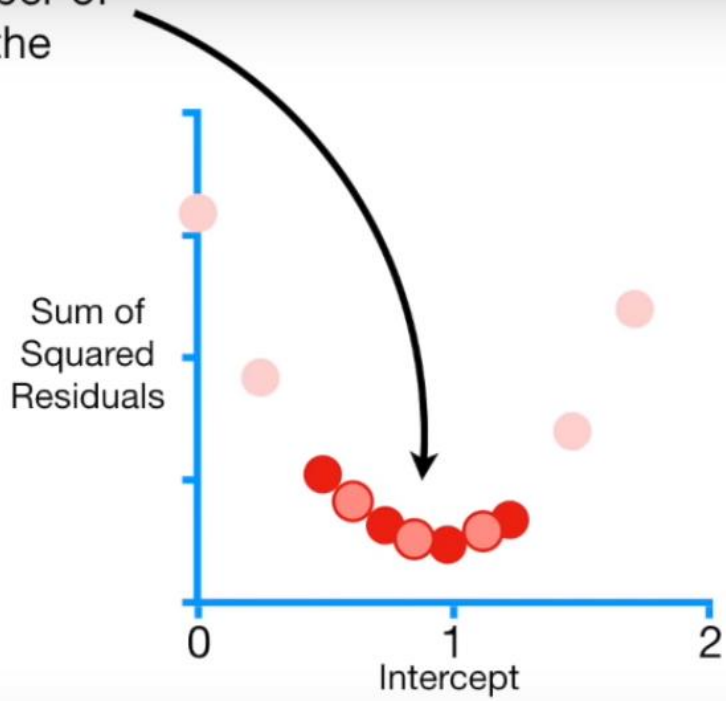


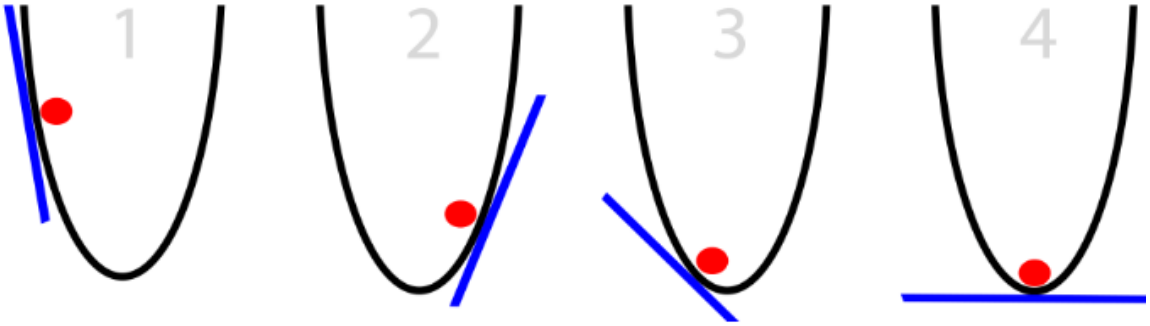


Gradient Descent only does a few calculations far from the optimal solution…



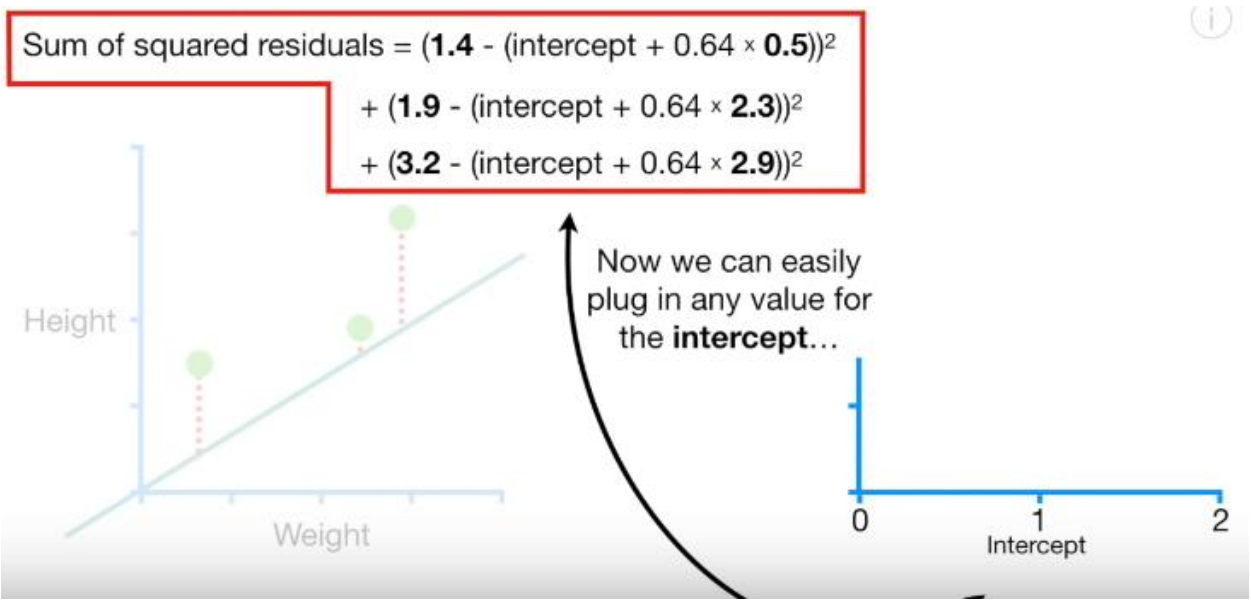
…and increases the number of calculations closer to the optimal value.

