Gradient Descent

Definition1: A Gradient Descent is a very famous optimization technique that is used in machine learning and in deep learning. The main purpose of gradient descent is to minimize the cost function. The cost function is nothing but a method to find out the error between the actual output and the predicted output. **In Gradient Descent we try to find out the slope at a given point and try to minimize the slope value and take the slope vlaue near to zero.**

Defination2 from Wikipedia: In mathematics, **gradient descent** (also often called **steepest descent**) is a first-order iterative optimization algorithm for finding a local minimum of a differentiable function. The idea is to take repeated steps in the opposite direction of the gradient (or approximate gradient) of the function at the current point, because this is the direction of steepest descent. Conversely, stepping in the direction of the gradient will lead to a local maximum of that function; the procedure is then known as gradient ascent.

Gradient descent is generally attributed to Augustin-Louis Cauchy, who first suggested it in 1847.[1] Jacques Hadamard independently proposed a similar method in 1907.[2][3] Its convergence properties for non-linear optimization problems were first studied by Haskell Curry in 1944,[4] with the method becoming increasingly well-studied and used in the following decades.[5][6]

An analogy for understanding gradient descent: [Wikipedia]

Fog in the mountains

The basic intuition behind gradient descent can be illustrated by a hypothetical scenario. A person is stuck in the mountains and is trying to get down (i.e., trying to find the global minimum). There is heavy fog such that visibility is extremely low. Therefore, the path down the mountain is not visible, so they must use local information to find the minimum. They can use the method of gradient descent, which involves looking at the steepness of the hill at their current position, then proceeding in the direction with the steepest descent (i.e., downhill). If they were trying to find the top of the mountain (i.e., the maximum), then they would proceed in the direction of steepest ascent (i.e., uphill). Using this method, they

would eventually find their way down the mountain or possibly get stuck in some hole (i.e., local minimum or saddle point), like a mountain lake. However, assume also that the steepness of the hill is not immediately obvious with simple observation, but rather it requires a sophisticated instrument to measure, which the person happens to have at the moment. It takes quite some time to measure the steepness of the hill with the instrument, thus they should minimize their use of the instrument if they wanted to get down the mountain before sunset. The difficulty then is choosing the frequency at which they should measure the steepness of the hill so not to go off track.

In this analogy, the person represents the algorithm, and the path taken down the mountain represents the sequence of parameter settings that the algorithm will explore. The steepness of the hill represents the slope of the function at that point. The instrument used to measure steepness is differentiation. The direction they choose to travel in aligns with the gradient of the function at that point. The amount of time they travel before taking another measurement is the step size.

Types of Gradient Descent:

There are three types of gradient descent:

1. Batch Gradient Descent.
2. Stochastic Gradient Descent.
3. Mini-Batch Gradient Descent.

Intuition: Here, we will try to understand how Gradient Descent work on **Linear Regression**. We know in linear regression we have to find out a best fit line or the relationship between independent and dependent variable.

We know, loss function of Linear Regression is: (I take mean squared error)

