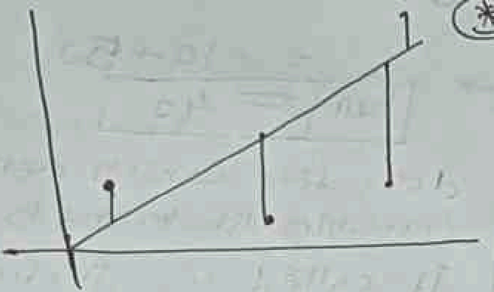
* Gradient descent *

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

  Optimization mean best performance achive.

* intution :- we want best fit line



* we try to make minimum loss function (value of)

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

$$\boxed{\hat{y}_i = mx_i + b}$$

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

This formula is depends on m & b → $L(m, b)$

→ mean L is function depend on (m, b)

Data

cgpa	Package
-	-
-	-
-	-
-	-

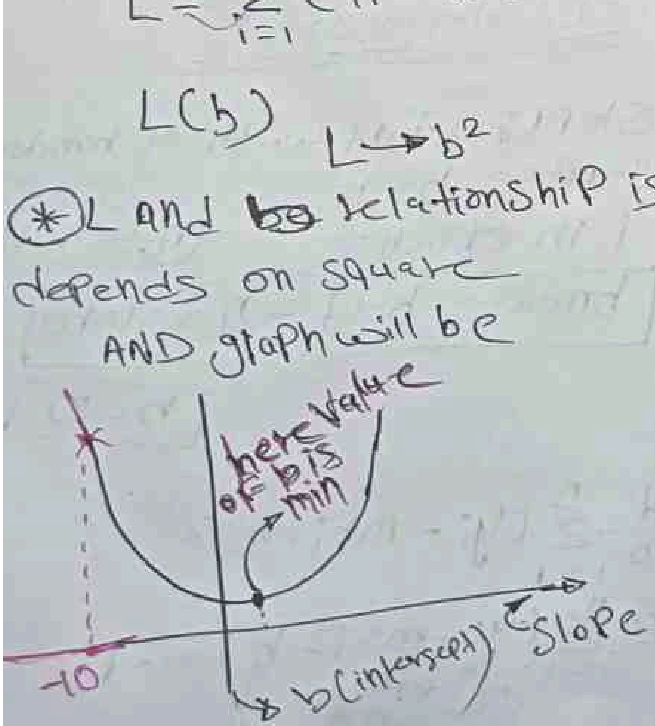
* Suppose our data m value is known to us which is

$$\boxed{m = 78.35}$$

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

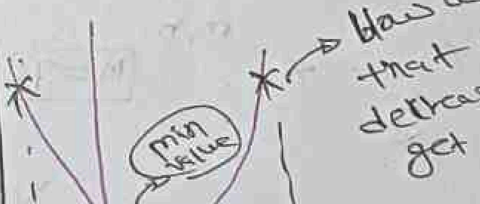
$L(b)$ $L \rightarrow b^2$

* L and b relationship is depends on square AND graph will be



* step 1: Select random value of b_{min}

$b = -10$

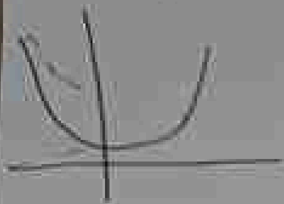


→ How would I know that we want to decrease the value to get min value of b

Ans is slop finding

* understanding of slop
* slop: It is how line is tilted
* differentiation is a tool to find out slop when graph of function is curve

Ex: $y = x^2$



By differentiating

$$\boxed{\frac{dy}{dx} = 2x}$$

$Slope(x) = 2x$

$x = 1$ slope $= 2$

$x = 2$ slope $= 4$

$x = -1$ slope $= -2$

* Very drastic changes in bnew values

To avoid this we multiply by learning rate (η)

Our equation becomes

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

* Slope value will tell us where is minimum value of b

* If slope is -ve then go forward
* If slope is +ve then go backward

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

$b_{old} = -10$ → assume slope

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

$= -10 + 50$

$$\boxed{b_{new} = 40}$$

derivative which depends on two variables like m and b both then it called as gradient or approximate gradient

Q When to stop?

