Lec. XII. Gradient Descent

- Gradient Descent is a method for unconstrained mathematical optimization. It's a first-order iterative algorithm for finding a local minimum of a differentiable multi-variate function.

- Intuition:

$$\hat{y}_i = m x_i + b$$

$$L = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

$$L = \sum_{i=1}^{n} (y_i - m x_i - b)^2$$

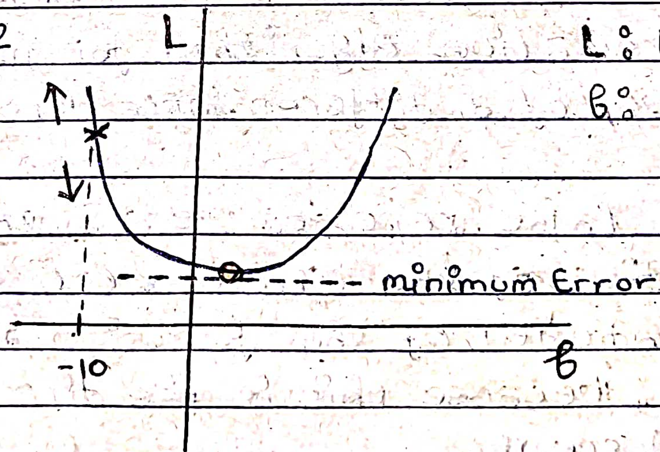
$$\to L(m, b)$$

Let, L is depending on 'b' only by putting $m = 78.35$

$$L(b) = \sum_{i=1}^{n} (y_i - 78.35 * x_i - b)^2$$

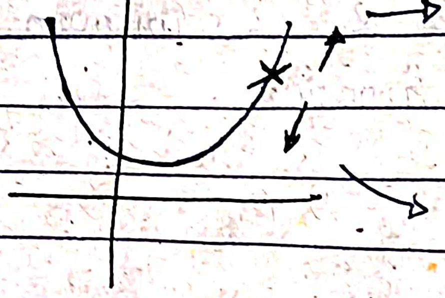
Now, we want to minimize error (L)

$L \propto b^2$



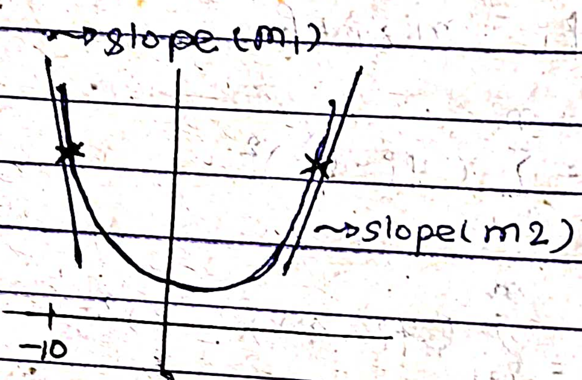
L: Loss function
b: intercept

suppose we select a random $b_{min} = -10$ (ie, checking at this value b is min or not) if we increase value of (b = -10) we move toward min. Error and go away we decrease value of b



→ increasing value of b, moving away from min Error

→ decreasing value of b, moving toward min Error.



m_1 → slope ⊖ve
we have to increase b

m_2 → slope ⊕ve
we have to decrease b.

$b_{new} = b_{old} - slope$

here, $b_{new} \Rightarrow -10 - (-78.35)$
$\Rightarrow -10 + 78.35 \Rightarrow 68.35$

- Hence, gradient descent is a first-order iterative optimization algorithm for finding a local minimum of a differentiable function
- The idea is to take repeated steps in the opposite direction of the gradient (or approximate gradient) of the function at the current point, because this is the direction of steepest descent.

- first order ⟶ finding slope 'm'

- iterative first order optimization ⟶ iteratively updating parameter (B) in search of an optimal solution.

- gradient ⟶ vector of partial derivative representing the rate of change of a multi-variate function

for func → $f(x_1, x_2, \ldots, x_n)$

∇f ⟶ $\left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n}\right)$
(gradient of func)

- gradient is instrumental in determining how to adjust the parameter to minimize the objective function.

- In gradient descent, the goal is to move in the opposite direction of the gradient to decrease the objective function.

