Date: 09-02-24

Gradient Descent

derivative

Gradient Descent:

- Gradient Descent is a 1st order iterative optimization algorithm for finding a local minimum of a differentiable function.
- used in linear regression, logistic regression, tsne

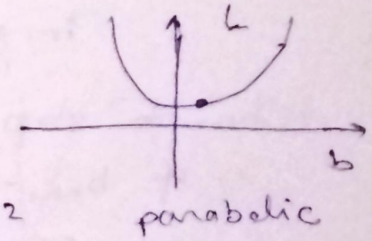
Types of Gradient Descent

- Batch Gradient Descent
- Stochastic Gradient Descent
- Mini Gradient Descent

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

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

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



parabolic

* Here the relation b/w L and b is $L \to b^2$

- इसमें m का value already known है above eqn से तब given graph में b का एक ऐसा value निकालना है जहाँ पर L is minimum

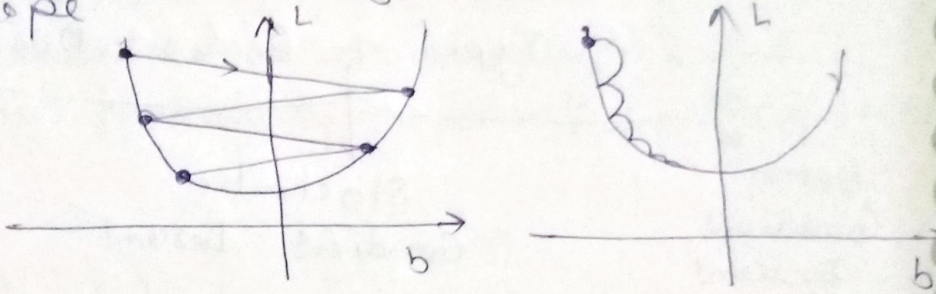
Approach of Gradient Descent:

step-1. Select the random value of b

step-2. find the slop. if slope = -ve move right (b increment) else move left (b decrement)

- In the Gradient Descent the idea is to take repeated steps in the opposite direction of the gradient (for approximate gradient) of the function at the current point, because this is the direction of steepest descent.

- for avaiding drastic change in the step size we have to multiply η (learning rate) with slope



in general $\eta = 0.01$

when to stop:

if $b_{new} - b_{old} \leq 0.0001$

or 1000 times iterations (epochs)

Mathematical Formulation:

let Here m is given

step- start with a random $b = b$

Here $\eta = 0.01$

for i in epochs:

$b_{new} = b_{old} - \eta \times slop$

let us assume $b = 0$

find slop at $b = 0$

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

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

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

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

above is equation of slope

for $b = 0$

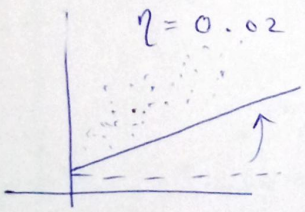
$$= -2\sum_{i=1}^{n}(y_i - mx_i)$$

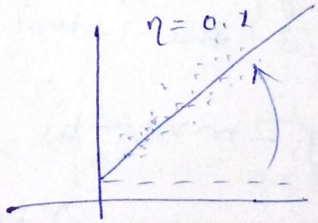
$$= -2\sum_{i=1}^{n}(y_i - 78.35x_i)$$

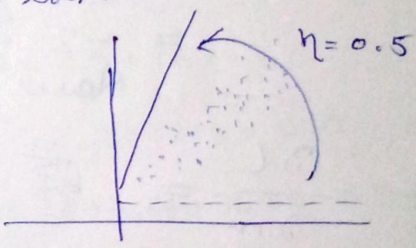
$$b_{new} = b_{old} - \eta \, Slope$$

Effect of Learning Rate:

Here epoch = 10 for all three case:







- gradient descent is work on all differentiable loss functions.

