"Gradient Descent" (GD)

Gradient descent is a first order iterative optimization algorithm for finding a local minimum of a differentiable function.

GD can find the minima of any function if it is differentiable at all points.

Let's take the loss function of Linear Regression

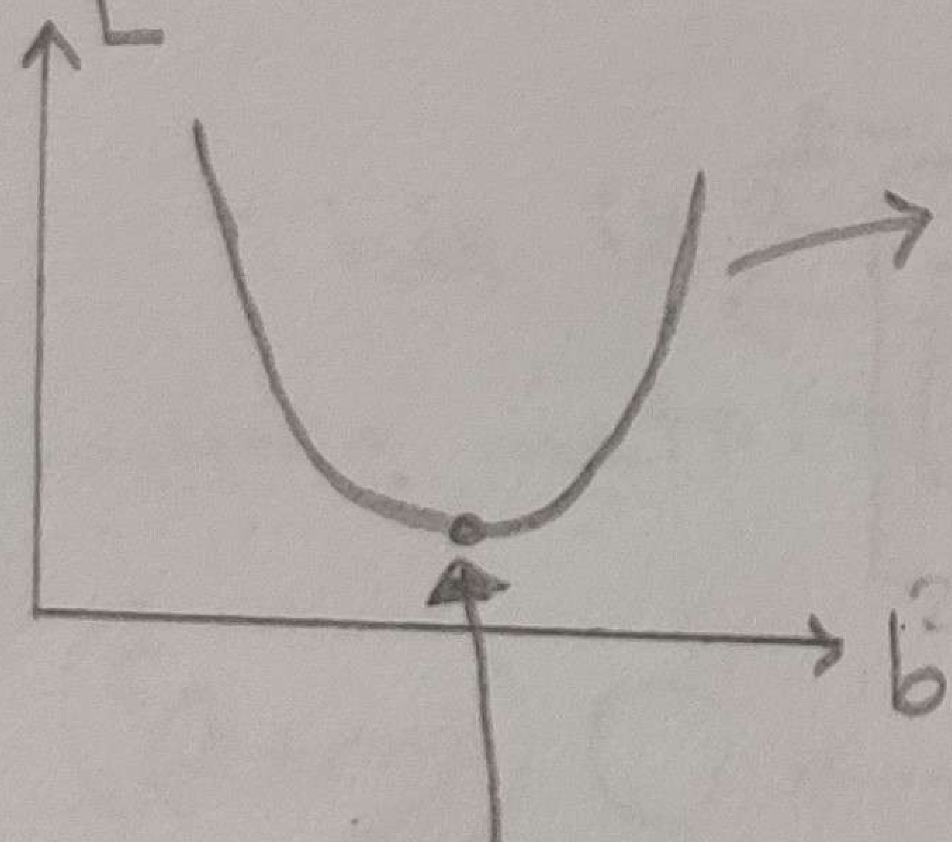
$$L = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$ → using SSE as Error Function

$$L(m, b) = \sum_{i=1}^{n} (y_i - m x_i - b)^2$$ → here m, b is the controller

let's say we already know the value of $m = 78.35$

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

now relation of L and b is $\boxed{L \rightarrow b^2}$ → Parabola



→ not exact but similiar

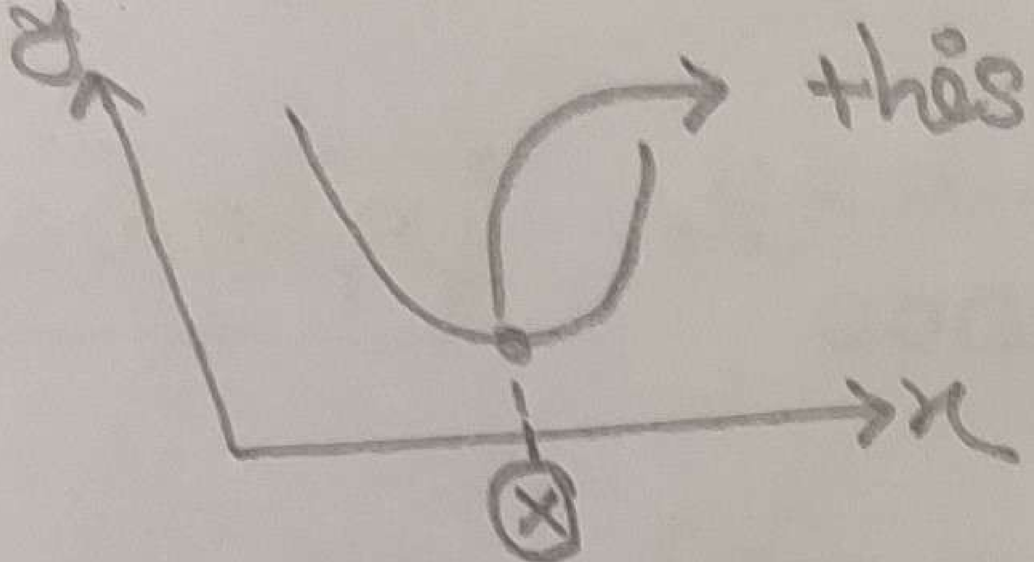
This is the point b where L is minimum

To find the minimum point of b we can do $\boxed{\frac{dL}{db} = 0}$ → using OLS

we can't do it in higher dimension → costly

How gradient descent works?