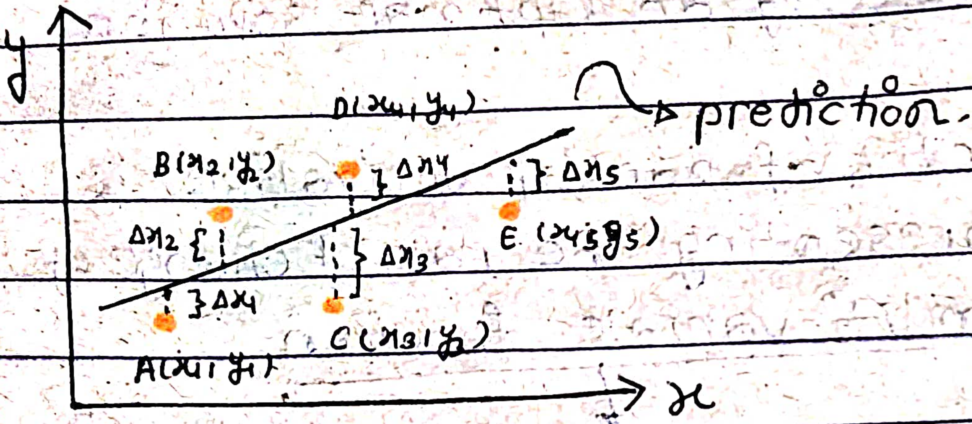
- The negative gradient $(-\nabla f)$ points in the dirn of the steepest decrease in the function.

$$(x_i)_{new} = (x_i)_{old} - \text{learning rate} \frac{\partial f}{\partial x_i}$$

Learning rate ⟶ rate at which we move toward minimum Error

eg 0.0001 , 0.1 , 0.11 ⟶ bit faster
0.0001 ⟶ very slowly moving toward min Error

Mathematical Formulation :