Calculating m and b both:

Steps:

1- initial random value for m and b

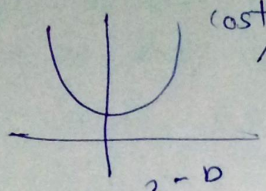
let m = 1 and b = 0

2. epochs = 100 and lr = 0.01

for i in epochs:

b = b - η slope

m = m - η slope

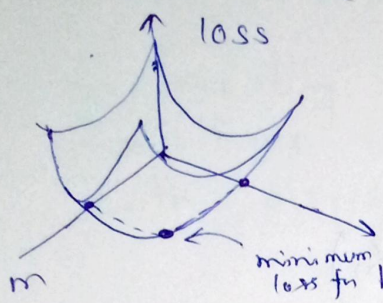


cost/loss function $L = \sum (y_i - \hat{y}_i)^2$

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

at the time of calculating b Here m is constant but

Now $L(m, b)$ = cost/loss function



यहाँ पर m and b का एक ऐसा value जहाँ पर loss function is minimum

b के respect में slope = $\frac{\partial L}{\partial b}$

m के respect में slope = $\frac{\partial L}{\partial m}$

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

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

Now put the initial value of b ($b = 0$)

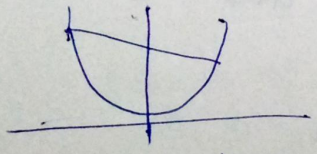
$$\frac{\partial L}{\partial m} = \sum (y_i - mx_i - b)^2$$

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

Now put initial value of m ($m = 1$)

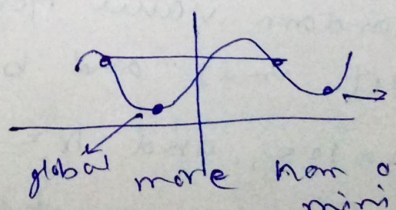
$L = \sum (y_i - \hat{y}_i)^2$ = mean square error

→ Convex function



one minima (global minima)

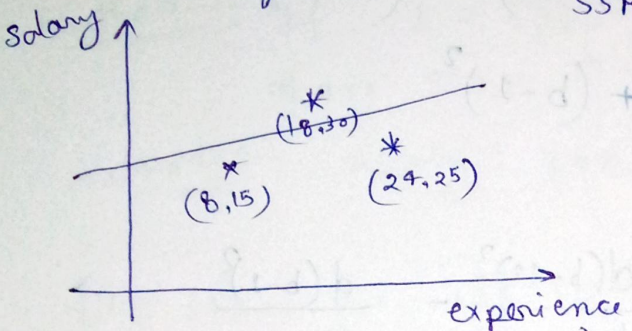
Convex loss function



Non convex loss function

Gradient Descent:

- This method is used when we have large number of dataset and large number of columns in dataset.
- This technique used in linear Regression, logistic regression, PCA and ~~Hestr~~ Neural Network
- it is optimization technique, parameter optimization technique.

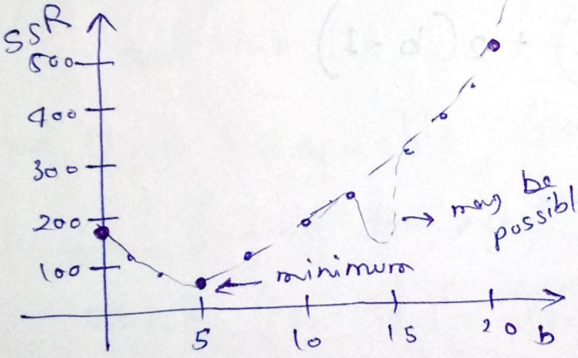


$$SSR = (P_{y_1} - a_{y_1})^2 + (P_{y_2} - a_{y_2})^2 + (P_{y_3} + a_{y_3})^2$$

where p = predicted
a = actual

$y = mx + b$
for $m = 1$ $b = 0$

$SSR = 49 + 44 + 1 = 194$



for $m = 1$ $b = 5$
$SSR = 4 + 49 + 16 = 69$

for $m = 1$ $b = 20$
$SSR = 169 + 64 + 361 = 594$

if SSR is function of b
$SSR = f(b) = b^2 - 4$ | $\frac{df}{db} = 2b \Rightarrow b = 0$