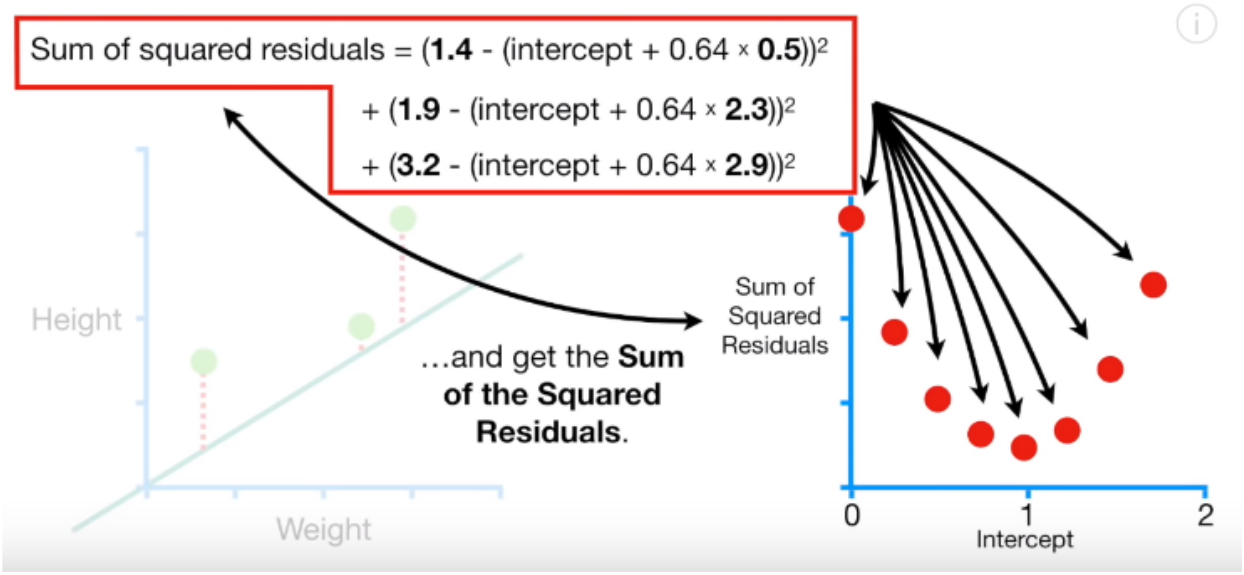


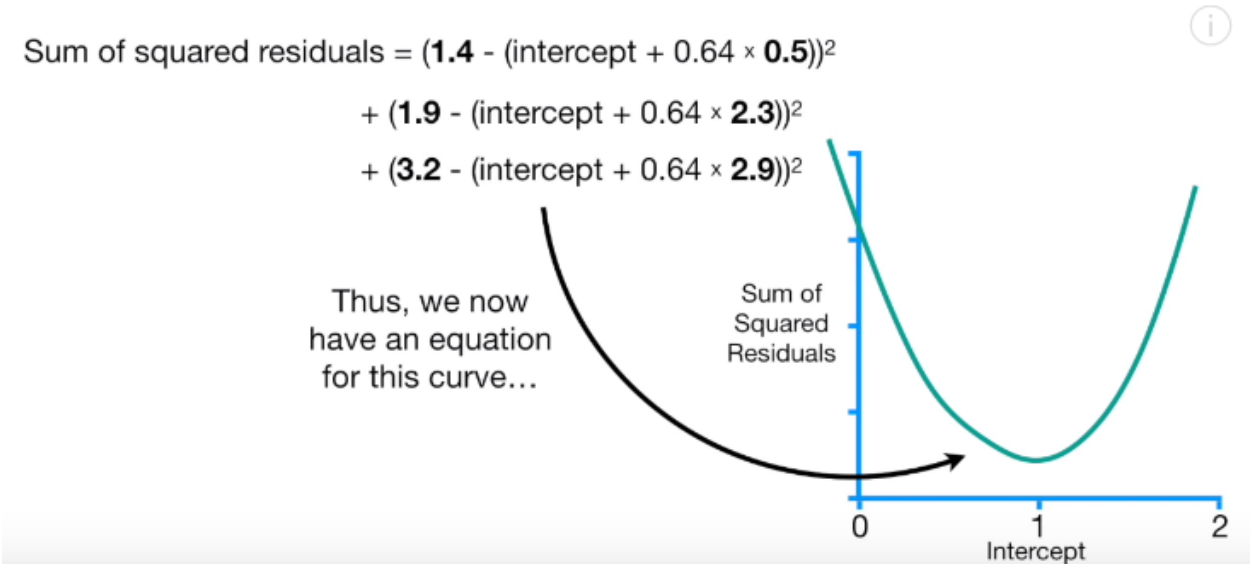


Oversimplified Gradient Descent:

- Calculate slope at current position
- If slope is negative, move right
- If slope is positive, move left
- (Repeat until slope == 0)

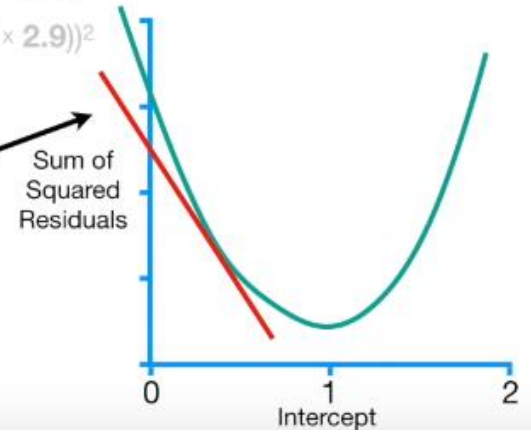



Sum of squared residuals = (**1.4** - (intercept + 0.64 × **0.5**))²
+ (**1.9** - (intercept + 0.64 × **2.3**))²
+ (**3.2** - (intercept + 0.64 × **2.9**))²
Height
Weight
…and get the **Sum of the Squared Residuals**.
Sum of Squared Residuals
0
1
2
Intercept


Sum of squared residuals = (**1.4** - (intercept + 0.64 × **0.5**))²
+ (**1.9** - (intercept + 0.64 × **2.3**))²