Error / Cost function $\quad J = y_i - \hat{y}_i$

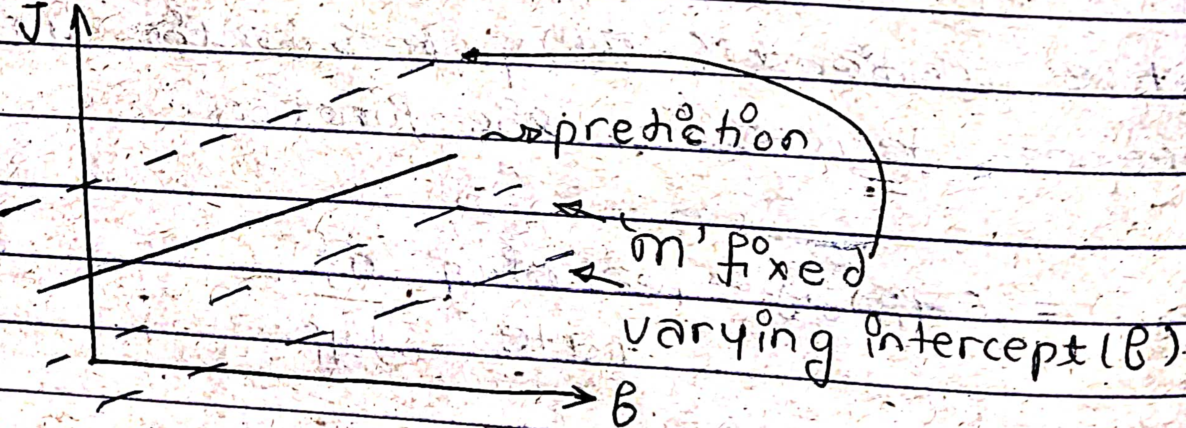
$$J = [y_i - (m x_i + b)]$$

$$J = \sum_{i=1}^{n=5} [y_i - (m x_i + b)]^2$$

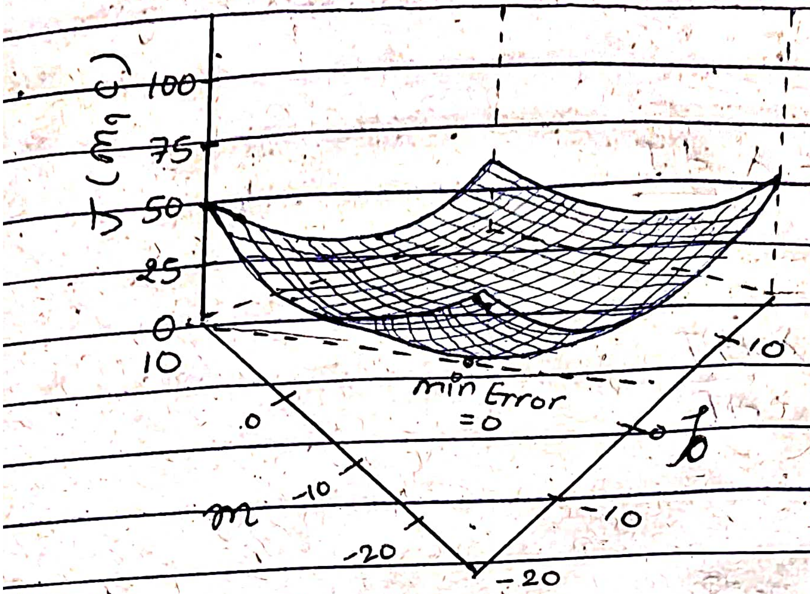
[Sum squared error (SSE)]

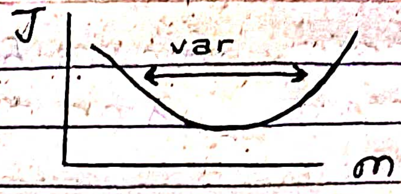
$$MSE = \frac{1}{n} \sum_{i=1}^{n} [y_i - (m x_i + b)]^2$$

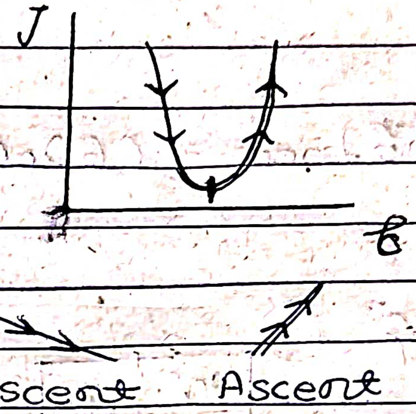
Mean Squared Error



★ Algorithm:







$$J = \frac{1}{n}\sum_{i=1}^{n}\left[y_i - (mx_i + b)^{8}\right]^2$$

$$\frac{dJ}{dm} = \frac{2}{n}\sum_{i=1}^{n}(-x_i)\left[y_i - (mx_i + b)\right]$$

$$0 = \sum_{i=1}^{n} -x_i y_i + m x_i^2 + x_i b$$

$$\boxed{\sum_{i=1}^{n} m x_i^2 + \sum_{i=1}^{n} x_i b = \sum_{i=1}^{n} x_i y_i}$$

$$\frac{dJ}{db} = \frac{2}{n}\sum_{i=1}^{n} -\left[y_i - (mx_i + b)\right]$$

$$0 = \sum_{i=1}^{n} -y_i + m x_i + b$$

$$\Rightarrow \boxed{m\sum_{i=1}^{n} x_i + \sum_{i=1}^{n} b = \sum_{i=1}^{n} y_i}$$

now, for each iteration we set a learning rate. (λ or α)

$$(m)_{new} = (m)_{old} - \lambda \frac{dJ}{dm}$$

$$(b)_{new} = (b)_{old} - \lambda \frac{dJ}{db}$$

i.e. (1)

	x_i	y_i	$x_i y_i$	x_i^2
	1	1	1	1
	3	3	9	9
	5	4	20	25
	7	3	21	49
	9	5	45	81
$\Sigma =$	25	16	96	125

equn (1) : $m\sum_{i=1}^{5} x_i^2 + c\sum_{i=1}^{5} x_i = \sum_{i=1}^{5} x_i y_i$

$$165m + 25c = 96$$

equn (2) : $m\sum_{i=1}^{n} x_i + 5c = \sum_{i=1}^{5} y_i$

$$25m + 5c = 16$$

$m(165) + c(25) = 96$

$m(25) + c(5) = 16 \times 5$

$125m + 25c = 80$

$40m = 16$

$m = 16$ put in eq - (1)

$c = 2/5$