* bnew - bold → is becomes very small (0.0001) then we will stop
* Iteration → 1000, 100, → epochs

* Mathematical formulation

$\boxed{m = 78.35}$

* Gradient descent In terms of b

Step 1: Start with a random value suppose $b = b'$

for i in epochs: $\boxed{\eta = 0.01}$

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

$\boxed{b = 0}$

Here we find value of slope

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

$$\frac{dL}{db} = \frac{d}{db}\left(\sum_{i=1}^{n} (y_i - \hat{y}_i)^2\right)$$

It means if we change b then how much change occurs in L

$$\boxed{\hat{y}_i = mx_i + b}$$

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

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

no b term it become 0 → -1

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

This is equation of Slope

for $b=0$ (assume) $m = 78.35$

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

Slope(b=0)

Code Time:-

This is use to make dataset

```
From sklearn.datasets import make_regression
import numpy as np
```

rows shape of x output feature

```
X, y = make_regression(n_samples=4, n_features=1, n_informative=1
                        , n_targets=1, noise=80, random_state=13)
```

how much output

Randomness NOT perfect linear data

```
import matplotlib.pyplot as
                    plt.
   plt.scatter(x, y)
```

```
From sklearn.linear_model import LinearRegression
reg = LinearRegression()
reg.fit(X, y)
reg.coef_
```

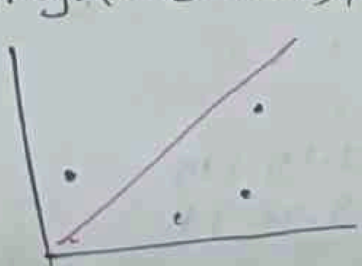
min slope (m)

output → 78.35

```
reg.intercept_
```

→ 26.1596 → min intercept value (b)

```
plt.Scatter(X, y)
plt.plot(X, reg.predict(x), color='red')
```



#Lets apply Gradient Descent by assuming slope is constant
m = 78.35

And let's assume the starting value of intercept b=0

```
ypred = ((78.35 * x) + 0).reshape(4)

m = 78.35
b = 0
loss_slope = -2 * np.sum(y - m * x.ravel() - b)
loss_slope
```

xslope = loss_slope

Now we want to do bnew = bold - η xslope

```
η = 0.01
b = 0

Stepsize = η * loss_slope

b = b - Stepsize
b
```

$\boxed{\rightarrow 20.92}$

Now we are making class to calculate only b for Now

```
from sklearn.datasets import make_regression
import matplotlib.pyplot as plt
import numpy as np
X, y = make_Regression(n_samples=100, n_feature=1, n_informative=1,
                       n_targets=1, noise=20)

plt.scatter(x, y)
```

#Actual result for comparing our result after making own class to calculate b.

```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(x, y)
print(lr.coef_)
print(lr.intercept_)
```

output:-
[29.19] m
[-3.35] b

m = 29.19

~~steps~~ ~~n-def GD~~

```
class GDRegressor:
    def __init__(self, learning_rate, epochs):
        self.m = 29.19
        self.b = -120
        self.lr = learning_rate
        self.epochs = epochs
    def fit(self, x, y):
        # calculate the b using GD
        for i in range(self.epochs):
            loss_slope = -2 * np.sum(y - self.m*x.ravel() - self.b)
            self.b = self.b - (self.lr * loss_slope)
        print(self.b)

gd.GDRegressor(0.01, 100)
```