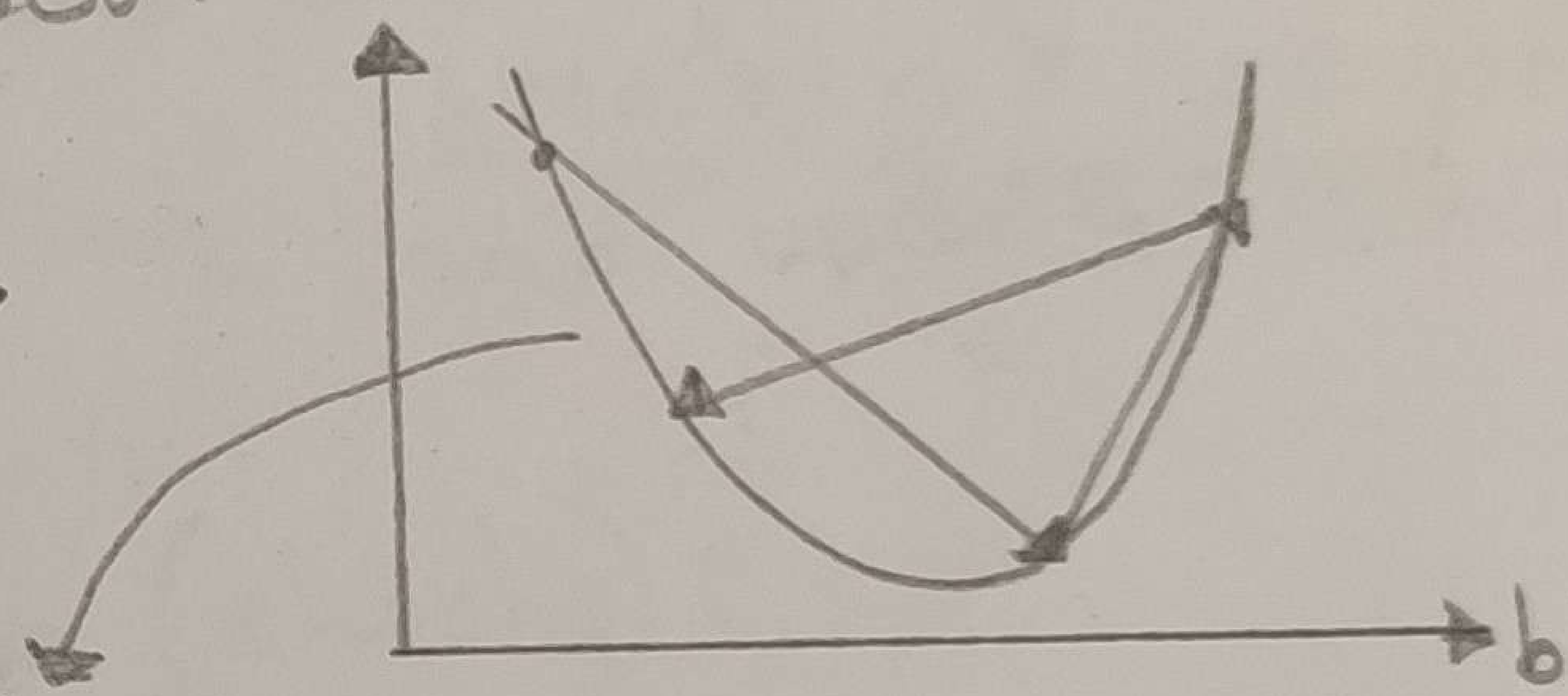
For a single parameter gd always try to find the lowest y value for the value of x.



this the min value of y. GD try to find what's the value of x there.

(ii) to do that we find the slope at this point. If we got -ve slope we move right in the x axis.
+ve slope we move left in the y axis.

$$\boxed{x_{new} = x_{old} - slope}$$



the movement of x value changes a lot. For this it may happens that we lost the min point/never reach the min point

To make it stable we multiple a learning rate/movement rate with the slope.

$$x_{new} = x_{old} - \eta\, slope$$

learning rate → usual value 0.01. may vary

when to stop the iteration:

(i) if: $x_{new} - x_{old} \to 0$ / $x_{new} - x_{old} < 0.0001$

(ii) second approch → limited iteration → 500/1000....
epochs

Mathematical Formation: let's assume we already know the value $m = 78.35$. Now find the value of b so that lost function become minimum.