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

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

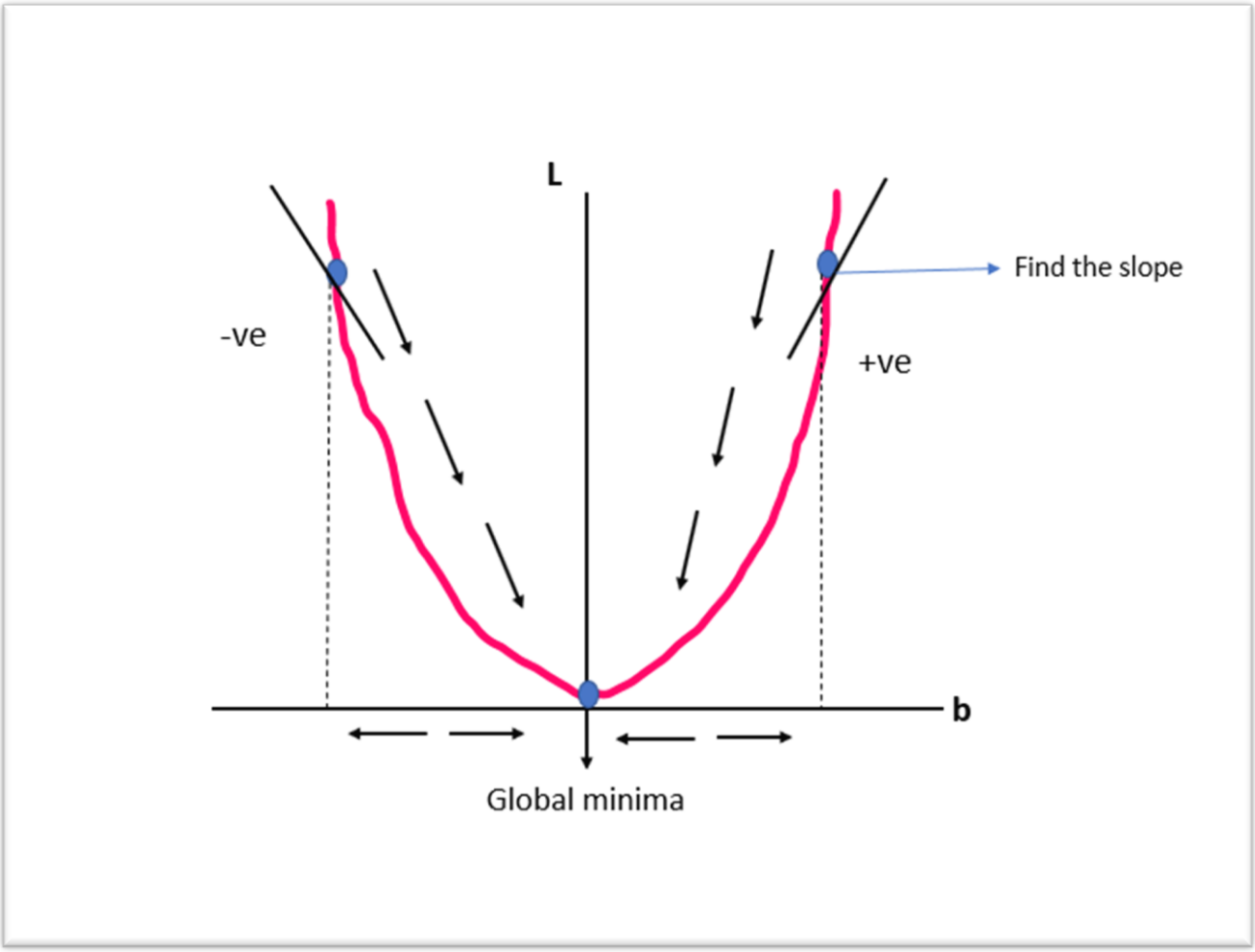
$L(m, b)$ denote we have to find out **'m'** and **'b'** value so that our loss function will be minimum.

Here, at first, I will assume **'m'** is constant or we know the **'m'** value. Let m=78.35

Now, loss function will be:

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

Now, Loss function will depend on b^2. If we draw **Loss vs b** in graph then it would be like this:



In this function we can see just one minima and our goal is to reach this point so that our loss will be minimum.

We will assume of random value of ***b***. You can start with b=10. Since Gradient Descent is an iterative method that's why we have to update the value of 'b' for very iteration. To do update the formula will be:

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

Now how we will calculate slope value. Let me give you a very good example to make you understand. Let **$y = x^2 + 2$ is a function** and we want to find out slope at the particular point **x=5** then we will differentiate **y** with respect to **x**.

$$y = x^2 + 2$$

$$\frac{dy}{dx} = 2x - 0 = 2x = 2 * 5 = 10$$

Note: If slope is negative then go forward or increase the **'b'** value and if slope is positive then go backward or decrement the **'b'** value.

Learning Rate: In order for Gradient Descent to work, we must set the learning rate to an appropriate value. This parameter determines how fast or slow we will move towards the optimal weights. If the learning rate is very large, we will skip the optimal solution. If it is too small, we will need too many iterations to converge to the best values. So, using a good learning rate is crucial.