+ (**3.2** - (intercept + 0.64 × **2.9**))²
Thus, we now have an equation for this curve…
Sum of Squared Residuals
0
1
2
Intercept

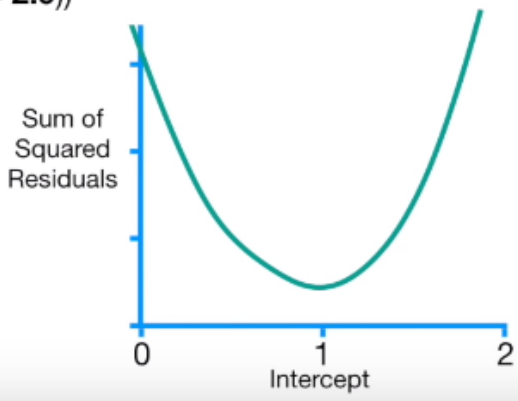
Sum of squared residuals = (**1.4** - (intercept + 0.64 × **0.5**))^2
+ (**1.9** - (intercept + 0.64 × **2.3**))^2
+ (**3.2** - (intercept + 0.64 × **2.9**))^2

…and we can take the derivative of this function and determine the slope at any value for the **Intercept**.



Sum of squared residuals = (**1.4** - (intercept + 0.64 × **0.5**))^2
+ (**1.9** - (intercept + 0.64 × **2.3**))^2
+ (**3.2** - (intercept + 0.64 × **2.9**))^2

So let's take the derivative of the Sum of the Squared Residuals with respect to the **Intercept**.