Effect of learning rate

$\eta = 0.02$ $\eta = 0.1$ $\eta = 0.5$

if very low
then very slow
and more epochs

The universality of Gradient Descent

Now How to calculate m & b

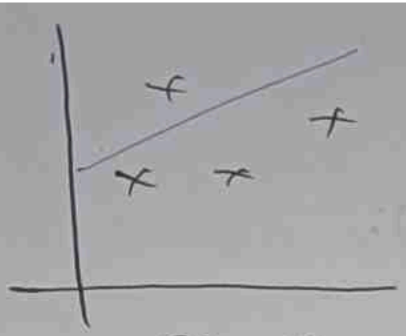
Step 1:- initiate random vals for m and b

m = 1 and b = 0

2. epochs = 100 lr = 0.01

for i in epochs:

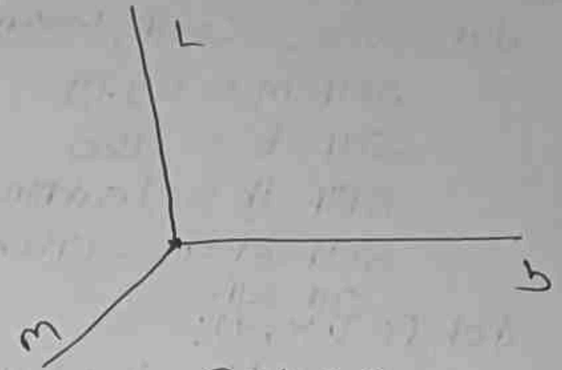
$$\left.\begin{aligned} b &= b - \eta\,\text{slope} \\ m &= m - \eta\,\text{slope} \end{aligned}\right\}\ \text{both slopes are different}$$



$F(x, y)$

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

$L(m, b)$



$F(m, b)$

$\boxed{\begin{array}{l} b\_slope = \frac{\partial L}{\partial b} \\ m\_slope = \frac{\partial L}{\partial m} \end{array}}$

$\sum (y_i - mx_i - b)^2$

$\frac{\partial L}{\partial b} = 2 \sum (y_i - mx_i - b)$

$\boxed{Slope_b = -2 \sum (y_i - mx_i - b)}$

$\boxed{= Slope\_b \text{ at } b = 0}$

$\frac{\partial L}{\partial m} = 2 \sum (y_i - mx_i - b)$

$2 \sum (y_i - mx_i - b)$

$-x_i$

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

$\boxed{Slope\_m \text{ at } m = 1}$

* Improvement In code *

Now data is little bit change

X, y = make_regression(n_samples=100, n_features=1, n_informative=1, n_targets=1, noise=20, random_state=13)

```
from sklearn.linear_model import LinearRegression
  lr = LinearRegression()
  lr.fit(x,y)
  print(lr.coef_) #value of m
  print(lr.intercept_) #value of b

→ [27.8280] #m
  -2.2947    #b

 # This is calculating to Tally the ans with our class
from sklearn.model_selection import cross_val_score
  np.mean(cross_val_score(lr, x,y, scoring='r2', cv=10))

class GDRegressor:
    def __init__(self, learning_rate, epochs):
        self.m = 100
        self.b = -120
        self.lr = learning_rate
        self.epochs = epochs

    def fit(sell, x, y):
        for i in range(self.epochs):
            loss-slope-b = -2 * np.sum(y - self.m * x.ravel()
                                        - self.b)

            loss_slope_m = -2 * np.sum((y - self.m * x.ravel()
                                        - self.b) * x.ravel())

            self.b = self.b - (self.lr * loss_slope_b)
            self.m = self.m - (self.lr * loss_slope_m)

        print(self.m, self.b)
  gd = GDRegressor(0.001, 100)
  gd.fit(x, y)
```

$$\underbrace{27.8280}_{m} \quad \xrightarrow{b} -2.254$$

Also making function to Predict

```
def Predict(self, x)
    return self.m * x + self.b

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_
                                    size = 0.2, random_state = 2)

y_Pred = lr.Predict(x_test)
from sklearn.metrics import r2_score
r2_score(y_test, y_Pred)
```

→ 0.63 → this is ~~before~~ before class

Now

```
gd.fit(x_train, y_train)
y_Pred = gd.Predict(x_test)
from sklearn.metrics import r2_score
r2_score(y_test, y_Pred)
```

→ 0.63 → our class result

(*) Impact on gradient Descent of Hyperparameter, loss Func^n & Data.