Now formula will be:

$$b_{new} = b_{old} - \eta * slope$$

When to stop iteration:

1. If $b_{new} - b_{old} = 0.0001(very\ small\ change)$
2. Fixed number of iteration (Ex: Epoch=100).

Mathematical Formulation:

Let's find out the equation of slope for our $\boldsymbol{b_{new}}$. The equation is the slope value will come from our loss function. Let's see:

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

Let's differentiate $\boldsymbol{L}$ with respect to $\boldsymbol{b}$:

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

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

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

And this is our slope equation. Now we can just put the value and we will get the slope value at the point of **b=10**

Till now we just try to find out of slope of the 'b'

Now we will work 'm' and 'b' together:

Let's write the steps of Gradient Descent for Simple Linear Regression:

- At first, we assume to random values for 'm' and 'b'. Let, m=1, b=0
- Since Gradient Descent is iterative approach so that every iteration, we will update the 'm' and 'b' value so that our **L (m, b)** will reduce.

  Epochs=100, learning rate=0.01
  For I in epochs:

$$b_{new} = b_{old} - \eta * slope$$

$$m_{new} = m_{old} - \eta * slope$$

- The iteration will be stop when $b_{new} - b_{old} = 0.0001$ or $m_{new} - m_{old} = 0.0001$ that means the change between new value and old value will be very less.

- After completing the iteration, we will get the optimal vlaue of ‘m’ and ‘b’ for our loss function.

Previous we just find out slope for **‘b’**. Now we will see how to find out the slope in terms of two variable **‘m’** and **‘b’**. Slope value of **‘m’** says the direction of **‘m’** value and slope value of **‘b’** says the direction of **‘b’** value. In nutshell, we can say we have to find out **‘m’** and **‘b’** values so that our loss function will be minimum.

We know our loss function:

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

- **Partial Derivatives of ‘L’ with respect to ‘b’:**

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

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

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

- **Partial Derivatives of ‘L’ with respect to ‘m’:**

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

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

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

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

This is two equations to find out the slope for 'm' and 'b'.

Let's see how to work Gradient descent for Multiple Linear Regression:

We know, in multiple linear regression work more than one features. And we have to draw a best fit line among all the data point. We know the multiple linear regression straight line formula that is:

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

For this formula we have to find out $\beta_0, \beta_1, \beta_2 \ldots \beta_n$. This all are coefficient values.

For this our loss function will be:

$$\boldsymbol{L(\beta_0, \beta_1, \beta_2 \ldots \beta_n) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Let's work with an example so that it is easy to understand.

CGPA	IQ	Gender	LPA
x_1	x_2	x_3	*Y*(target)
8.1 [x_{11}]	89 [x_{12}]	Male [x_{13}]	3.1 [y_1]
7.5 [x_{21}]	78 [x_{22}]	Female [x_{23}]	3.5 [y_2]

This is our toy dataset. In this dataset there are 4 columns and 2 input rows.

Let's write the steps of Gradient Descent for Multiple Linear Regression:

- At first, we assume to random values for $\beta_0 = 0$ and $\beta_1, \beta_2, \beta_3 = 1$
- Since Gradient Descent is iterative approach so that every iteration, we will update the β_0 and $\beta_1, \beta_2, \beta_3$ values so that our $\boldsymbol{L(\beta_0, \beta_1, \beta_2, \beta_3)}$ will reduce.

Epochs=100, learning rate=0.01
For I in epochs:

$$\beta_0 = \beta_0 - \eta * slope$$
$$\beta_1 = \beta_1 - \eta * slope$$
$$\beta_2 = \beta_2 - \eta * slope$$
$$\beta_3 = \beta_3 - \eta * slope$$

- The iteration will be stop when the change between new value and old value will be very less.
- After completing the iteration, we will get the optimal vlaue of β_0 and $\beta_1, \beta_2, \beta_3$for our loss function.

Let's find out intercept ($\boldsymbol{\beta_0}$) and all coefficient ($\boldsymbol{\beta_0, \beta_1, \beta_2, \beta_3}$) values using partial derivatives as we early done.