Sum of squared residuals = $(\mathbf{1.4} - (\text{intercept} + 0.64 \times \mathbf{0.5}))^2$

$+ (\mathbf{1.9} - (\text{intercept} + 0.64 \times \mathbf{2.3}))^2$

$+ (\mathbf{3.2} - (\text{intercept} + 0.64 \times \mathbf{2.9}))^2$

…plus the derivative of the third part.

$$\frac{d}{d\ intercept} \text{Sum of squared residuals} = \frac{d}{d\ intercept}(\mathbf{1.4} - (\text{intercept} + 0.64 \times \mathbf{0.5}))^2 + \frac{d}{d\ intercept}(\mathbf{1.9} - (\text{intercept} + 0.64 \times \mathbf{2.3}))^2 + \frac{d}{d\ intercept}(\mathbf{3.2} - (\text{intercept} + 0.64 \times \mathbf{2.9}))^2$$

$$\frac{d}{d\ intercept}(\mathbf{1.4} - (\text{intercept} + 0.64 \times \mathbf{0.5}))^2 = 2(\mathbf{1.4} - (\text{intercept} + 0.64 \times \mathbf{0.5}))$$

So we start by moving the square to the front…

$$\frac{d}{d\ intercept}(\mathbf{1.4} - (\text{intercept} + 0.64 \times \mathbf{0.5}))^2 = 2(\mathbf{1.4} - (\text{intercept} + 0.64 \times \mathbf{0.5})) \times -1$$

…and multiply that by the derivative of the stuff inside the parentheses.

$$\frac{d}{d\ intercept}\ \mathbf{1.4} - (\text{intercept} + 0.64 \times \mathbf{0.5})$$

$$\frac{d}{d\ intercept}\ \mathbf{1.4} + (-1)\text{intercept} - 0.64 \times \mathbf{0.5} = -1$$

$$\frac{d}{d\,intercept}(\mathbf{1.4} - (intercept + 0.64 \times \mathbf{0.5}))^2 = 2(\mathbf{1.4} - (intercept + 0.64 \times \mathbf{0.5})) \times -1$$

$$= -2(\mathbf{1.4} - (intercept + 0.64 \times \mathbf{0.5}))$$

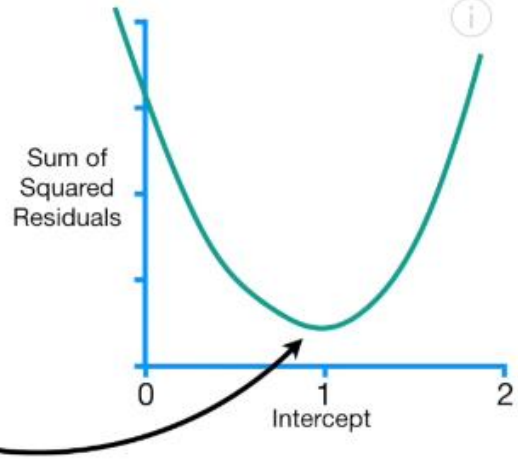
…and this…

…is the derivative of the first part…

$$\frac{d}{d\,intercept}\text{Sum of squared residuals} = \frac{d}{d\,intercept}(\mathbf{1.4} - (intercept + 0.64 \times \mathbf{0.5}))^2$$

$$\frac{d}{d\,intercept}\text{Sum of squared residuals} =$$
$$-2(\mathbf{1.4} - (intercept + 0.64 \times \mathbf{0.5})$$
$$+ -2(\mathbf{1.9} - (intercept + 0.64 \times \mathbf{2.3}))$$
$$+ -2(\mathbf{3.2} - (intercept + 0.64 \times \mathbf{2.9}))$$

$$\frac{d}{d\,intercept}\text{Sum of squared residuals} =$$
$$-2(\mathbf{1.4} - (intercept + 0.64 \times \mathbf{0.5})$$
$$+ -2(\mathbf{1.9} - (intercept + 0.64 \times \mathbf{2.3}))$$
$$+ -2(\mathbf{3.2} - (intercept + 0.64 \times \mathbf{2.9}))$$



Now that we have the derivative, **Gradient Descent** will use it to find where the Sum of Squared Residuals is lowest.

