28-09-2025

Agenda – ML - III

→ Gradient, Gradient Descent, practical, Linear Regression (EDA - Model prediction)

Dataset:

Years_of_exp (x1)	Completed_projects (x2)	Salary_k (y)
1 x m_1	0 x m_2	→ 39.764
2	1	48.400
3	1	56.978
4	2	68.240
5	3	77.867
6	4	85.022

$y = mx + c$

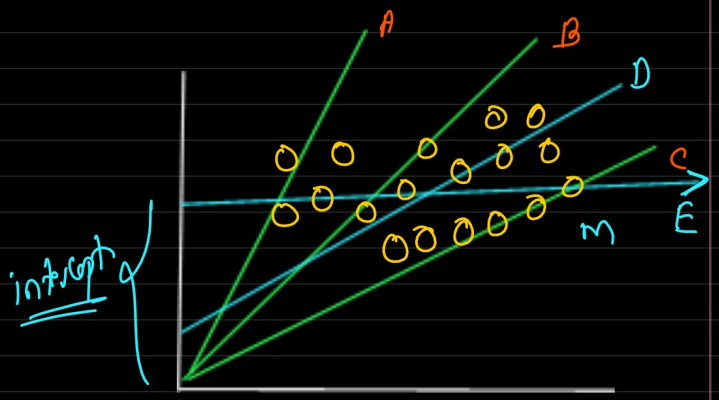
$y = mx + c$ → intercept / bias

10 0

1 Y

$\hat{y} = mx + c$

$x = 10$
$m = 1.5$
$c = 0$
$y = 20$
$\hat{y} \rightarrow 15$] error → 5



1 data point for prediction calculation

$y = m_1 x_1 + c$ (x_0)

11 0.5 1

x_1	y	$\hat{y}$	$\hat{y}$ m=11
1	11.5	11	11.5
2	22.5	22	22.5
3	33.5	33	33.5
4	44.5	44	44.5

vectorised method → array → 1D → vector
2D → matrix
3D → matrix
ND → N-dim matrix

previous

$y = mx + c$

New

multiple features [x1, x2]

$y = Mx + C$

multiple m

only one intercept

$y = [m_1 x_1 + m_2 x_2] + c$

M

X

Matrix Multiplication:

condition: Column of Matrix 1 = Row of Matrix 2

Rule of output shape: A(R) × B(C)

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \quad \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} = B$$

output → 2×2 matrix

2×3 → R c

3×2 → R c

C(3) = R(3)

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \quad \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix}$$

1×7 + 2×9 + 3×11 → 58

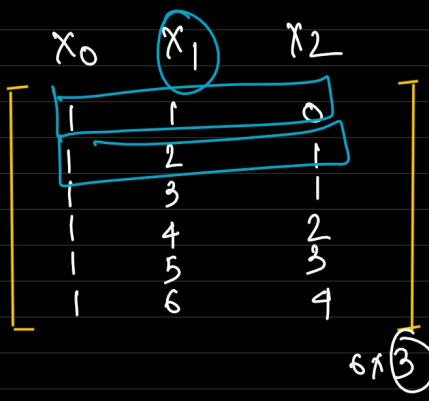
1×8 + 2×10 + 3×12 → 64

4×7 + 5×9 + 6×11 → 139

4×8 + 5×10 + 6×12 → 154

$$\begin{bmatrix} 58 & 64 \\ 139 & 154 \end{bmatrix}$$

2×2



$[m_0 \quad m_1 \quad m_2]$ → 1×3

↓ Transpose → 3×1

$\begin{bmatrix} m_0 \\ m_1 \\ m_2 \end{bmatrix}$ → output shape : 6×1

3×1

Python :

$X @ B$

$[6 \times 1) \leftarrow np.dot(X, B)$

intercept c

$x_0 m_0 + x_1 \times m_1 + x_2 \times m_1$

Gradient Descent :—

Years_of_exp (x1)	Completed_projects (x2)	Salary_lk (y)
1	0	39.761
2	1	48.900
3	1	56.938
4	2	68.240
5	3	77.867
6	4	85.022

