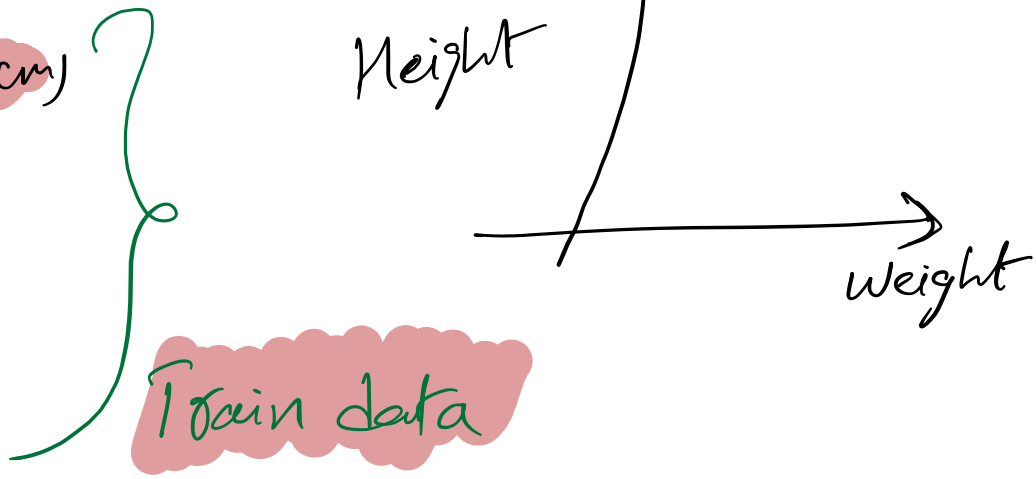
In statistics we call it ordinary least Square.....

→ Used in Regression (Supervised Learning)

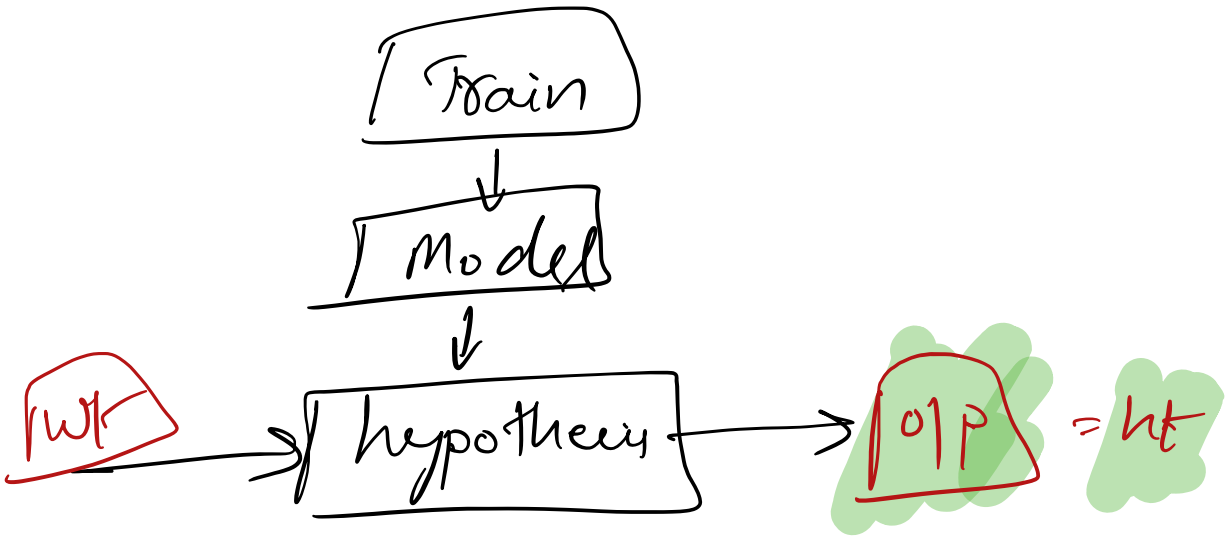
Mathematical Intution:-

Ex:

Wt	Ht (cm)
73	150
63	155
78	170
82	165



Train data

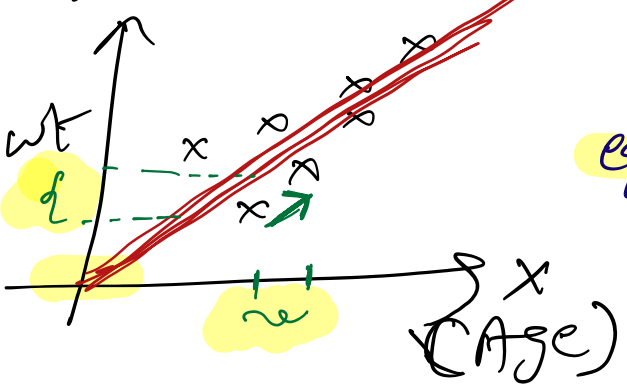


Note:

ht is a Continuous Variable

Krish Naik

y



equation of a st-line:

$$y = mx^{(i)} + c$$

it can also written in the form of

$$h_\theta(x) = \theta_0 + \theta_1 x$$

intercept = θ_0
where is it touching the y-axis

θ_1 = slope of co-efficient

Aim:
we should find out the best fit line, in such way that the distance b/w data points and the predicted values is min.

[we take the sum^n of all these distances, then it should be minimal]

⇒ Now, there might be many such lines which can be drawn across the points...

→ so, for this confusion, we start at point and we go in a path such that we reach (or) get the best fit line at the end of all the data points.

→ So to guide us, what path (or) what way we should follow we have smtg called as cost function.

Cost function:-

the dist formula b/w the predicted (and) the real data

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x) - y \right)^2$$

hypothesis function

↳ $\theta_0 + \theta_1 x$

m = all the points present

↳ so, (i) will run from i=1 till m

we need to keep on updating θ_0 and θ_1 unless we get the best fit line.

note: we can just divide by 'm' but we are taking 1, so that it will help us in derivation.

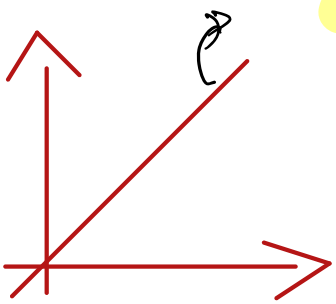
The above formula is also called as Squared Error Function

What we need to solve:

$$\underset{\theta_0, \theta_1}{\text{minimize}} \quad \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x)^{(i)} - y^{(i)} \right)^2$$

minimize $J(\theta_0, \theta_1)$
$\theta_0\ \theta_1$

$h_\theta(x) = \theta_1 x$

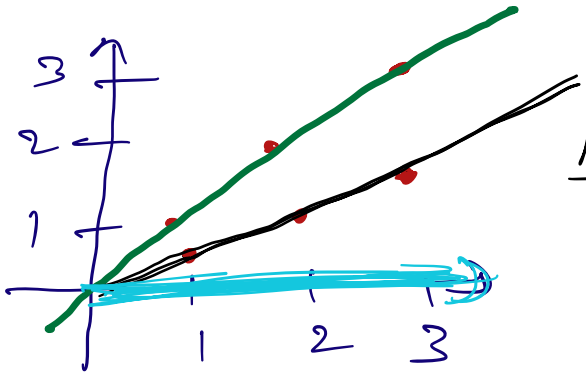


② $h_\theta(x) = \theta_0 + \theta_1 x$ If $\theta_0 = 0$

intercept is passing through origin.

ex!