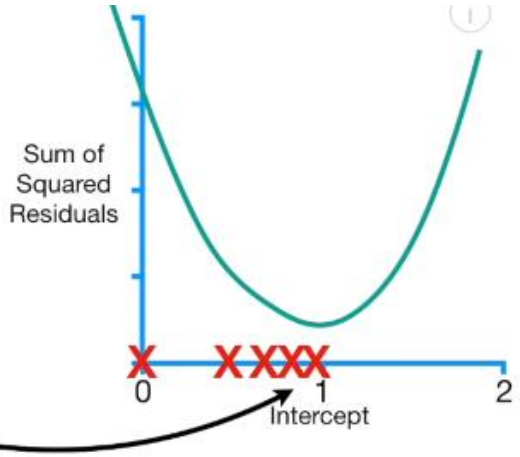
NOTE: If we were using **Least Squares** to solve for the optimal value for the **Intercept**, we would simply find where the the slope of the curve = **0**.

$\frac{d}{d\ intercept}$ Sum of squared residuals =

-2(**1.4** - (intercept + 0.64 × **0.5**)

+ -2(**1.9** - (intercept + 0.64 × **2.3**))

+ -2(**3.2** - (intercept + 0.64 × **2.9**))



In contrast, **Gradient Descent** finds the minimum value by taking steps from an initial guess until it reaches the best value.

$\frac{d}{d\ intercept}$ Sum of squared residuals =

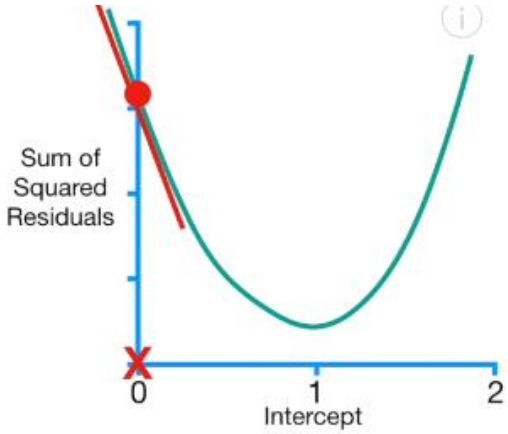
-2(**1.4** - (0 + 0.64 × **0.5**)

+ -2(**1.9** - (0 + 0.64 × **2.3**))

+ -2(**3.2** - (0 + 0.64 × **2.9**))

= -5.7

Step Size = -5.7



Gradient Descent determines the **Step Size** by multiplying the **slope**…

Step Size = -5.7 × 0.1

…by a small number called **The Learning Rate**.

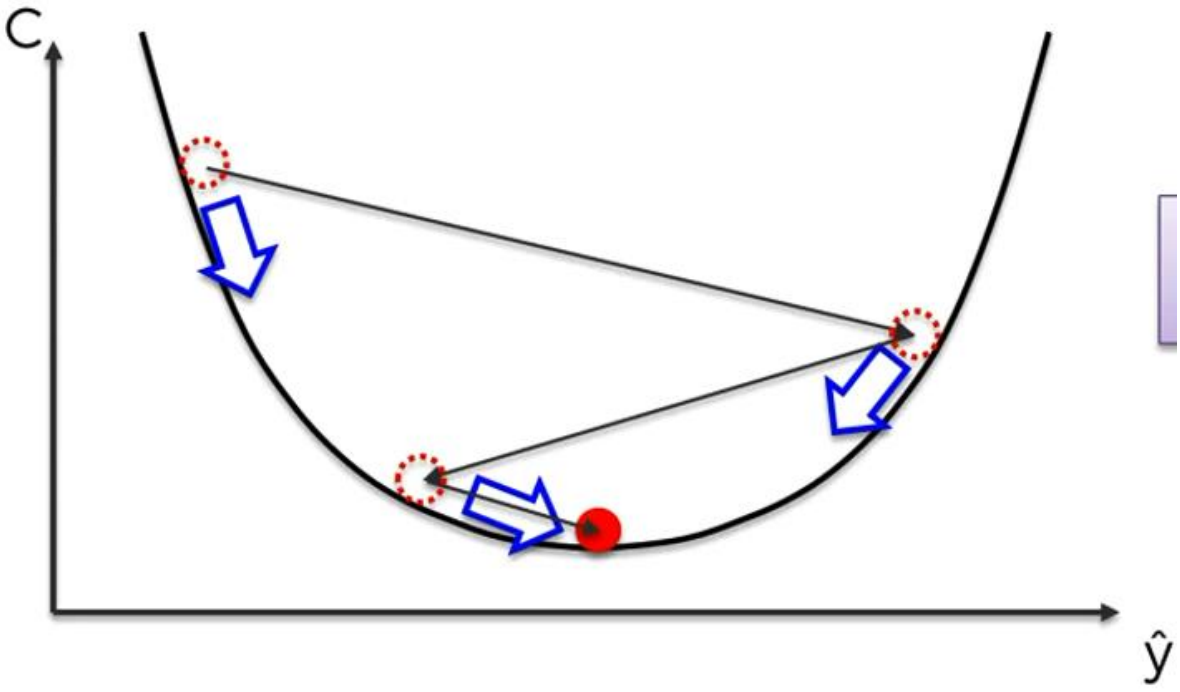
New Intercept =

With the **Step Size**, we can calculate a **New Intercept**.

