Gradient Descent

Gradient descent is an Optimization algorithm used to reduce the cost (error) by repeatedly updating the model weights in the direction of steepest decrease of the cost function.

OLS vs Gradient descent:-

Point	OLS	Gradient Descent
i) Computation Cost	very high → matrix inversion $O(n^3)$	Much lower — Scalable for large datasets
ii) Dataset Suitability	Works well for small/medium datasets	Best for large datasets/Big data.
iii) Feature Count	Struggles when features >>> samples (high-dimens.)	works well in high dimen.
iv) memory usage	Needs entire dataset in memory	Can use stochastic GD → mini batches

GD

→ Generalize to Complex models:-

- OLS only works for Linear regression, GD works for almost every ML algorithm.

→ Avoids Expensive matrix inversion:

No need to compute $(X^T X)^{-1}$

→ works even when $X^T X$ is non invertible

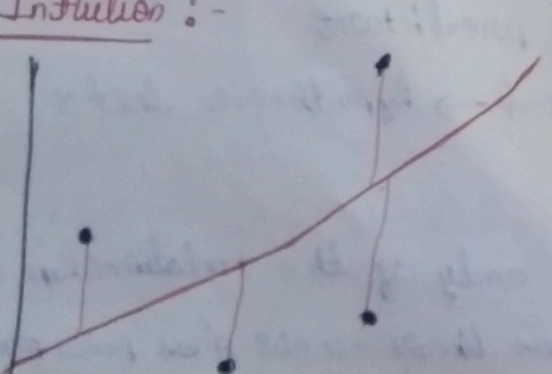
multicollinearity doesn't block GD.

→ Online learning possible :- can update model continuosly (real-time learning).

Intuition :-

cgpa	lpa
≈	≈

$\hat{y}_i = mx + b$



$$L = \sum_{i=1}^{4} (y_i - \hat{y}_i)^2$$

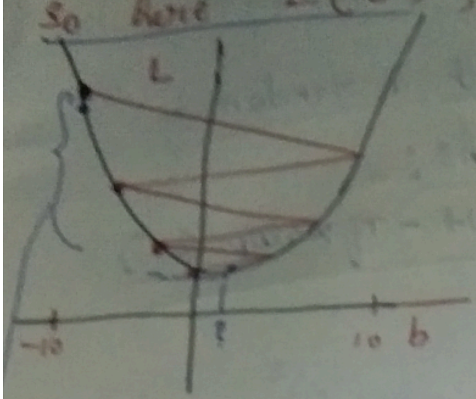
$$L = \sum_{i=1}^{4} (y_i - mx_i - b)^2$$

could write

Consider if m=78...

$$L = \sum_{i=1}^{4} (y_i - 78.35 * x_i - b)^2$$

So here



Step 1:- Select a random b_{min} such that L_{min}.

→ slope = -ve ⇒ move forward

→ Slope = +ve ⇒ move backward

$$\boxed{b_{new} = b_{old} - \text{Slope}}$$

if $b = -10$, & consider Slope = -50

$b_{new} = -10 - (-50) \Rightarrow 40$

if $b = 10$, Slope = +50

$b_{new} = 10 - (50) = -40$

→ There is a drastic changes here, Some time we may miss actual minima point. To reduce, we have to add (η)

$$\boxed{b_{new} = b_{old} - \eta \text{ slope}}$$

i) $b_{new} = -10 + (0.01 \times 50) = -9.5$

ii) $b_{new} = -9.5 + (0.01 \times 50) = -9$

here there is slight change

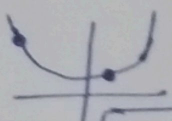
→ η = learning rate

default {0.01}

→ The idea is to take repeated steps in the opposite direction of gradient of the function at the current point. because this is the direction of steepest descent.

→ stepping in the direction of the gradient will lead to local

when to stop:-



i) Diff b/w old & new ⇒ $\boxed{b_{new} - b_{old} > 0.001}$ should

$\boxed{b_{new} - b_{old} = 0}$ ⇒ should stop } Convergence

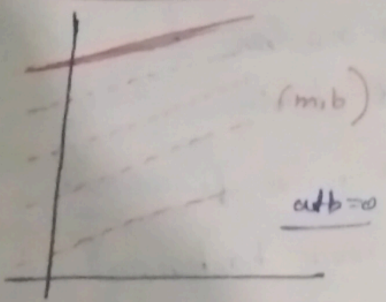
ii) max Iteration / epochs ⇒ 1000, 500, 100, etc

→ Convergence : Gradient becomes very small

If the changes in parameters is almost zero

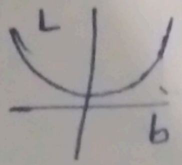
→ (divergence): if learning rate is too high → model may overshoot.

Mathematical Formulation: (consider we know $m = 78.35$)



→ Start with a random b

$$\boxed{\begin{array}{l} \text{for } i \text{ in epochs}: 1000 \\ \quad b_{new} = b_{old} - \eta \times slope_{(b=0)} \end{array}}$$



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

$$\frac{dL}{db} = \frac{d}{db} \sum_{i=1}^{n} (y_i - mx_i - b)^2$$

$$\boxed{= 2 \sum_{i=1}^{n} (y_i - mx_i - b)(-1)}$$

$$= -2 \sum_{i=1}^{n} (y_i - 78.35\, x_i - 0)$$

at slope = 0,

$b_{new} = b_{old} - \boxed{\eta\, slope_{b=0}} \rightarrow$ stepsize

then increment $i = 1$, calculate b_{new} again. the loop will run until the last epochs.

→ In previous mathematical Intuition, we considered m as constant

Now Performing Gradient Descent by adding 'm':

Steps :- 1) initializing values for m and b

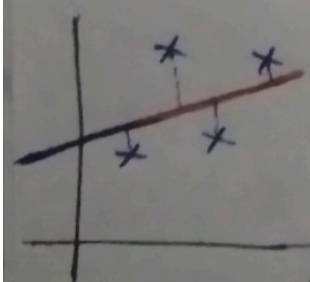
$\boxed{m = 1}$ and $\boxed{b = 0}$

2) epochs = 100, lr = 0.01

for i in epochs:

$b = b - \eta\, slope$

$m = m - \eta\, slope$



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

→ here $L(m, b)$, L is a function $(m \& b)$

$\boxed{b\_slope = \frac{\partial L}{\partial b}}$ $\boxed{m\_slope = \frac{\partial L}{\partial m}}$

$$\frac{\partial L}{\partial b} = -2 \sum_{i=1}^{n} (y_i - mx_i - b) \quad (-1)$$

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

⇒ slope-b at $\boxed{b = 0}$

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

$$= -2 \sum_{i=1}^{n} (y_i - mx_i - b) x$$

⇒ Slope-m at $\boxed{m = 1}$