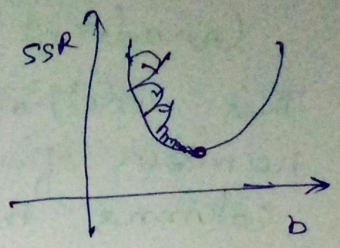
* SSR is also called loss function

step-1 find the equation of f(b)
Step-2 differentiate
Step-3. put equat to 0

then we get where curve goes minimum and if there is no any minima in curve then gradient descent calculate it

assume $m = 1$

$$SSR = (Py_1 - ay_1)^2 + (Py_2 - ay_2)^2 + (Py_3 - ay_3)^2$$



let us equation $y = mx + b$

$$y = x + b$$

$$SSR = (Py_1 - 15)^2 + (Py_2 - 30)^2 + (Py_3 - 25)^2$$

$$SSR = ((8+b) - 25)^2 + ((18+b) - 30)^2 + ((24+b) - 25)^2$$

$$SSR = (b-17)^2 + (b-12)^2 + (b-1)^2$$

$$SSR = f(b)$$

$$SSR' = \frac{d f(b)}{db} = \frac{d}{db}(b-17)^2 + \frac{d(b-12)^2}{db} + \frac{d(b-1)^2}{db}$$

$$= 2(b-17) + 2(b-12) + 2(b-1)$$

$$SSR' = \frac{d f(b)}{db} = 6b - 60$$

for $b = 0$

$$SSR' = -60$$

learning Rate

step size = SSR' x learning rate

$$= -60 \times 0.1$$

step size = 6

new intercept = old intercept - step size

$$= 0 - (-6)$$

$$= 6$$

for $b = 6$, $SSR' = -24$

step size $= -24 \times 0.1 = -2.4$

new intercept $= 6 - (-2.4) = 8.4$ (approx 9)

for $b = 9$, $SSR' = -6$

step size $= -0.6$ new intercept $= 9.6$

* How many times we do above steps:

method-1 if $1 >$ step size $= 0.001$ (absolute) then stop the above step.

method-2 apply 1000 times above steps

till Now in all in the above we take a assumption, the assumption is that the

$m = 1$

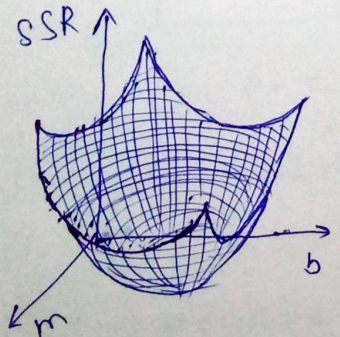
Multiple Variable Gradient Descent:

for finding ~~tin~~ equation of line $y = mx + b$

$$SSR = (Py_1 - ay_1)^2 + (Py_2 - ay_2)^2 + (Py_3 - ay_3)^2$$

$$SSR = ((8m+b) - 15)^2 + ((24m+b) - 25)^2 + ((18m+b) - 30)^2$$

$$SSR = f(m, b)$$



step-1. find equation

step-2. Calculate derivative.

step-3. putting value of m, b

step-4. find minima

$$\frac{df(m,b)}{db} = 2((8m+b-15) \times \frac{d}{db}(8m+b-15) + 2((29m+b-25)) + 2((18m+b)-30)$$

$$\frac{df(m,b)}{db} = 2((8m+b)-15) + 2((29m+b)-25) + 2((18m+b)-30)$$

Now,

$$\frac{df(m,b)}{dm} = 16((18m+b)-15) + 58((29m+b)-25) + 36((18m+b)-30)$$

let $m = 1$ and $b = 0$

$(SSR)'_b =$

step size $= (SSR)'_b \times 0.1$

NI = OI - step size

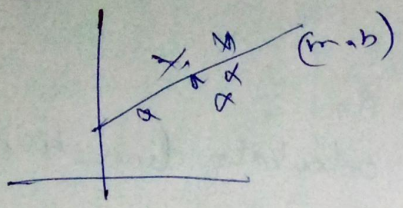
$(SSR)'_m =$

step size $= (SSR)'_m \times 0.1$

NI = OI - step size

* There are three varient of gradient descent, which differ in how much data we use to compute the gradient of the objective function. Depending on the amount of data, we make a trade-off between the accuracy of the parameter update and the time it takes to perform an update.