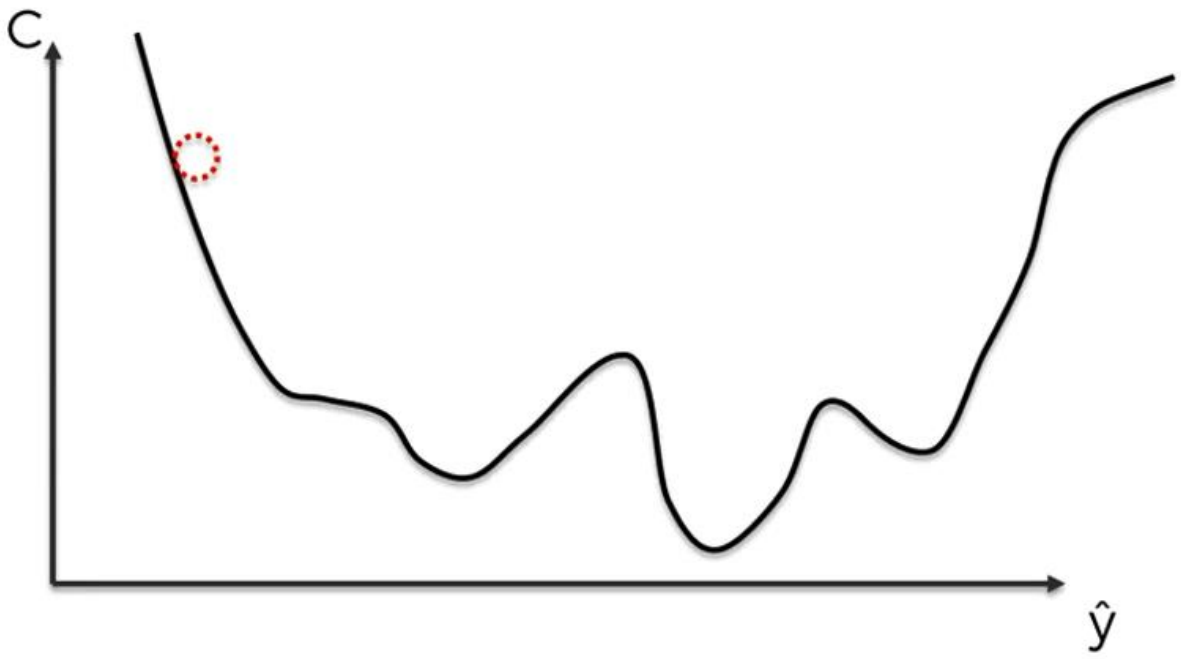
New Intercept = Old Intercept - Step Size