(1,1) (2,2) (3,3) — data points



$\theta_1 = 1$

Scenario = 1

best fit line is given by green line

$$J(\theta_1) = \frac{1}{2m} \sum_{i=1}^{3} \left(h_\theta(x)^i - y^{(i)}\right)^2$$

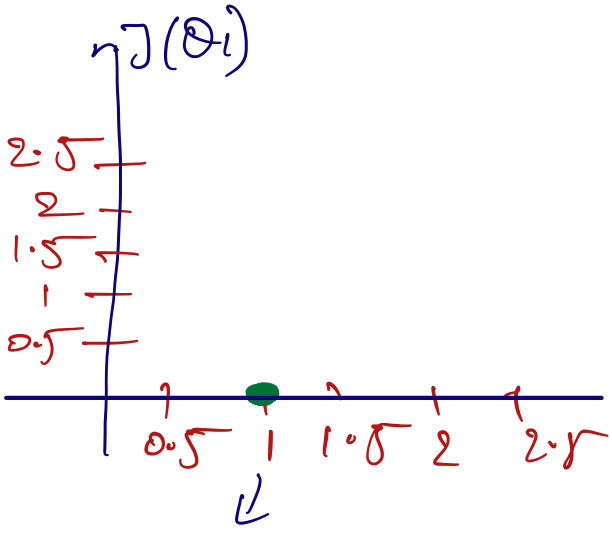
$$= \frac{1}{2m} \left[\theta_0 + \theta_1 x \quad + (2-2)^2 + (3-3)^2 \right]$$

$\theta_0 = 0$

$\left((1 \times 1) - 1\right)^2$

$J(\theta_1) = 0$

Cost func^n graph →



Scenario = 2

$\theta_1 = 0.5$

$h_\theta(x) = \theta_0 + \theta_1 x$

$= 0 + 0.5(1)$

$= 0.5$

My $x=2 \Rightarrow 1$

$x=3 \Rightarrow 1.5$

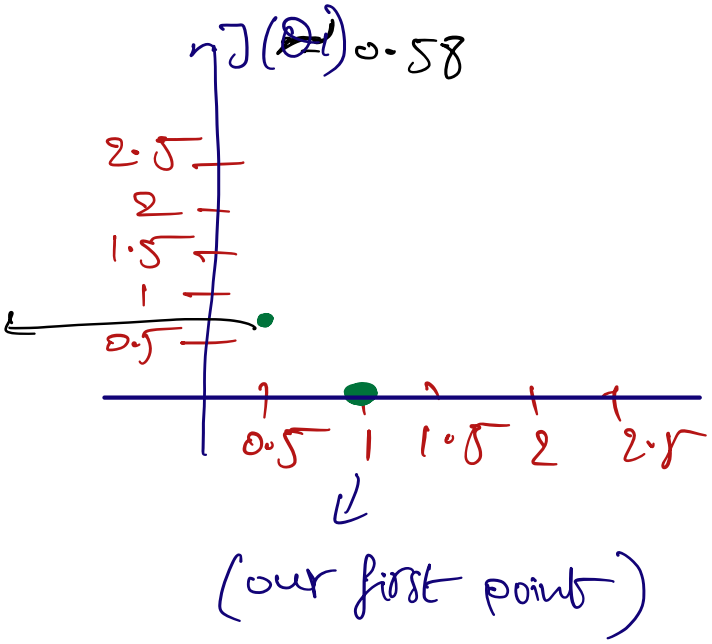
cost function

$\Rightarrow \frac{1}{2m}\left[(0.5-1)^2 + (1-2)^2 + (1.5-3)^2\right]$

predicted — actual

$= \frac{1}{2\times 3}\left[0.25 + 1 + 2.25\right]$

plotting the best fit line in the above graph with black color.



