BATCH GRADIENT DESCENT (FOR multiple Linear Regression)

→ Gradient Descent is a first order derivative optimization algorithm of a differenciable function. Here we try to find best optimum values of parameters for the best fit hyperplane.

* LET US SEE THE MATHEMATICAL FORMULATION

→ Let us take a dataset

TV	Radio	Sales
4	11	29
7	14	40

Intentionally, I have taken small dataset so that we can understand GD.

* LET US TAKE REVIEW OF HOW GRADIENT DESCENT WORKS

Step 1: Start with Random values of Parameters (In our case $\beta_0, \beta_1, \beta_2$)

Step 2: Inside loop we try to update values of coefficients, we also initialize epochs, learning-rate:

$$\beta_0 = \beta_0 - (lr * slope)$$
$$\beta_1 = \beta_1 - (lr * slope)$$
$$\beta_2 = \beta_2 - (lr * slope)$$

→ all these slopes will be different as they are calculated particularly with respect to parameters

$$\beta_0 = \beta_0 - (lr * slope) \rightarrow \frac{\partial L}{\partial \beta_0}$$
$$\beta_1 = \beta_1 - (lr * slope) \rightarrow \frac{\partial L}{\partial \beta_1}$$
$$\beta_2 = \beta_2 - (lr * slope) \rightarrow \frac{\partial L}{\partial \beta_2}$$

* Let's Understand Loss Function

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

→ since we have only two columns here let's expand loss function

$$L = \frac{1}{2}\left[(y_1 - \hat{y}_1)^2 + (y_2 - \hat{y}_2)^2\right] \quad \text{where } \hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

$$= \frac{1}{2}\left[(y_1 - \underbrace{\beta_0 - \beta_1 X_{11} - \beta_2 X_{12}}_{\hat{y}_1})^2 + (y_2 - \underbrace{\beta_0 - \beta_1 X_{21} - \beta_2 X_{22}}_{\hat{y}_2})^2\right]$$

– Let us first find out $\frac{\partial L}{\partial \beta_0}$

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

here we are taking 2 because we have only 2 independent features in our dataset

→ for n features

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

* Let us find $\frac{\partial L}{\partial \beta_1}$ $\rightarrow \frac{1}{2}\left[(y_1 - \underbrace{\beta_0 - \beta_1 X_{11} - \beta_2 X_{12}}_{\hat{y}_1}) + (y_2 - \underbrace{\beta_0 - \beta_1 X_{21} - \beta_2 X_{22}}_{\hat{y}_2})^2 \right]$

$$\frac{\partial L}{\partial \beta_1} = \frac{1}{2}\left[2(y_1 - \hat{y}_1)(-X_{21}) + 2(y_2 - \hat{y}_2)(-X_{22}) \right]$$

→ for n features

$$-\frac{2}{n}\left[(y_1 - \hat{y}_1)(-X_{11}) + (y_2 - \hat{y}_2)(-X_{21}) + (y_3 - \hat{y}_3)(-X_{31}) - - - - - - - + (y_4 - \hat{y}_4)(-X_{m1}) \right]$$

→ Generalizing this equation

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

→ Similarly $\frac{\partial L}{\partial \beta_2}$ will be

$$= -\frac{2}{n} \sum_{i=1}^{n} \left[(y_i - \hat{y}_i) X_{i2} \right]$$

→ Generalizing the loss of any coefficient w.r.t to any feature

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

* Using this formula we can find $\hat{y}$ in one go

$$\Rightarrow \boxed{\beta_0 + np.dot(x\_train, coefficients)}$$

→ by this we will be able to calculate slopes of all the coefficients in one go except β_0.

→ we have found slope of β_0 but now we find slope of all the coefficients in one go.

$$\Rightarrow -\frac{2}{n} \times \left[np.dot((\underset{(n,1)}{y_i - \hat{y}_i}), \underset{(n,m)}{x\_train}) \right]$$

→ we will transpose $(y_i - \hat{y}_i)$ so shape will become $np.dot(\underset{(1 \times n)}{y_i - \hat{y}}, \underset{(n \times m)}{x\_train}) \Rightarrow \boxed{1 \times m}$

→ and here we get slopes of all our coefficients w.r.t slope.

```
class BatchGD:

  def __init__(self,epochs,learning_rate):

    self.epochs = epochs
    self.learning_rate = learning_rate
    self.intercept_ = None
    self.coef_ = None


  def fit(self,x_train,y_train):

    import numpy as np
    self.intercept_ = 0
    self.coef_ = np.ones(x_train.shape[1])

    for i in range(self.epochs):

      y_hat = self.intercept_ + np.dot(x_train,self.coef_)

      intercept_derivative = -2 * np.mean(y_train - y_hat)
      self.intercept_ = self.intercept_ - (self.learning_rate *
intercept_derivative)

      coef_derivatives = -2 * np.dot((y_train -
y_hat),x_train)/x_train.shape[0]
      self.coef_ = self.coef_ - (self.learning_rate *
coef_derivatives)

    print(self.intercept_)
    print(self.coef_)


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

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

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)
lr = LinearRegression()

lr.fit(x_train,y_train)
```

```
LinearRegression()

lr.coef_

array([  -9.15865318, -205.45432163,  516.69374454,  340.61999905,
       -895.5520019 ,  561.22067904,  153.89310954,  126.73139688,
        861.12700152,   52.42112238])

lr.intercept_

151.88331005254167

mlr = BatchGD(1000,0.5)
mlr.fit(x_train,y_train)

152.01351687661833
[  14.38990585 -173.7235727   491.54898524  323.91524824  -39.32648042
 -116.01061213 -194.04077415  103.38135565  451.63448787
97.57218278]
```