1. learning rate
2. loss Function
3. Data

1. learning rate:-
   - low LR: • If LR is 0.002
     - very small steps are geting
     - more epoch value
     - not much optimize

③ Moderate LR: ⊙ Give optimize result
(0.2) ⊙ Not want more number of epochs
⊙ Get faster result

⊙ High LR: ⊙ It get bigger and bigger step
0.80
⊙ And some-times it may not converge to minimum value of loss.

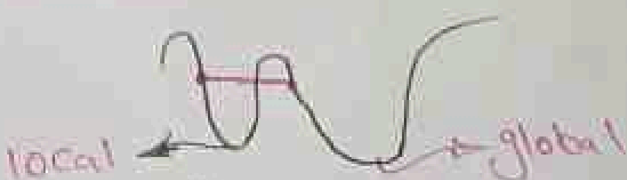
② Effect of loss function:

⊙ $\boxed{L = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2}$ → this loss function is convex function

⊛ Now what is convex function
⊙ If we draw a line between two points that line never cross the function that function is called convex function.

Ex:

Non convex Ex:



Non convex

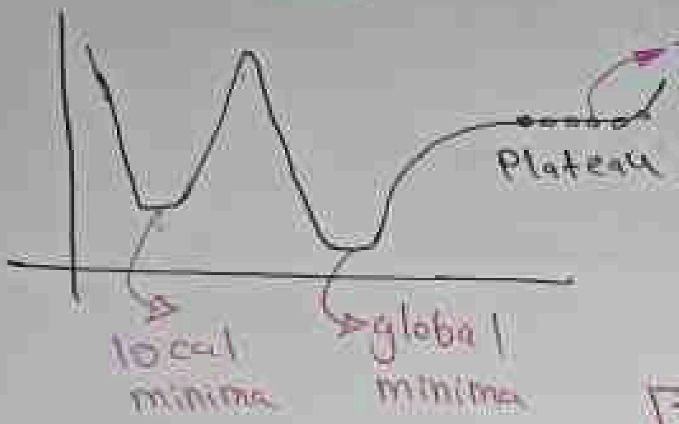
⊙ In convex function there is only one minima and that is global minima

⊙ In Non convex function there is more than one minima we called them global minima and local minima

In Non convex function there are two Problems

① • local minima is here :- → our algo. converge here and we should not go anyware.
• more than 2 minima
→ Solution may be converge on local minima.

② Plateau :-



To get out from Plateau there are we want to increase our epochs

* And more time taken to train

In 3D Plateau become Saddle Point

Only Name Change idea is Same

* Effect of Data :

• If data is well scaled then our algorithm will fast converge towards the minima.

Types of gradient descent

Gradient Descent
- Batch GD
- Stochastic GD
- minibatch GD

$$\left.\begin{aligned} m_n &= m_0 - \eta \times (\text{slope})_{m=0} \\ b_n &= b_0 - \eta \times (\text{slope})_{b=0} \end{aligned}\right\}$$ We update the value of m & b by viewing all data

This is batch GD.

Stochastic GD

- In this we update the values of m & b by viewing single row
- This is fast
- suitable for big/large data

mini batch GD

- Fixed Batch size
- If data is of 300 rows and batch size=30 our values of m & b will change after 30 rows not after full data.