(our second point)

(our first point)

Scenario = 3

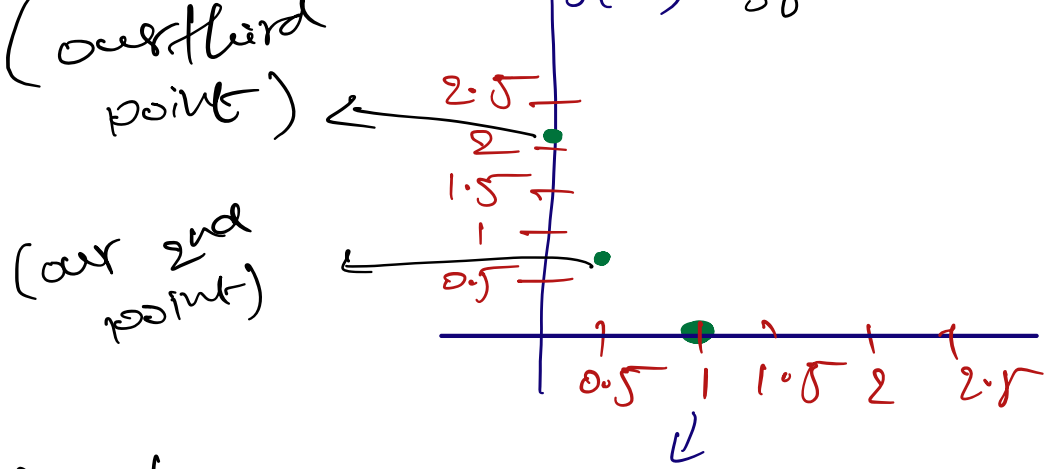
$\theta_1 = 0$

then in the plot our best fit line is given by blue color, Calculating cost function.

$J(\theta_0, \theta_1) = \frac{1}{2\times 3}\left[(0-1)^2 + (0-2)^2 + (0-3)^2\right)$

$= \frac{1}{6}(1+4+9) \simeq 2.3$

J(θ1) 0.58



(our first point)

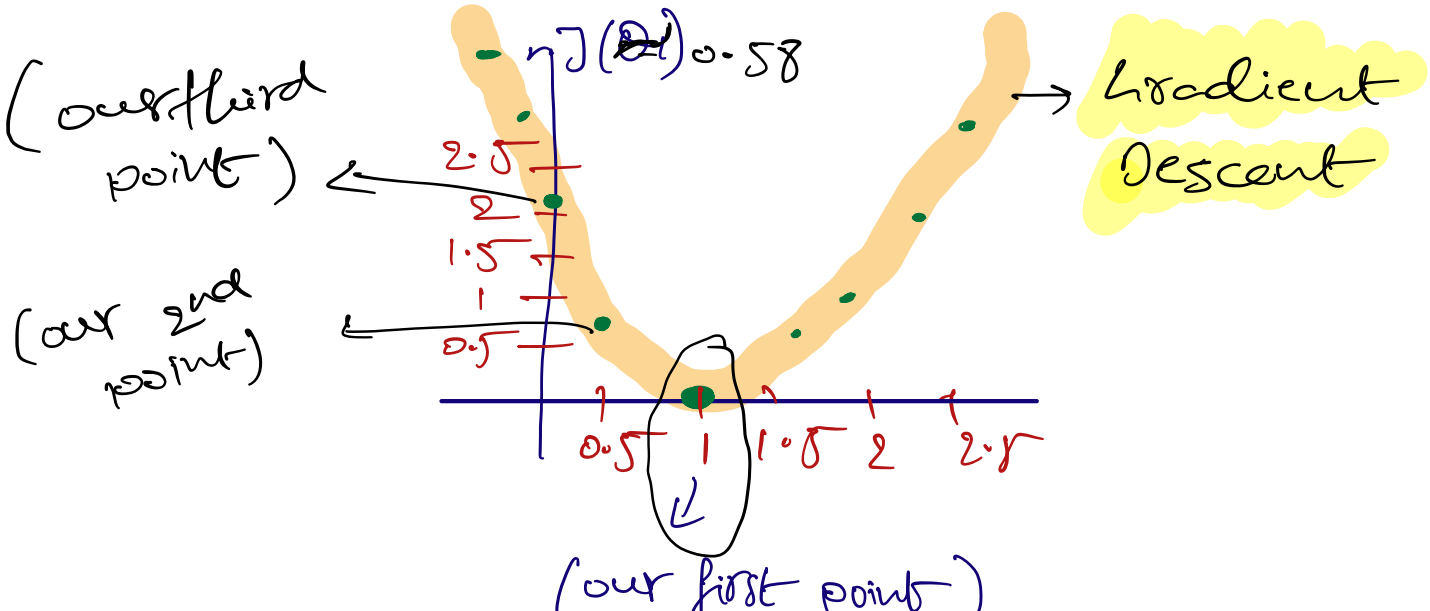
note:

⇒ So, for diff values of θ_1 we get diff points on the graph of the cost function.

