① { Problems with batch GD }

Let us suppose we have 1000 rows in our dataset and 3 predictors Then we need to calculate $\beta_0, \ldots, \beta_3$ using GD. Let us say we do 100 epochs. Then for each epoch we will update 4 values $\{\beta_0, \ldots, \beta_3\}$ using 1000 rows. Thus, for each epoch, we will be making 4000 calculations. Since, we have 100 epochs, the total number of calculations is $4000 \times 100 = 4 \times 10^5 = 4$ lakh.

So, even for such a simple dataset, we need to do 4 lakhs calculations which is not feasible if the dataset were huge.

So, let $n \triangleq$ # of rows
$m \triangleq$ # of predictor columns
$e \triangleq$ # of epochs

Then total number of calculations $= m * e * n = men$

The other problem with batch GD is that we need to load the entire dataset for calculating the $\hat{y}$'s. This is needed for calculating the derivatives each time.

$$\hat{y} = m * X + b$$

y_pred = X_train @ self.coef_ + self.intercept_

⇓

{ We need to load X_train into the RAM completely. This is not feasible if X_train is too huge }

② {Stochastic GD}

Stochastic GD updates the parameters $\beta_0, \ldots, \beta_m$ using only a single data point.

This data point is selected at random from our dataset and thus due to this, a noise gets introduced in the updates which can help escape local minima and find better solutions.

For each epoch, we select a data point at random from our dataset and calculate the $\hat{y}$ using this data point. We then calculate all the derivatives and update all our coefficients.

In each epoch, we select data points randomly. This is done for n times, where n = # of rows in the dataset.

This means that in each epoch, we update our coefficients $(\beta_0, \ldots, \beta_m)$ n times. This is in contrast to the vanilla GD, where in each epoch, we do only 1 updation.

$$\text{Total \# of updations in stochastic GD} = en$$

where $e \triangleq$ # of epochs & n = # of rows in X

③ {Mathematical formulation of SGD}

let $X \longrightarrow \begin{bmatrix} [x_{11}, x_{12}, \ldots, x_{1m}], \\ [x_{21}, x_{22}, \ldots, x_{2m}], \\ \vdots \\ [x_{n1}, x_{n2}, \ldots, x_{nm}] \end{bmatrix}$

$y \rightarrow [y_1, y_2, \ldots, y_n]$

let the equation of the hyperplane be

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_m x_m$$

We know from batch GD, that

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

and

$$\frac{\partial L}{\partial \beta_j} = \frac{-2}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i) x_{ij}, \quad j \neq 0$$

Since in stochastic GD, we make use of only a single point while updation, We do not need Σ while calculating $\frac{\partial L}{\partial \beta_j}$'s in SGD. So, we select a random row $i_0 \in \{0, \ldots, n-1\}$ and calculate $\frac{\partial L}{\partial \beta_k}$, $k \in \{0, \ldots, m\}$.

$$\frac{\partial L}{\partial \beta_0} = 2(\hat{y}_{i_0} - y_{i_0})$$

$$\frac{\partial L}{\partial \beta_j} = 2(\hat{y}_{i_0} - y_{i_0}) x_{i_0 j}, \quad j \neq 0$$

Here $\hat{y}_{i_0} = \beta_0 + \beta_1 x_{i_0 1} + \beta_2 x_{i_0 2} + \cdots + \beta_m x_{i_0 m}$

$$= \beta_0 + \sum_{k=1}^{m} \beta_k x_{i_0 k}.$$

$y_{i_0} = i_0^{th}$ row of y

$X_{i_0} = i_0^{th}$ row of $X = \begin{bmatrix} x_{i_0 1} & x_{i_0 2} & \cdots & x_{i_0 m} \end{bmatrix}$

and thus, $X_{i_0 j} = j^{th}$ element of X_{i_0}

Finally, we can write

$$(\beta_0, \ldots, \beta_m)_{new} = (\beta_0, \ldots, \beta_m)_{old} - \eta \left(\frac{\partial L}{\partial \beta_0}, \ldots, \frac{\partial L}{\partial \beta_m} \right)_{\substack{\text{evaluated at} \\ (\beta_0, \ldots, \beta_m)_{old}}}$$

Another thing to notice about the given derivatives is that $\frac{\partial L}{\partial \beta_k}$ is a real number for all $k \in \{0, \ldots, m\}$

$$\frac{\partial L}{\partial \beta_0} = 2(\hat{y}_{i_0} - y_{i_0}) \in \mathbb{R}$$

$$\frac{\partial L}{\partial \beta_j} = 2(\hat{y}_{i_0} - y_{i_0}) x_{i_0 j}, \quad j \neq 0 \in \mathbb{R}.$$

In python code, the above equations would be written as

$$\frac{\partial L}{\partial \beta_0} = 2 * (y\_pred - y[idx]) \in \mathbb{R}$$

$$\mathbb{R}^m \ni \left(\frac{\partial L}{\partial \beta_j} \right)_{j=1}^{m} = 2 * (y\_pred - y[idx]) * X[idx]$$

fancy indexing

④ { Code for stochastic GD }

The `__init__()` will have 3 parameters namely self, learning rate and epochs.

```
def __init__(self, learning_rate, epochs):

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

The `predict()` method is also fairly easy.

```
def predict(self, X_test):

    return X_test @ self.coef_
           + self.intercept_
```

The `fit()` method takes in two parameters, namely X_train and y_train.

```
def fit(self, X_train, y_train):

    n = X_train.shape[0]
    m = y_train.shape[1]

    self.intercept_ = 0
    self.coef_ = np.ones(m)

    for _ in range(self.epochs):
        for __ in range(n):
```

```
# select a random data point
idx = random.sample(range(n),1)[0]
y_pred = self.coef_ @ X_train[idx]
                + 
         self.intercept_

# calculate the derivatives
intercept_der = 2*(y_pred - y_train[idx])
coef_der = 2*(y_pred - y_train[idx])
           * X_train[idx]

# update the coef's
self.coef_ = self.coef_ -(self.lr)* coef_der
self.intercept_ = self.intercept_ - (self.lr)*
                                    intercept_der
```

⑤ {Speed comparison between batch and stochastic GD}

It is generally said that Stochastic GD is much faster than the batch GD. However, if we keep the # of epochs same, then

$$T(\text{Batch GD}) \lll T(\text{Stochastic GD})$$

Then why do we say that SGD is faster than Batch GD?

↓

The reason is because SGD usually takes much less # of epochs to reach to the minima and thus converges to the optimal solution faster than BGD.

⑥ {When to use SGD?}

* we can use SGD in case of Big data. The Batch GD is NOT suitable for huge dataset.

* we can use SGD in case when cost function is NOT a convex function