Here, n=2 because our dataset number of input rows is two.

$$\hat{y_i} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3$$

From this formula we can write individually

$$\boldsymbol{\hat{y_1} = \beta_0 + \beta_1 x_{11} + \beta_2 x_{12} + \beta_3 x_{13}}$$
$$\boldsymbol{\hat{y_2} = \beta_0 + \beta_1 x_{21} + \beta_2 x_{22} + \beta_3 x_{23}}$$

We know:

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

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

$$= \frac{1}{2}[(y_1 - \hat{y_1})^2 + (y_2 - \hat{y_2})^2]$$

$$= \frac{1}{2}[(y_1 - (\beta_0 + \beta_1 x_{11} + \beta_2 x_{12} + \beta_3 x_{13}))^2 + (y_2 - (\beta_0 + \beta_1 x_{21} + \beta_2 x_{22} + \beta_3 x_{23}))^2]$$

- Partial derivatives of L with respect to β_0:

$$\frac{\partial L}{\partial \beta_0} = \frac{1}{2}\frac{\partial}{\partial \beta_0}[(y_1 - (\beta_0 + \beta_1 x_{11} + \beta_2 x_{12} + \beta_3 x_{13}))^2 + (y_2 - (\beta_0 + \beta_1 x_{21} + \beta_2 x_{22} + \beta_3 x_{23}))^2]$$

$$= \frac{1}{2}[2(y_1 - \hat{y_1}).(-1) + 2(y_2 - \hat{y_2}).(-1)]$$

$$= \frac{2}{2}[(y_1 - \hat{y_1}).(-1) + (y_2 - \hat{y_2}).(-1)]$$

$$= \frac{-2}{2}[(y_1 - \hat{y_1}) + (y_2 - \hat{y_2})]$$

If the number of rows is 'n' then:

$$= \frac{-2}{n}[(y_1 - \hat{y_1}) + (y_2 - \hat{y_2}) + \cdots + (y_n - \hat{y_n})]$$

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

Again:

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

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

$$= \frac{1}{2}[(y_1 - \hat{y_1})^2 + (y_2 - \hat{y_2})^2]$$

$$= \frac{1}{2}[(y_1 - (\beta_0 + \beta_1 x_{11} + \beta_2 x_{12} + \beta_3 x_{13}))^2 + (y_2 - (\beta_0 + \beta_1 x_{21} + \beta_2 x_{22} + \beta_3 x_{23}))^2]$$

- Partial derivatives of L with respect to β_1:

$$\frac{\partial L}{\partial \beta_1} = \frac{1}{2}\frac{\partial}{\partial \beta_1}[(y_1 - (\beta_0 + \beta_1 x_{11} + \beta_2 x_{12} + \beta_3 x_{13}))^2 + (y_2 - (\beta_0 + \beta_1 x_{21} + \beta_2 x_{22} + \beta_3 x_{23}))^2]$$

$$= \frac{1}{2}[2(y_1 - \hat{y_1}).(-x_{11}) + 2(y_2 - \hat{y_2}).(-x_{21})]$$

$$= \frac{2}{2}[(y_1 - \hat{y_1}).(-x_{11}) + (y_2 - \hat{y_2}).(-x_{21})]$$

$$= \frac{-2}{2}[(y_1 - \hat{y_1}).(x_{11}) + (y_2 - \hat{y_2}).(x_{21})]$$

If the number of rows is 'n' then:

$$= \frac{-2}{2}[(y_1 - \hat{y_1}).(x_{11}) + (y_2 - \hat{y_2}).(x_{21}) + (y_3 - \hat{y_3}).(x_{31}) + \cdots + (y_n - \hat{y_n}).(x_{n1})]$$

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

Let's understand what is represent x_{i1}:

x_{i1}	x_{11}
	x_{21}
	x_{31}
	x_{n1}

x_{i1} represent the first column of our dataset and β_1 represent the coefficient values of first column. So, if we have *'m'* number of columns or features then the equation will be:

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

Note: And this is the general formula to find out any coefficient value for any column.