⇒ Joining all those points give us a kind of curve, which we call as

Gradient Descent

→ This GDF play a main role in getting the right θ_1 value



The point on X-axis ($x=1$) is called as Global minima

→ Our total concentration will be on getting this global minima, where we get the best fit line. If we see, our $\theta_1=1$ gave us the best fit line.

note: we have actually assumed $\theta_1=1$ for the first time and we got the best-fit directly.....

→ Else, we pick some value for θ_1 and check do we get best fit line (or) not
→ then we update our values of θ_1 accordingly. In a way, that our slope in the Gradient Descent will give us the Global minima

→ to assume diff values of θ_1 we can't randomly pick values, for this reason we take the help of learning rate and try to move towards our global minima

Conversion algorithm:

→ after getting a particular θ_1 value we just need to get close to our global minima. So, in this process we are actually leaving out few value of θ_1 which may not guide us to global minimum

[life got simple :)]

→ repeat until convergence # suppose a while loop

{

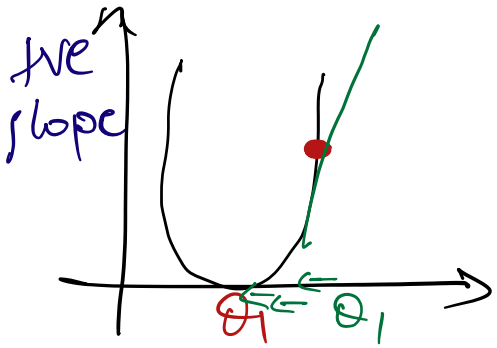
$$\theta_j := \theta_j - \alpha \left[\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \right]$$

}

→ learning rate

→ we use this formula to find global minima.

→ derivative to find slope



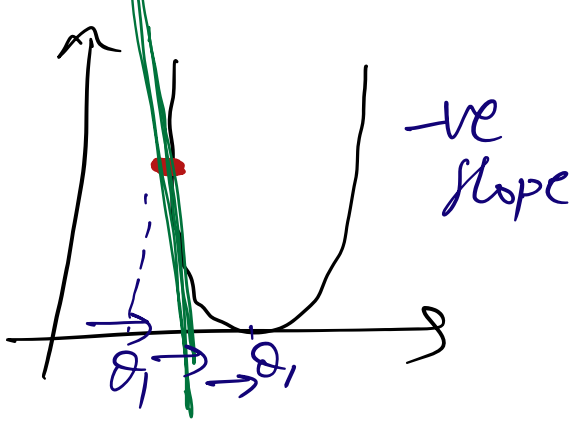
right hand side of slope is pointing top.