$m_1 \times x_1 + m_2 \times x_2 + m_0 \rightarrow \hat{y}$ $\quad y$

$\sum_{i=1}^{N}$ error → $(y - \hat{y})$ → Total loss

↑ cost function

$m_1 = 0, \; m_2 = 0, \; m_0 = 0$

y	$\hat{y}$	error
39.761	0	39.761
48.900	0	48.900
56.938	0	56.938
68.240	0	68.240
77.867	0	77.867
85.022	0	85.022

add → $\frac{1}{n} \sum (y - \hat{y})^2$

→ 200 loss $(m_1 = 0, \; m_2 = 0, \; m_0 = 0)$

→ new loss $(m_1 = 0.01, \; m_2 = 0.1, \; m_0 = 0.1)$

for each value of parameter we will get different loss

if we plot loss w.r.t m_1, m_2, m_0

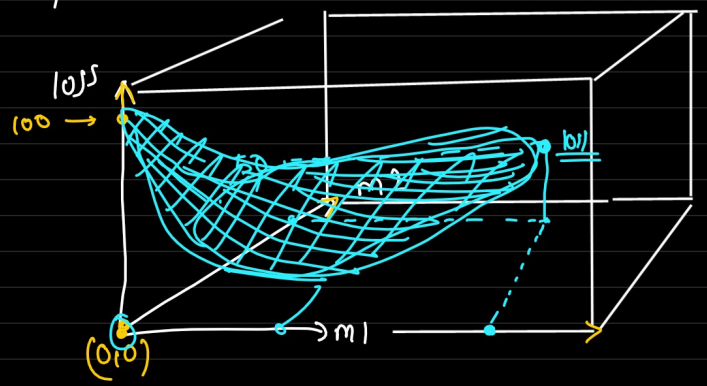
Set 1 $(m_0, m_1, m_2) \rightarrow 100$

Set 2 $(m_0, m_1, m_2) \rightarrow 20$ ⟶

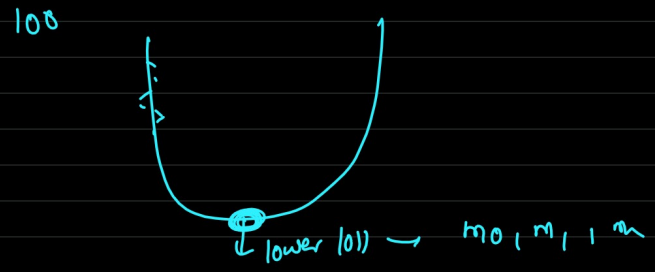
Set 3 $(m_0, m_1, m_2) \rightarrow 400$

Set 4 $(m_0, m_1, m_2) \rightarrow 80$

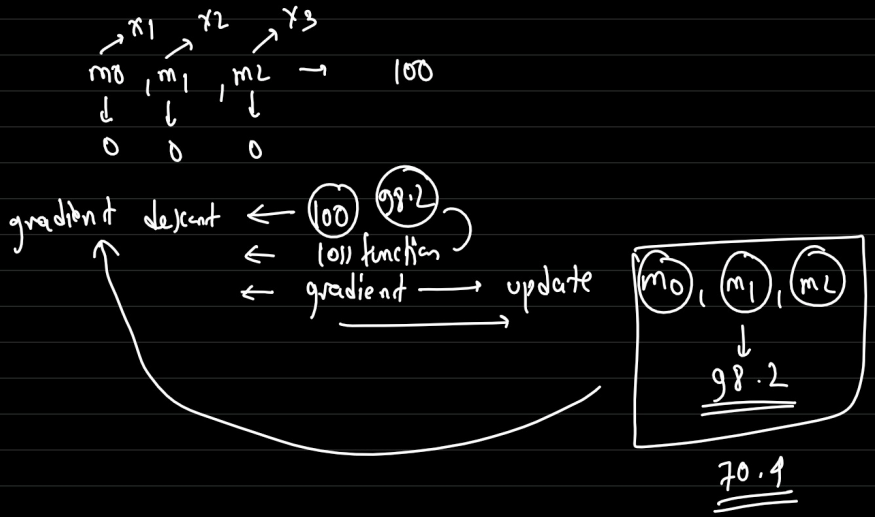
C



Ideally we want to have lower loss → m_0, m_1, m_2



Gradient Descent is an optimization algo used in ML. Its a iterative method to adjust parameter of a model to find minimum loss of a function.