(1.) Batch GD (ndim)



$m=1, b=0$

$m_1 = m_0 - \eta \alpha (\text{slope})_{m=0}$ → $\frac{\partial L}{\partial m}$

$b_n = b_0 - \eta \times (\text{slope})_{b=0}$ → $\frac{\partial L}{\partial b}$

* Stochastic GD is suitable for large dataset
* mini batch GD is intermediate b/w the Batch GD and stochastic GD
* Batch GD used when we have the convex function. work only on small dataset
* also called vanilla gradient descent.

Here we work on more than two dimension data (ndim) example.

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

$\Rightarrow \{\beta_0, \beta_1, \beta_2, \beta_3\}$ ← coefficients.

→ let us assume we have three column data set that is

cgpa x_1	iq x_2	LPA y
8.1	92	3.2
7.5	95	3.5

x_{11} x_{12} y_1

x_{21} x_{22} y_2

Here $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2$

Steps:

(1.) start with Random values

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

(2.) epoch = 100, learning rate (η) = 0.1

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

$\beta_1 = \beta_1 - \eta \text{ slope}$

$\beta_2 = \beta_2 - \eta \text{ slope}$

4-b graph.

(3.) $L(\beta_0, \beta_1, \beta_2)$ = loss function

calculate $\frac{\partial L}{\partial \beta_0}$, $\frac{\partial L}{\partial \beta_1}$, $\frac{\partial L}{\partial \beta_2}$

for n-dim we calculate derivative from $(n+1)\ \beta_0$ to β_n

we know Loss function $= \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 =$ MSE

{row = 2 , cols = 2+1}

$$\hat{Y} = \beta_0 + \beta_1 X_1 + \beta_2 X_2$$

$$L = \frac{1}{2}\left[(Y_1 - \hat{Y}_1)^2 + (Y_2 - \hat{Y}_2)^2\right]$$

$$\hat{Y}_1 = \beta_0 + \beta_1 X_{11} + \beta_2 X_{12}$$

$$\hat{Y}_2 = \beta_0 + \beta_1 X_{21} + \beta_2 X_{22}$$

$$L = \frac{1}{2}\left[(Y_1 - \beta_0 - \beta_1 X_{11} - \beta_2 X_{12})^2 + (Y_2 - \beta_0 + \beta_1 X_{21} - \beta_2 X_{22})^2\right]$$

Now, calculate slope-1 , slope-2 , slope-3.

(I) $$\frac{\partial L}{\partial \beta_0} = \frac{1}{2}\left[2(Y_1 - \hat{Y}_1).(-1) + 2(Y_2 - \hat{Y}_2).(-1)\right]$$

$$= -\frac{2}{2}\left[(Y_1 - \hat{Y}_1) + (Y_2 - \hat{Y}_2)\right]$$

for n raw

$$= \frac{-2}{n}\left[(Y_1 - \hat{Y}_1) + (Y_2 - \hat{Y}_2) + \cdots + (Y_n - \hat{Y}_n)\right]$$

$$\frac{\partial L}{\partial \beta_0} = \frac{-2}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i) = \text{slope-1}$$