$$\theta_1 : \theta_1 - \alpha(+ve)$$

$|-ve|$

↳ +ve number

So, after few iterations, we come to our global minima



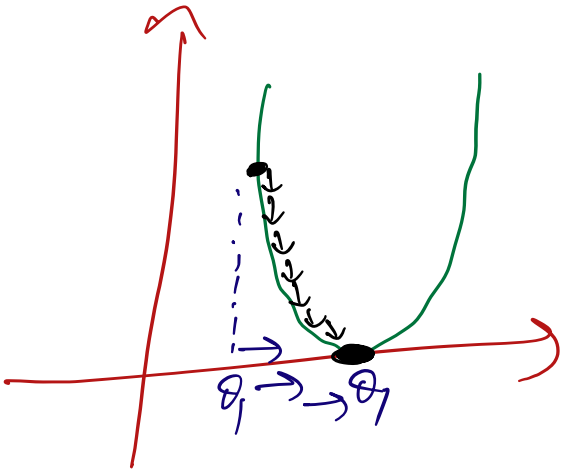
$$\theta_1 : \theta_1 - \alpha(-ve)$$

$$\theta_1 : \theta_1 + \alpha(ve)$$

So, after few iterations, we come to our global minima

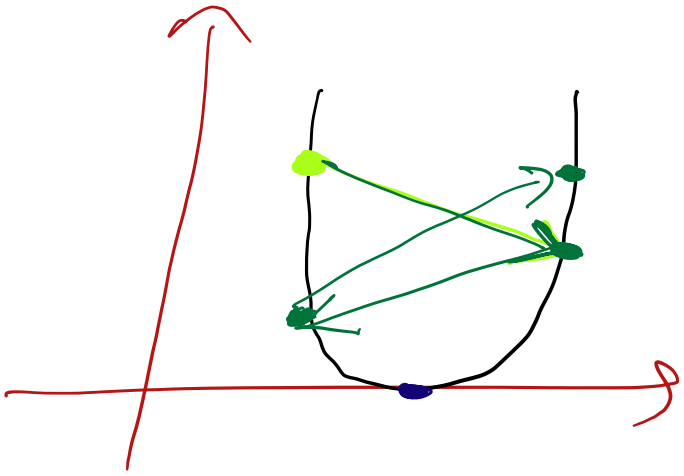
Learning rate: By what speed we should come towards our global minima

ex: for $\alpha = 0.01$



it actually takes small steps to reach our global minima which is so time consuming

for α = huge



this creates a mess and it keeps on jumping and we can't reach global minima.....

note! so, acc to the dataset, acc to our data values we should decide what value of α is best for our model.

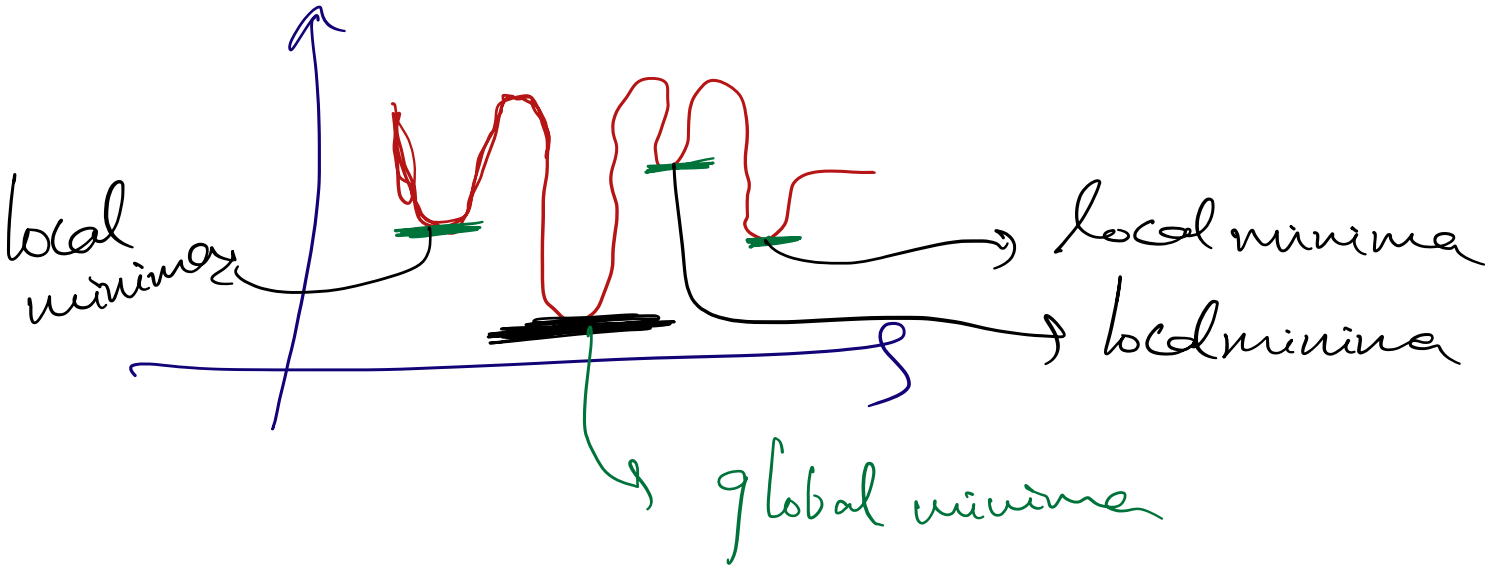
global minima is a point where the slope = 0