$\boxed{\text{Step-1}}$ set a random value for $b = 0$, $\boxed{\eta = 0.01}$

for i in range(epochs): $\rightarrow 1000$

$b_{new} = b_{old} - \eta \boxed{\text{slope}}$ we need to find the slope at current b

$\boxed{\text{Step size} = \eta \times \text{slope}}$

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

$\boxed{\text{slope}}$:

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

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

Universality Gradient Descent:

In ML, we always want to minimize the error. So For ML algorithm if we have a loss function and the loss function is differentiable at any point then we can apply gradient descent to implement these algorithm no matter what's algorithm is.

Gradient Descent with 2 variable (m, b):

$\boxed{\text{Steps}}$ as now we have 2 variable so we need find the value of m and b so that value of Loss Function become minimum.

$\boxed{\text{Step-1}}$ set a random value of m and b

$m = 0 \;,\; b = 0$

$\boxed{\text{Step-2}}$ Take a good ephochs and learning rate (η)

$\boxed{\text{Step-3}}$ Perform the iteration and update the m and b.

for epoch in range (ehochs):

$m = m - \eta * \text{slope}$ (1st derivate with respect to m)

$b = b - \eta * \text{slope}$ (1st derivative with respect to b)

$\boxed{\text{Loss function}}$

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

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

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

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

$$= -2 \sum_{i=1}^{n} (y_i - \hat{y}_i) . x_i = 2 \sum_{i=1}^{n} \text{error} . x_i$$

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

$$= -2 \sum_{i=1}^{n} \underbrace{(y_i - \hat{y}_i)}_{\boxed{\text{error}}}$$

Types of Gradient Descent: 3 types.

(i) Batch GD

(ii) Stochastic GD

(iii) Mini- Batch GD

Batch Gradient Descent:

$$m = m - \text{learningRate} * (\text{slope})_m$$

$$b = b - \text{learningRate} * (\text{slope})_b$$

to find the slope each time we look at the full data and then update m, b one time

For this reason Batch GD becomes slow.

→ used rarely

→ used when loss function is a convex function.

Mathmatical Formation of Batch GD on n dimensions:

lets take a 3 col data (3D)

$$y = \beta_0 + \beta_1 X_1 + \beta_2 X_2$$

$\boxed{\text{Step-1}}$ initialize Beta's with random values

$$\beta_0 = 0 \;,\; \beta_1 = \beta_2 = \dots \beta_n = 1$$

$\boxed{\text{Step-2}}$ take ephochs and learning rate (lr)

$\boxed{\text{Step-3}}$ Find slope with respect to every β then update beta's

$$\beta_0 = \beta_0 - lr * \text{slope} \rightarrow \frac{\partial L}{\partial \beta_0}$$

$$\beta_1 = \beta_1 - lr * \text{slope} \rightarrow \frac{\partial L}{\partial \beta_1}$$

$$\beta_2 = \beta_2 - lr * \text{slope} \rightarrow \frac{\partial L}{\partial \beta_2}$$

Loss Function:

$$L(\beta_0, \beta_1, \beta_2)$$

If columns was n we need to find $(n+1)$ slopes.
So all update of β's will be done inside a loop.

Loss function: (MSE) $L = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$ $n \rightarrow$ number of rows.

Let's take we have only $n=2$ row and cols $= 2 + 1$
expand the loss function for this example.

$2 \uparrow$ Input col, $1 \rightarrow y$

$$L = \frac{1}{2}\left[(y_1 - \hat{y}_1)^2 + (y_2 - \hat{y}_2)^2\right]$$

$$L = \frac{1}{2}\left[(y_1 - \beta_0 - \beta_1 x_{11} + \beta_2 x_{12})^2 + (y_2 - \beta_0 - \beta_1 x_{21} - \beta_2 x_{22})^2\right]$$

	x_1	x_2	y
x_{11}	8.1	93	3.2 $\rightarrow y_1$
x_{21}	7.5	95 ($\uparrow x_{22}$)	3.5 ($\uparrow y_2$)