* We already learn batch gradient descent to understand the ~~to~~ gradient descent for two variables

we learn for:

cgpa	lpa

Now we are learning Batch gradient descent for more than two variable

cgpa	IQ	gender	CPa

$$\boxed{y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3}$$

* understand the equation *

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3$$

β_0: Base Salary same as c

β_1: Effect of cgpa on Salary

x_1: CGPA

β_2: Effect of IQ on Salary

x_2: IQ

β_3: Effect of gender on Salary

x_3: gender

* N Dimensional Data *

$$\{\beta_0, \beta_1, \beta_2 \ldots \beta_n\}$$

Values required to find the value of y Predict

* Mathematical formulation *

cgpa | iq | lpa

x_1	x_2	y
8.1	93	3.2
7.5	95	3.5

$$\underset{(lpa)}{y} = \beta_0 + \beta_1 \underset{(cgpa)}{x_1} + \beta_2 \underset{(lpa)}{x_2}$$

Step 1: Start with random value

generally $\beta_0 = 0$, $\beta_1 \beta_2 = 1$

Step 2:- epoch = 100, lr = 0.1

Intercept

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

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

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

$$\boxed{L(\beta_0, \beta_1, \beta_2)}$$

$$\boxed{L = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \quad \{rows = 2,\ cols = 2+\}$$

(MSE)

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

$$\hat{y}_i = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

$$\boxed{\hat{y}_1 = \beta_0 + \beta_1 x_{11} + \beta_2 x_{12}}$$

$$\boxed{\hat{y}_2 = \beta_0 + \beta_1 x_{21} + \beta_2 x_{22}}$$

x1	x2	y
8.1	9.3	3.2
7.5	9.5	3.5

x_{12} y_1 y_2 x_{11} x_{21} x_{22}

$$= \frac{1}{2}\left[(y_1 - \underbrace{\beta_0 - \beta_1 x_{11} - \beta_2 x_{12}}_{\hat{y}_1})^2 + (y_2 - \beta_0 - \beta_1 x_{21} - \beta_2 x_{22})^2\right]$$

$$\frac{\partial L}{\partial \beta_0} = \frac{1}{2}\left[2(y_1 - \hat{y}_1)(-1) + 2(y_2 - \hat{y}_2) - 1\right]$$

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

for N rows

$$\frac{\partial L}{\partial \beta_0} = \frac{-2}{n}\left[(y_1 - \hat{y}_1) + (y_2 - \hat{y}_2) + (y_3 - \hat{y}_3) + \dots + y_n - \hat{y}_n\right]$$

$$\boxed{\frac{\partial L}{\partial \beta_0} = \frac{-2}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)}$$

This is give the value of slope

$$\beta_0 = \beta_0 - \eta(\text{slope})$$

this value is calculated by $\frac{\partial L}{\partial \beta_0}$

(*) Now calculating of β_1 slope (*)

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

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

$$L = \frac{1}{2}\left[(y_1-\beta_0-\beta_1x_{11}-\beta_2x_{12})^2 + (y_2-\beta_0-\beta_1x_{21}-\beta_2x_{22})^2\right]$$

$$\frac{\partial L}{\partial \beta_1} = \frac{1}{2}\left[2(y_1-\hat{y}_1)(-x_{11}) + 2(y_2-\hat{y}_2)(-x_{21})\right]$$

$$\boxed{\frac{\partial L}{\partial \beta} - \beta_1 x_{11} = -x_{11}}$$

$$\frac{\partial L}{\partial \beta_1} = \frac{-2}{n}\left[(y_1-\hat{y}_1)(x_{11}) + (y_2-\hat{y}_2)(x_{21}) + (y_3-\hat{y}_3)(x_{31}) + \dots + (y_n-\hat{y}_n)(x_{n1})\right]$$

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

x_1	x_2	y
8.1	93	3.2
7.5	95	3.5

x_{i1} is represent all values of column 1

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

* For m cols *

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

Class GDRegressor:
    def __init__(self, learning_rate=0.01, epochs=100):
        self.coef_ = (None)
        self.intercept_ = None
        self.lr = learning_rate
        self.epochs = epnochs.

    def fit(self, x_train, y_train):
        self.intercept_ = 0
        self.coef_ = np.ones(x_train.shape[1])
        for i in range (self.epochs):
            #updating all the coef & intercept

First understand the intercept_ formula.

$$\frac{\partial L}{\partial \beta_0} = \frac{-2}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)$$