Local minima Vs Global minima:

→ In a cost function graph when take a big dataset and we have to find more values of θ_1

→ then we might get stuck at a local minima, but we choose the first θ_1 value randomly

→ There might be a chance for some other θ_1 value there is a global minima, which we dont know.



note:
These ML algorithms can't deal with this issue, we need neural networks to solve this, which we will learn in ~~ATI~~.

↓
Deep learning. (GD Algo's
RMS prop
Adam optimizer
etc...)

Gradient Descent Algo:-

repeat until Convergence # Suppose a while loop

{

$$\theta_j := \theta_j - \alpha \left[\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \right]$$

}

↳ learning rate

$$\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_j} \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

$\theta_0 + \theta_1 x$

i) $j=0 \Rightarrow \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)$

ii) $j=1 \Rightarrow \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x^{(i)}$

↳ called as Conversion Algorithm

Repeat until Convergence

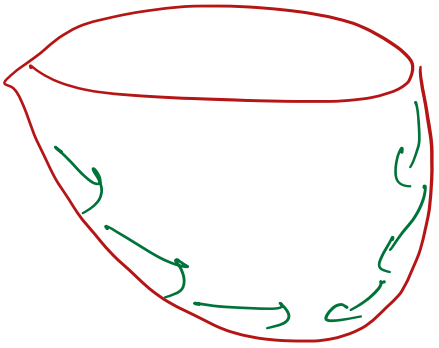
{

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})\, x^{(i)}$$

}

Convex:



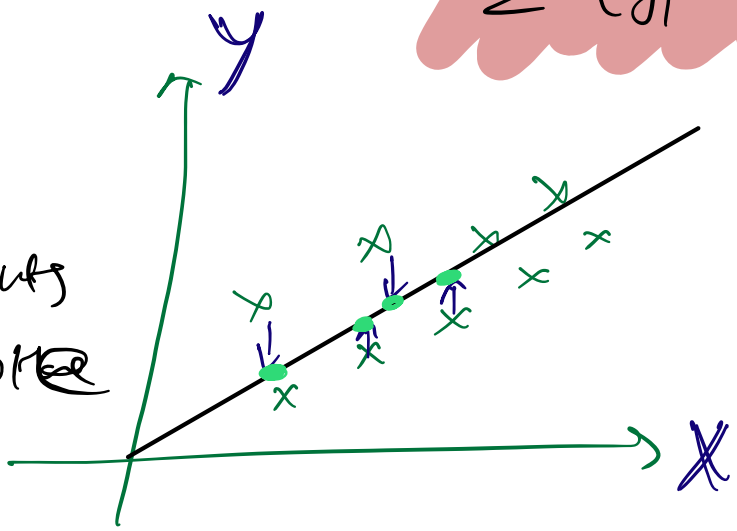
for a 3-d GD

performance matrix: R^2, Adjusted R^2

$$R^2 = 1 - \frac{SS_{Res}}{SS_{Total}} = 1 - \frac{\sum (y_i - \hat{y}_i)^2}{\sum (y_i - \bar{y})^2}$$