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

$$= \frac{1}{2} \left[(Y_1 - \hat{Y}_1)^2 + (Y_2 - \hat{Y}_2)^2 \right]$$

(I) $$\frac{\partial L}{\partial \beta_1} = \frac{1}{2} \left[2(Y_1 - \hat{Y}_1)(-X_{11}) + 2(Y_2 - \hat{Y}_2)(-X_{21}) \right]$$

for n-dimension data

$$\frac{\partial L}{\partial \beta_1} = \frac{-2}{n} \left[(Y_1 - \hat{Y}_1)(X_{11}) + (Y_2 - \hat{Y}_2)(X_{21}) + \dots (Y_n - \hat{Y}_n)(X_{n1}) \right]$$

$$\frac{\partial L}{\partial \beta_1} = \frac{-2}{n} \sum_{i=1}^{n} (Y_i - \hat{Y}_i)(X_{i1}) = \text{slope-2}$$

→ 1st column data

(III) $$\frac{\partial L}{\partial \beta_2} = \frac{-2}{n} \sum_{i=1}^{n} (Y_i - \hat{Y}_i).(X_{i2}) = \text{slope-3}$$

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

A problem with Batch GD:

1. let, n (number of rows) = 10^5

columns = 10^2

epoch = 10^3

total derivative calculate = 10^{10}

← Hence high computation. the program will be slow

2. Hardware problem.

Code:

```
for i in range(nb_epochs):
    params_grad = evaluate_gradient(loss_function, data, params)
    params = params - learning_rate * params_grad
```

* Here no. of epoch is equal to number of updates

(2.) Stochastic Gradient Descent

- it is better than the Batch GD because. it is require less number of epochs.
  Here the updates is taken by row (randomly)
- it gives us faster convergence (less no. of epochs require)
- easy to handle.

stochastic: having a random probability distribution or pattern that may be analysed statistically but may not be predicted precisely.

- it is not give steady solution.
- for the same number of epochs.

e = 100

Batch GD
100 = updates
is better than stochastic

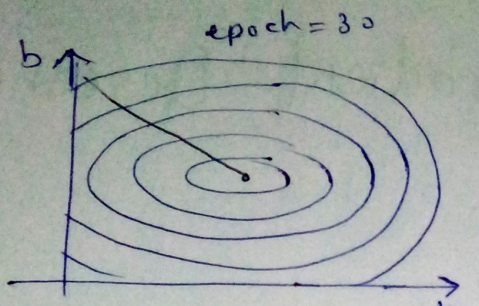
stochastic GD
100 x n = updates

- but stochastic not require large number of epochs (eg. 100 here) it provide similar performance in less number of epochs.

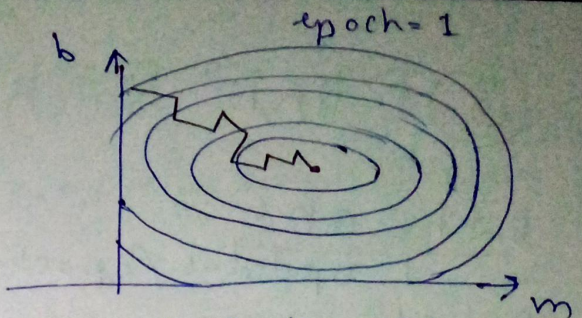
Batch GD
e = 100

stochastic GD
e = 90
fast computation

same performance



Batch GD



Stochastic GD

when to use stochastic GD:
- when data is very big
- when we have non-convex functions

problem with stochastic GD:
- in the graph m and b value are very fluctuating
- To resolve this we use learning schedules

```
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)
```

Here i = epoch.

e.g. n = 100
epoch = 1 $\quad$ $\lambda = 0.1$

(3.) Mini Batch Gradient Descent

Batch GD:
- 1 updates / epoch
- slow
- useful for small data, convex function.

Stochastic GD:
- n (raws) updates / epoch
- it is fast execution
- use ~~fun~~ full for big data, non convex function.
- not give us a optimal solution. (randomness)

Mini Batch GD:
- Batch means group of raws (n) = 10000
  in one batch = 1000
  hence number of batch = 10
- Batches update / epochs.
- it is intermediate b/w the Batch GD and Stochastic GD.
- it is the methods to reduce the randomness in stochastic GD.
- highly used in DL.