This is nothing but our y_train

$$(y_i - \hat{y}_i) \rightarrow x$$

$$= \frac{-2}{n}\sum_{i=1}^{n} x$$

This is nothing but formula for calculating <u>mean</u>

understand calculation to finding y hat

x_1	x_2	x_3	y
-	-	-	-

for first row

$$\hat{y}_1 = \beta_0 + \beta_1 x_{11} + \beta_2 x_{12} + \beta_3 x_{13}$$

$$\hat{y}_2 = \beta_0 + \beta_1 x_{21} + \beta_2 x_{22} + \beta_3 x_{23}$$

$$\hat{y}_i = \beta_0 + [x_{11}\ x_{12}\ x_{13}]\begin{bmatrix}\beta_1\\ \beta_2\\ \beta_3\end{bmatrix}$$

we can calculate for all values in x train By using vectorization

$$\boxed{\hat{y} = \beta_0 + np.dot(coef\_, xtrain)}$$

Formula becomes $\boxed{\beta_0 + np.dot(x\_train, coef\_)}$

(353, 10) (10, 1)

xtrainshape coef shape (353, 1)

****

y-hat = np.dot(x-train, self.coef_) + self.intercept_

intercept_der = -2 * np.mean(y-train - yhat)

self.intercept_ = self.intercept_ - (self.lr * intercept_der)

Now time to calculate coef_

For single: $\frac{\partial L}{\partial \beta_1}, \frac{\partial L}{\partial \beta_2}, \frac{\partial L}{\partial \beta_n}$

~~Simul~~

How to find out this ~~at~~ same time?

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

$=$

x_1	x_2	y	$\hat{y}$
1	2	5	6
3	4	7	8

y $[5 \; 7]$ $\hat{y}$ $[6 \; 8]$

$y - \hat{y} = [-1, -1]$

$[-1, -1]\begin{bmatrix}1\\3\end{bmatrix}$

$[-4 + -4]$

-8

$$\boxed{\left[[y - \hat{y}] \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \right] \times \frac{-2}{n}}$$

$\frac{\partial L}{\partial \beta_1} \; \ldots \; \frac{\partial L}{\partial \beta_{10}}$

$= [(y_i - \hat{y}_i) \times \text{train}] \times \frac{-2}{n}$

$(353, 1)^T \quad (353, 10)$

$(1, 353) \quad (353, 10)$

$(1, 10) \times \frac{-2}{n} = (1, 10)$ → coef_der

```
coef_der = -2 * np.dot((y_train - y_hat), x_train)/
                                    x_train.shape[0]

self.coef_ = self.coef_ - (self.lr * coef_der)

def predict(self, x_test):
    return np.dot(x_test, self.coef_) + self.intercept_

gdr = GDRegressor(epochs=100, learning_rate=0.5)
gdr.fit(x_train, y_train)
```

152.0135 [14.3891 -173.72 491.54 323.91,
-39.32 -116.010. -194.04 103.38
451.63 97.57]

```
y_pred = gdr.predict(x_test)
r2_score(y_test, y_pred)
```

0.45

* Stochastic Gradient Descent *

* Problem with Batch GD *

n=1000

cols=5 → 6 coefs

epoch=50

50

1 → 1000

6 → 6000 → derivatives want to calculate

6000×50 = 300000

- It makes algorithm slow
  - → Deep learning
    - → CNN → Images } Both are high dimensional Data
    - → RNN → Text

* And In Deep Learning most of the time we are not use Batch GD

In coding part we did vectorization to calculate y-hat

y-hat = np.dot(x-train, self.coef_) + self.intercept_

In this we used numpy and avoid loop

* When we are calculating y-hat we load full x-train data load on RAM.

* And if data is very large then it could give (error) Not exactly error but system not support (Hardware Problem)

* To resolve this all problem we shifted toward Stochastic GD

* In Batch gradient descent value of coef. is change after checking full data. only one time changes

* In Stochastic we change value of coef. by Seeing/Checking a single row.