$\hat{y}_i = h_\theta(x)$

• = predicted points which we denote $\hat{y}_i$

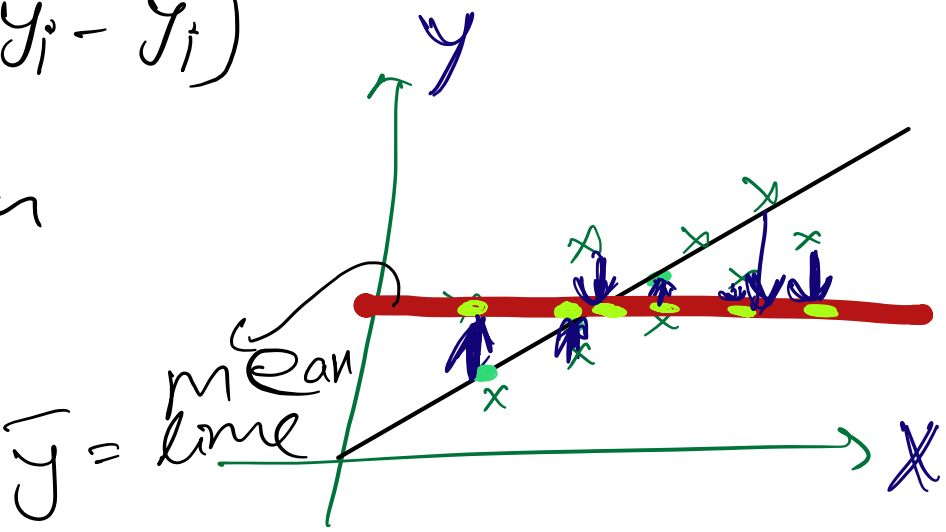


SS_{res} = [diff b/w (x) and (•)] ← Summation of these

(x) Point (•) predicted (or) $\hat{y}_i$

$$SS_{Total} = (y_i - \bar{y}_i)^2$$

$\bar{y}_i$ = mean



Note: The diff of dist is high in SS_{total}

↳ i.e, the mean value dist will be higher.

↳ $1 - \frac{\text{low}}{\text{high}}$ → small num

$\Rightarrow 1 -$ small num

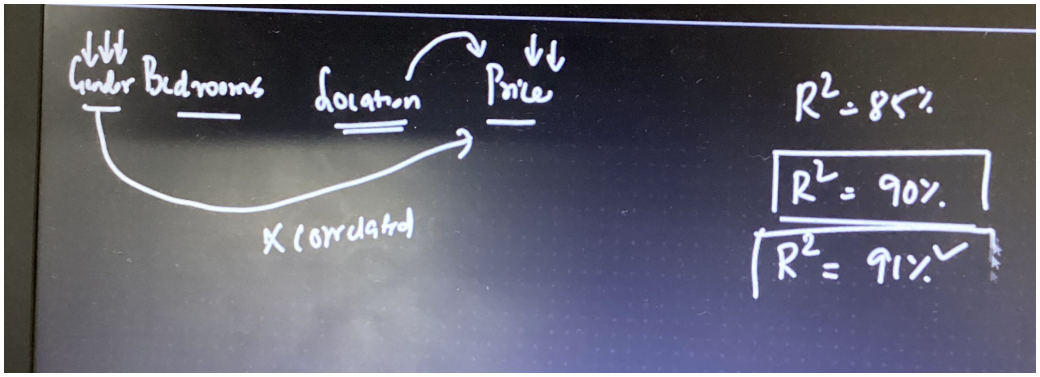
$\Rightarrow$ big number.

$\Rightarrow$ R^2 value is good (if fitted well)

Note:

If we add values in our dataset, even though they are not at all correlated, R^2 value will keep on increasing.

exr



we use adjusted R^2 for this issue.

$$1 - \frac{(1-R^2)(N-1)}{N-p-1}$$

N = total no. of samples / data points

p = no. of features (or) predictors

exr $p = 2 \Rightarrow R^2 = 90\% \Rightarrow R^2_{adj} = 86\%$

$p = 3 \Rightarrow R^2 = 91\% \Rightarrow R^2_{adj} = 82\%$

⇒ This is because of the formula we took, which balances the performance acc to our N and p values