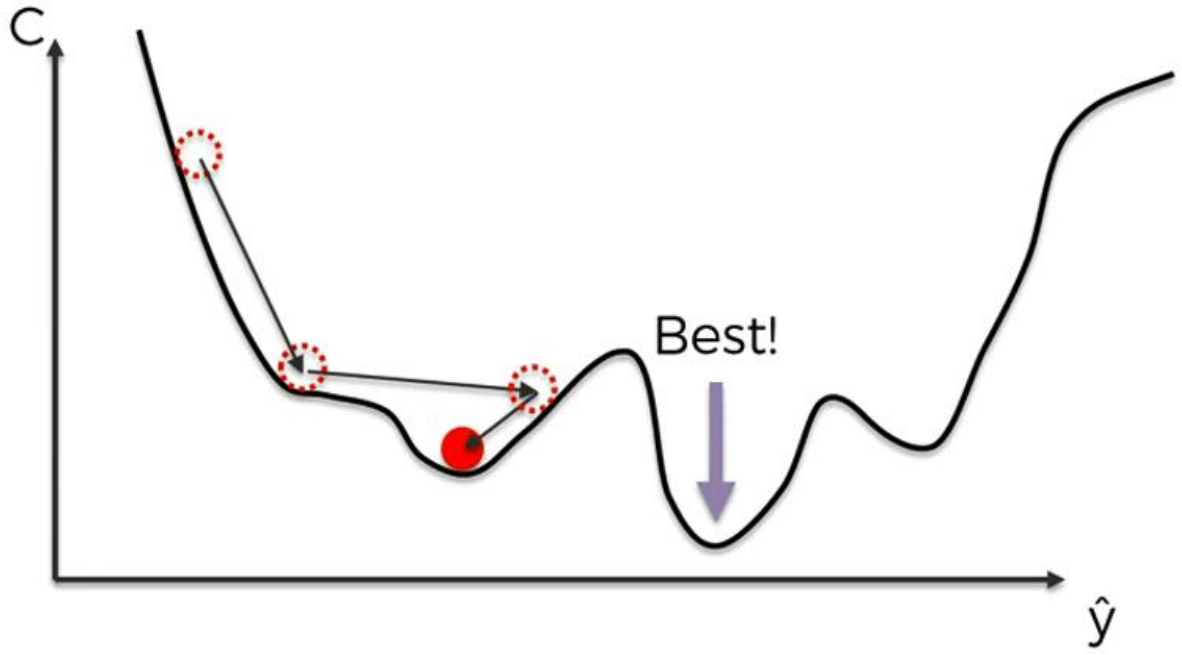
Stochastic Gradient Descent

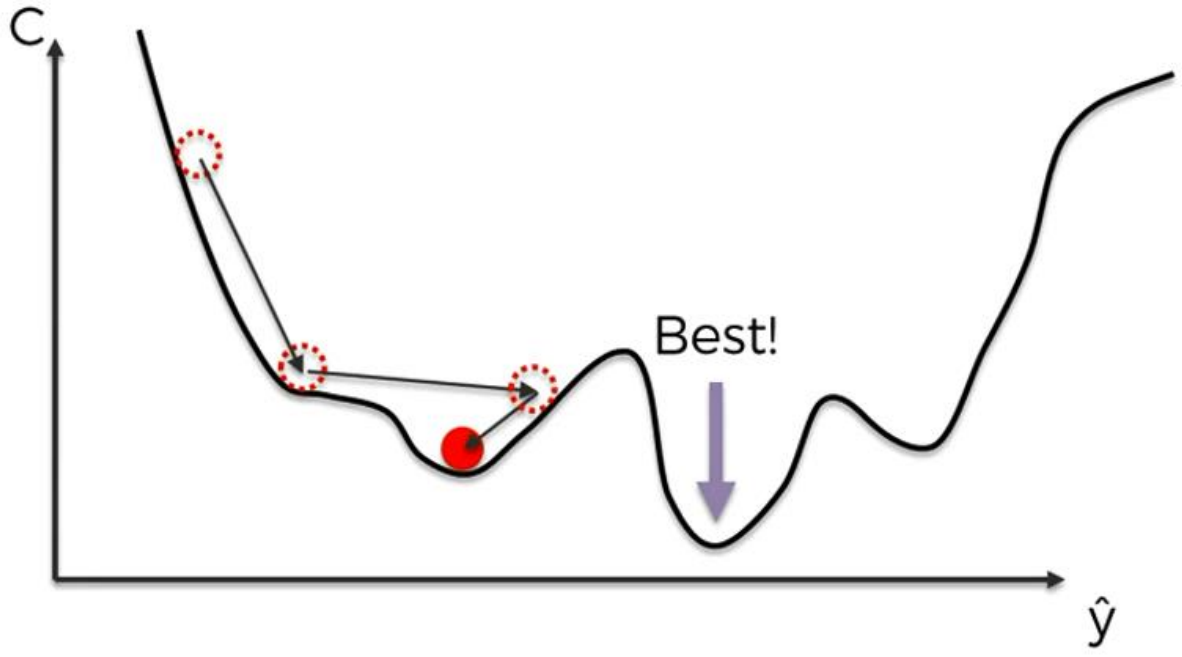
The GD/Batch GD requires the Cost function to be convex



But what if our Cost function is not convex. What if our cost function look something like


C
ŷ


C
Best!
ŷ


C
Best!
ŷ

Row ID	Study Hrs	Sleep Hrs	Quiz	Exam
1	12	6	78%	93%
2	22	6.5	24%	68%
3	115	4	100%	95%
4	31	9	67%	75%
5	0	10	58%	51%
6	5	8	78%	60%
7	92	6	82%	89%
8	57	8	91%	97%