—— → row update

* In 1 single epoch n updates will do

- → faster convergen
- → row is select randomly
- → Not give stady (ans) soln

* Now, coding Part *

```
from sklearn.datasets import load_diabetes.
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score.
from sklearn.model_selection import train_test_split.

x,y = load_diabetes (return_x_y=True)

print(x.shape)
print(y.shape)
```

→ (442, 10)
→ (442,)

```
x,y = load_diabetes(return_x_y = True)

x_train, x_test, y_train, y_test = train_test_split(x, y,
                                   test_size=0.2, random_state=2)

reg = LinearRegression()
reg.fit(x_train, y_train)
print(reg.coef_) #m
print(reg.intercept_) #b
```

#m → [-9.16 -205.46 516.68 340.62 -895.54 561.21 153.88 126.73 861.121 52.41]

#b → 151.88

```
y-pred = reg.predict(x-test)
r2-score(y-test, y-pred)
```
→ 0.439938

Now we make our class to predict coefficient

```
Class SGDRegressor:
    def __init__(self, learning-rate=0.01, epochs=100):
        self.coef_ = None
        self.intercept_ = None
        self.lr = learning-rate
        self.epochs = epochs

    def fit(self, x-train, y-train):
        self.intercept_ = 0
        sef.coef_ = np.ones(x-train.shape[1])

        for i in range(self.epochs):
            for j in range(x-train.shape[0]):
                idx = np.random.randint(0, x-train.shape[0])
                y-hat = np.dot(x-train[idx], self.coef_)
                        + self.intercept_
```

In BGD

$$\frac{\partial L}{\partial \beta_0} = \frac{-2}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)$$

1

collectively

Not required in SGD

for SGD **

$$\boxed{\frac{\partial L}{\partial \beta_0} = -2(y_i - \hat{y}_i)}$$

```
                intercept-der = -2 * (y-train[idx] - y-hat)
                self.intercept_ = self.intercept_ - (self.lr
                                  * intercept-der)
```

Coef_der.

$$\boxed{\frac{\partial L}{\partial \beta_1} = \frac{-2}{n}\sum_{i=1}^{n}(y_i-\hat{y}_i)\,x_{i1}}$$

In BatchGD

$$\boxed{\frac{\partial L}{\partial \beta_1} = -2\,(y_i-\hat{y}_i)\,x_{i1}}$$

```
coef_der = -2 * ((y_train[idx] - y_hat), X_train[idx])
Self.coef_ = Self.coef_ - (Self.lr * coef_der)
  X_train.shape
→ (353, 10)
```

Rule of matrix multiplication

A → (m×n) Shape

B → (n×p) Shape

C → (m×p) Shape

(353,10) * (1,1) = (10)

In this SGD Algorithm we converge to our minimum loss value by giving less number of epochs.

```
Sgd = SGDRegressor(learning_rate=0.01, epochs=50)
Sgd.fit(x_train, y_train)
→ 150.8948 [54.28, -67.43 351.20 251.58 17.26
-28.74 -165.85 123.27 314.84 123.52]
y_pred = sgd.predict(x_test)
r2_score(y_test, y_pred)
→ 0.4325
```

Time comparison

no. of epoch → fixed

e = 100

Batch → 100

Stochastic → 100 × n

When to use Stochastic GD

① Big data

② When we have non convex function

Non-Convex Convex

Because of stochastic nature of SGD it could be ~~...~~ get out from local minima (It have that velocity which is not stop it to local minima.)

Problem with SGD

① In Stochastic GD, even after reaching the optimal solution, the algorithm continues to update parameters and does not stop exactly at the minimum.