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$
$$\hat{y}_1 = \beta_0 + \beta_1 x_{11} + \beta_2 x_{12}$$
$$\hat{y}_2 = \beta_0 + \beta_1 x_{21} + \beta_2 x_{22}$$

$$\frac{\partial L}{\partial \beta_0} = \frac{1}{2}\left[2(y_1 - \beta_0 - \beta_1 x_{11} - \beta_2 x_{12})(-1) + 2(y_2 - \beta_0 - \beta_1 x_{21} - \beta_2 x_{22})(-1)\right]$$

$$\frac{\partial L}{\partial \beta_0} = \left[(y_1 - \hat{y}_1) + (y_2 - \hat{y}_2)\right] * \frac{-2}{2}$$

we take only 2 input rows so it get's 2. If we had n rows as input then it will be $n/2$

For n rows as input/train data

$$\frac{\partial L}{\partial \beta_0} = \frac{-2}{n}\left[(y_1 - \hat{y}_1) + (y_2 - \hat{y}_2) + \cdots + (y_n - \hat{y}_n)\right]$$

$$\boxed{\frac{\partial L}{\partial \beta_0} = \frac{-2}{n} \sum_{i=1}^{n} \underbrace{(y_i - \hat{y}_i)}_{\text{error}}} \rightarrow \frac{\partial L}{\partial \beta_0} = -2 * \text{mean(error)}$$

For β_1 slope: $(n=2)$

$$\frac{\partial L}{\partial \beta_1} = \frac{1}{2}\left[2(y_1-\beta_0-\beta_1 x_{11}-\beta_2 x_{12})(-x_{11}) + 2(y_2-\beta_0-\beta_1 x_{21}-\beta_2 x_{22}) * (-x_{21})\right]$$

$$= \frac{-2}{2}\left[(y_1-\hat{y}_1)(x_{11}) + (y_2-\hat{y}_2)(x_{21})\right]$$

If n rows then

$$\frac{\partial L}{\partial \beta_1} = \frac{-2}{n}\left[(y_1-\hat{y}_1)(x_{11}) + (y_2-\hat{y}_2)(x_{21}) + \dots + (y_n-\hat{y}_n)(x_{n1})\right]$$

$$\frac{\partial L}{\partial \beta_1} = \frac{-2}{n}\sum_{i=1}^{n}(y_i-\hat{y}_i)(x_{i1})$$ First col and n rows data

$$\frac{\partial L}{\partial \beta_2} = \frac{-2}{n}\sum_{i=1}^{n}(y_i-\hat{y}_i)(x_{i2})$$

now if we have m colums and n rows

$$\frac{\partial L}{\partial \beta_m} = \frac{-2}{n}\sum_{i=1}^{n}(y_i-\hat{y}_i)(x_{im}) = \frac{-2}{n}(\text{error}.\ X_{im})$$

we have to find the derivate of all coef in 1 step. without using loop.

In our case x_train → 353 × 10 y_train = 353 × 1

↓

y_Pred = 353 × 1

Error = 353 × ~~1~~ 1

$x\_train^T = 10 \times 353$

$(x\_train)^T .\ \text{Error} = (10 \times 353).(353 \times 1)$

$= (10 \times 1) \longrightarrow$ 1 col matrix with 10 beta values.

Stochastic Gradient Descent:

In Batch GD we go through the whole data update once.
But in Stochastic GD we just look on a single row then update the coefficent once. This make it more fast on big dataset.

in a single epoch/iteration we update coefs n times. So in 10-20 epochs we converge a lot and almost reach the answer.

↑
number of rows

As we update coefs just by seeing 1 row so no need to load the full data on ram. So memory efficient.

→ Don't give stable answer [not the best answer]

→ In Batch GD in every update value improves. But in Stochastic in some step it improves and some times it gets more bad. But after a period of time it reached near the answer.

When to use Stochastic GD

(i) For large dataset

(ii) If we have non-convex function

Mini-Batch Gradient Descent:

we create a few batches [a group of rows]

If we have m batches then in 1 epoch iteratio[n] coef_ and intercept_ gets updated m times

If m = 1 → batch GD

If m = num of rows → Stochastic GD

Learning Schedule:

For the randomness of stochastic GD is good to escape the local minima but bad because it means the model may not set at minimum.

To prevent this behaviour we used to dilemma is gradually reduce the learning rate.

lr ↑ ⟶ Large jump/step at each iteration

lr ↓ ⟶ Small " " " "

AS it get's close to the minima, lr become smaller and smaller to make sure it doesn't cross the minima. The process is called "simulated annealing"

The function that determines the learning rate at each iteration is called Learning Schedule.

If lr is reduced too quickly, we may stuck in a local minima./or ends up frozen halfway to the minima. If lr is reduced too slowly, we may jump around the minima for a long time.

Function

$t_0 = 5, \; t_1 = 50$ ⟶ random

return $t_0 / (t_1 + t)$

$t = \text{epoch} * m + i$

m ⟶ total rows

i ⟶ inner loop variable