Upd w's

Batch Gradient Descent

Row ID	Study Hrs	Sleep Hrs	Quiz	Exam
1	12	6	78%	93%
2	22	6.5	24%	68%
3	115	4	100%	95%
4	31	9	67%	75%
5	0	10	58%	51%
6	5	8	78%	60%
7	92	6	82%	89%
8	57	8	91%	97%

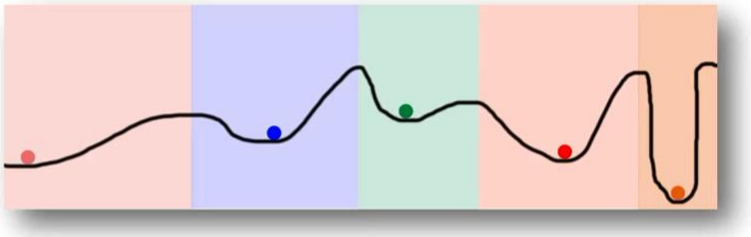
Upd w's
Upd w's
Upd w's
Upd w's
Upd w's
Upd w's
Upd w's
Upd w's

Stochastic Gradient Descent

Additional Reading:

A Neural Network in 13 lines of Python (Part 2 - Gradient Descent)

Andrew Trask (2015)



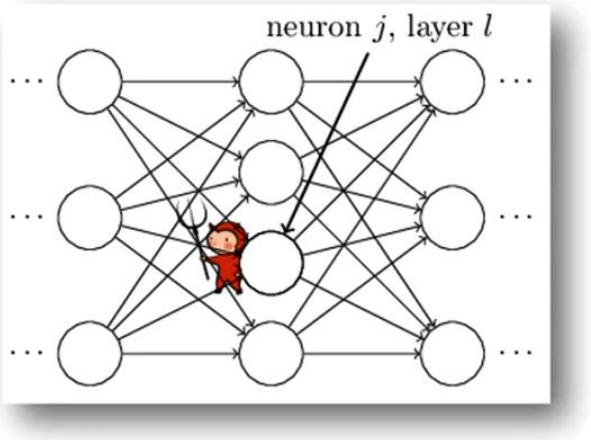
Link:

https://iamtrask.github.io/2015/07/27/python-network-part2/

Additional Reading:

Neural Networks and Deep Learning

Michael Nielsen (2015)



Link:

http://neuralnetworksanddeeplearning.com/chap2.html

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.

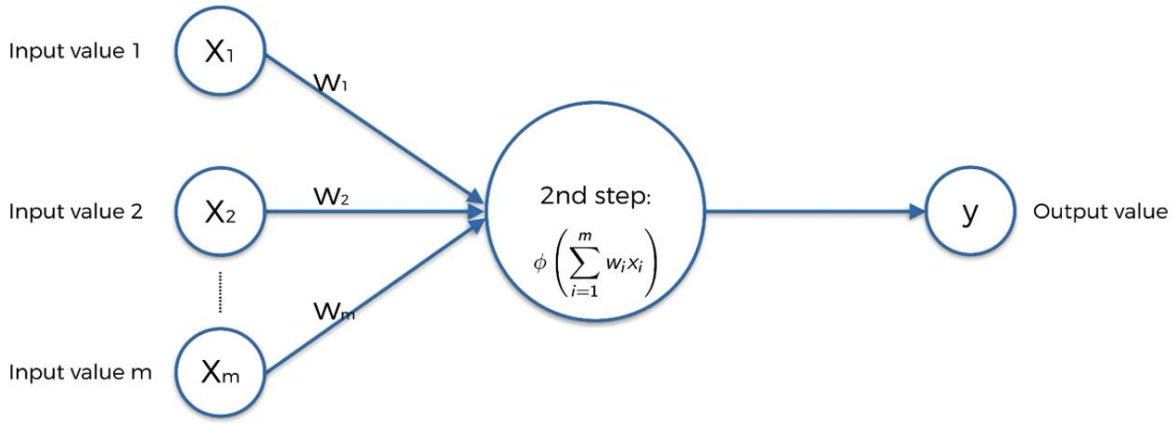
STEP 3: Forward-Propagation: from left to right, the neurons are activated in a way that the impact of each neuron's activation is limited by the weights. Propagate the activations until getting the predicted result y.

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.



The neuron applies the activation function to this sum. The closer the activation function value is to 1, the more activated is the neuron, and the more the neuron passes on the signal.