To solve this problem we use one concept called learning schedule.

learning schedule:- Learning rate schedule is a strategy in machine learning where the learning rate is changed (usually reduced) during training instead of keeping it constant, so that the model learn fast at the beginning and converges smoothly later

Code:-

```
t0,t1 = 5,50
def learning_rate(t):
    return t0(t**t1)
for i in range(epochs):
    for j in range(x.shape[0]):
        lr = learning_rate(i*x.shape[0]+j)
```

Using Sklearn SGD Regressor

```
From sklearn.linear_model import SGDRegressor
reg = SGDRegressor(max_iter=100, learning_rate="constant",
                                         eta=0.01)

reg.fit(x_train, y_train)
y_pred = reg.predict(x_test)
r2_score(y_test, y_pred)
→ 0.4305
```

* Mini-Batch Gradient Descent *

GD

Batch
1 update/epoch
{Slow}
{Small, convex Data}

Stochastic (n rows)
n updates/epoch
{fast, big data, nonconvex}

Minibatch GD: Combination of Batch & Stochastic GD

Batch → Group of rows

n=1000
↳ Batches → 100 → TB → 10

Batches updates/epochs

1000
10
Batches → 10 updates

⊛ Coding Part ⊛

```
From Sklearn.datasets import load_diabetes
import numpy as np
From sklearn.linear_model import LinearRegression
From sklearn.metrics import r2_score
From sklearn.model_selection import train_test_split
Import random
x,y = load_diabetes(return_x_y=True)
Print(x.shape)
Print(y.shape)
```

→ 442, 10
(442,)

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,
                                               random_state=2)

reg=LinearRegression()
reg.fit(X_train,y_train)
print(reg.coef_)
print(reg.intercept_)
--> [-9.160 -205.46 516.68 340.62 -895.54 561.21
     153.88 126.73 861.12 52.41 151.88]
   151.88

y_pred = reg.predict(X_test)
r2score = (y_test,y_pred)
--> 0.4399

class MBGDRegressor:
    def __init__(self,batch_size,learning_rate=0.01,
                                epochs=100):
        self.coef_=None
        self.intercept_=None
        self.lr = learning_rate
        self.epochs = epochs
        self.batch_size=batch_size

    def fit(self,X_train,y_train):
        self.intercept_ =0
        self.coef_ = np.ones(X_train.shape[1])

        for i in range(self.epochs):
            for j in range(int(X_train.shape[0]/
                               self.batch_size)):
                idx=random.sample(range(X_train.shape
                                  [0]),
```

```
        y_hat = np.dot(x_train[idx], self.coef_) +
                                    self.intercept_

        intercept_der = -2 * np.mean(y_train[idx] -
                                    y_hat)

        coef_der = -2 * np.dot((y_train[idx] - y_hat),
                                    x_train[idx])

        self.coef_ = self.coef_ - (self.lr * coef_der)

    print(self.intercept_, self.coef_)

def predict(self, x_test)
    return np.dot(x_test, self.coef_) + self.intercept_
```

```
mbr = MBGDRegressor(batch_size = int(x_train.shape[0]/10)
                    , learning_rate = 0.01,
                    epochs = 500)
```

```
mbr.fit(x_train, y_train)
```

→ 154.8374 [38.40 -142.67 457.28 303.60 -17.99 -85.81 -192.04 116.18 407.24 105.80]

```
ypred = mbr.predict(x_test)
r2_score(y_test, y_pred)
```

→ 0.45188

✱ How to use MBGDR in sklearn ✱

```
from sklearn.linear_model import SGDRegressor
sgd = SGDRegressor(learning_rate = 'constant', eta0 = 0.2)
# partial_fit(x, y, sample_weight=None)
  → using this we can pass subset of x_train,
                                     y_train)
```

```
batch-size = 35
for i in range(100)
    idx = random.sample(range(x-train.shape[0]), batch size)
    sgd.partial-fit(x-train[idx], y-train[idx])

    sgd.coef_
• → [49.19, -67.84, 338.57, 247.97, 25.30, -24.71,
          -155.45, 116.19, 312.91, 133.36]

   sgd.intercept_
   → 148.61

  y-pred = sgd.predict(x-test)
  r2_score(y-test, y-pred)
 → 0.4271
```