1	11.5
2	22.5

m = 1, m = 2, ... m = 11 → ≈ 0 (-ve / +ve direction; how much! 0.5, 1)

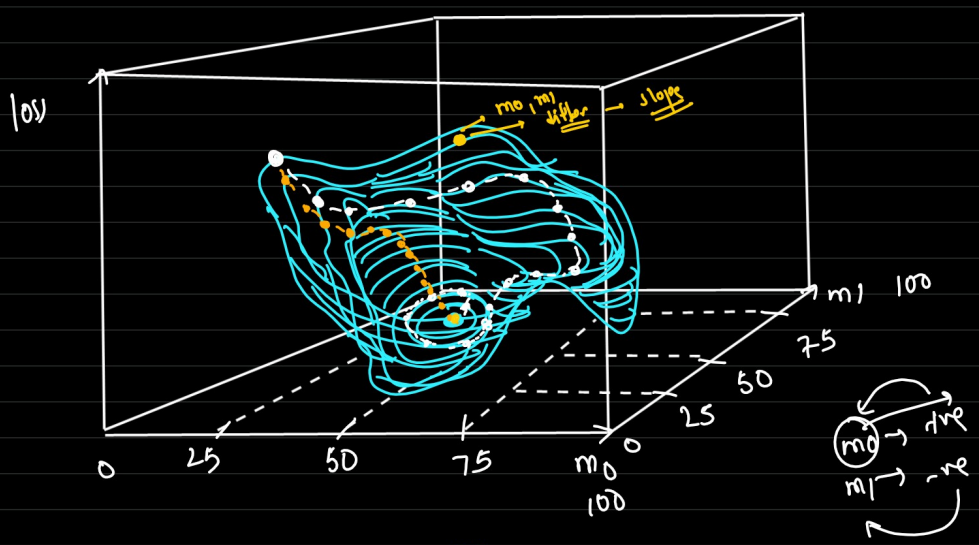
m = 1
m = 100]

x_1	m	y	$\hat{y}$
1 x	1	11.5	1
2 x	1	22.5	2

10.5 + 20.5 → 33 → loss → 28

1 x 2	11.5	2
2 x 2	22.5	4

reduce loss, for lowest loss get parameters.



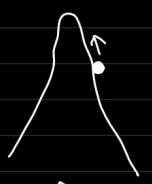
loss ≈ 0 → $m_0 \to 25$
$m_1 \to 75$

Gradient Descent: Goal is to find minimum loss

→ we use differentiation to calculate slope of the cost function at our current position.

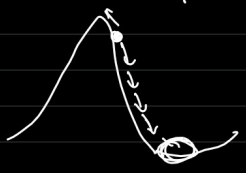
→ slope / gradient will tell us direction of the steepest ascent. (uphill)

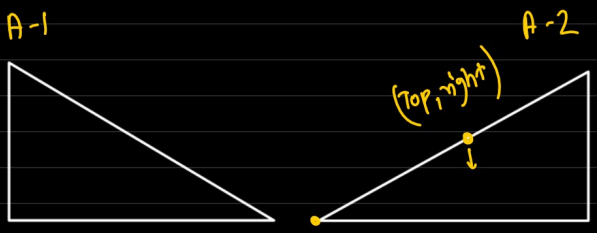
→ Take a step in the opposite direction. steepest descent. (downhill)

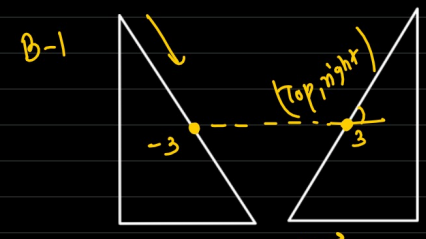


steepest ascent (uphill)

direction of valley = opposite of steepest ascent







B-2 has higher slope

left to right upwards → +ve slope

left to right downwards → -ve slope

Differentiation :

$$BMI = \frac{w}{h^2}$$

parameter → w, h

height = 1.8

→ change weight, keeping the height of constant.

$$\frac{\partial Bmi}{\partial w} = \frac{1}{h^2} = \frac{1}{(1.8)^2} \approx 0.309$$

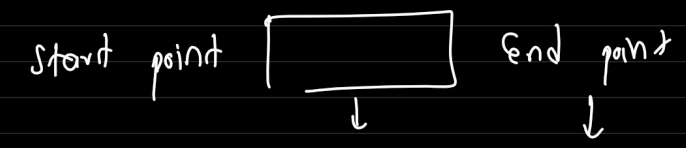
$w = 100$
$w = 101$] +1kg

$BMI = 1 \times 0.309 \rightarrow +.309$

$w = 100$
$w = 98$] (-2kg) $= -2 \times (.309) \rightarrow -0.618$

If i change my parameter by a small value, Derivate tells you how much the output changes per unit

Tells us : What will be the rate of change if input changes



Differentiation of the gradient:

$J \rightarrow$ cost function
$\beta \rightarrow$ parameter $(m_0, m_1, \ldots m_n)$

$$J(\beta) = \frac{1}{2m}(y-\hat{y})^2 \rightarrow \text{start point}$$

$q^2 \rightarrow q \times q$
$A = [1, 2, 3]$
$A^2 \rightarrow A \times A^T$

$$= \frac{1}{2m}(y - X\cdot\beta)^2$$

$$= \frac{1}{2m}(y - X\cdot\beta)^T(y - X\cdot\beta)$$

$\frac{1}{2m}$: keep this constant aside

$(y - X\cdot\beta)^T(y - X\cdot\beta)$: differentiation

$$S(\beta) = (y - X\beta)^T(y - X\beta)$$

Rule 1: $(A - B)^T = A^T - B^T$

Rule 2: $(AB)^T = B^T \cdot A^T$

$$S(\beta) = (y^T - (X\beta)^T)(y - X\beta) \rightarrow \text{Rule 1}$$

$$S(\beta) = (y^T - \beta^T X^T)(y - X\beta) \rightarrow \text{Rule 2}$$

$(a-b)(c-d) \rightarrow ac - ad - bc + bd$

$$= y^T y - y^T X\beta - \beta^T X^T y + \beta^T X^T X\beta$$

$\beta^T X^T y \rightarrow y^T X\beta$

simplify: since $y^T X\beta$ is a scalar value, it is equal to its transpose

$= \beta^T X^T y$

$$S(\beta) = y^Ty - 2y^TX\beta + \beta^TX^TX\beta$$

Take the gradient: $\frac{\partial S}{\partial \beta}$, take partial derivative.

$$\frac{\partial}{\partial \beta}(y^Ty) = 0 \qquad \frac{\partial}{\partial x}(1) = 0$$

$$\frac{\partial}{\partial \beta}(-2y^TX\beta) = -2(X^Ty)^T\beta = -2X^Ty \qquad \frac{\partial}{\partial x}(a^Tx) = a$$

$$\frac{\partial}{\partial \beta}(\beta^TX^TX\beta) = (X^TX + (X^TX)^T)\beta \qquad \frac{\partial}{\partial x}(x^TAx) = (A + A^T)x$$

β^T: X^T, $X\beta$: $A\ x$, $(X^TX)^T$: $X^T \cdot (X^T)^T$, $(X^T)^T$: X, so $X^T \cdot X$

$$= (X^TX + X^TX)\beta$$
$$= 2X^TX\beta$$

Combine:

$$\frac{\partial S}{\partial \beta} = 0 - 2X^Ty + 2X^TX\beta$$

$$= 2x^T x\beta - 2x^T y$$

$$= 2x^T(x\beta - y)$$

$$= 2x^T(\hat{y} - y)$$

bringing the constant :

$$= \frac{1}{2m} \cdot 2x^T(\hat{y} - y)$$

$$= x^T(\hat{y} - y)/m$$

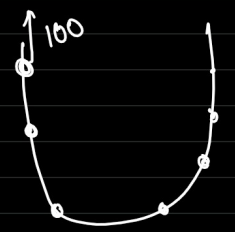
final derivation of our cost function ↓ gradient

```
gradient = (X.T @ (y_pred - y)) / m
```

= direction & rate of change x 0.1

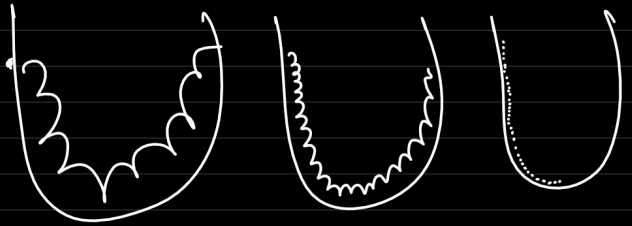
100

learning rate



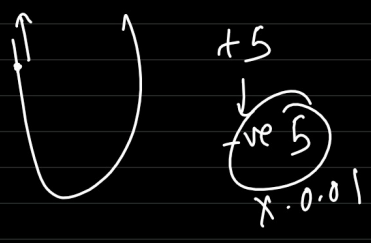
→ 100 r/. depth → start, end, 0.1

→ direction



$\beta \rightarrow$ matrix $\rightarrow$ learnable parameter

$$\beta_{new} = \beta_{old} - (lr * gradient)$$



$lr \rightarrow$ learning rate