Now we can find out easily slope value for β_2 **and** β_3**.** Those equation will be:

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

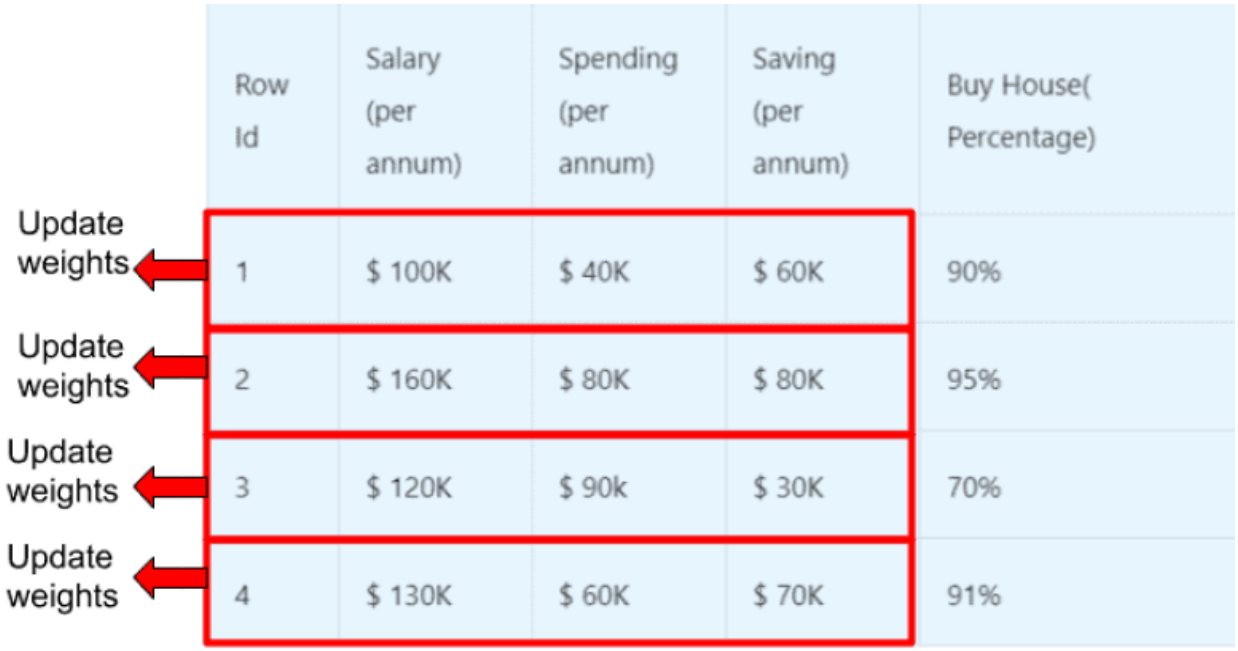
$$\frac{\partial L}{\partial \beta_3} = \frac{-2}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)x_{i3}$$

Different types of Gradient Descent:

Row Id	Salary (per annum)	Spending (per annum)	Saving (per annum)	Buy House(Percentage)
1	$ 100K	$ 40K	$ 60K	90%
2	$ 160K	$ 80K	$ 80K	95%
3	$ 120K	$ 90k	$ 30K	70%
4	$ 130K	$ 60K	$ 70K	91%

Update weight

Batch Gradient Descent: Suppose this is our dataset where our algorithm has to predict the percentage that a person will buy the house or not, based on his/her salary, spending, and saving. We have actual output "Buy House" as y. So, in normal gradient descent or batch gradient descent we take all of our rows and plug them into our algorithm. After plugging into algorithm, we calculate the cost function with the help of formula **cost = ½*square(y-y^).** Based on the cost function, we update the intercepts and coefficient. This process is of normal Gradient Descent. It is also known as **Batch Gradient Descent.** Batch Gradient update the weights after seeing the all rows. For example, in our dataset there are four rows. So, in batch Gradient Descent a single update will happen after seeing the four rows. Batch Gradient Descent is for small dataset and not efficient for large dataset. Because, when a single update is happened in that time full input data should be load in memory. So, if the dataset is too large then it is not possible to do.



Stochastic Gradient Descent: In SGD we take the row one by one. So, we take one row and calculate the cost and update the weights or intercept and coefficient. Then we move the second row. This process repeats for all other rows. So, in SGD basically, we are update the weights after very single row rather than doing everything together and then adjusting the weights. In our dataset there are four rows. So, SGD will update the weights or intercept and coefficient four times. SGD is most useful algorithm in Gradient descent. Here, we just take one row at a time so this technique not occupy full ram and it is faster than Batch Gradient Descent.

Mini-Batch Gradient Descent: There is one more method that is mini-batch gradient descent. In mini-batch gradient descent method, you can run batches of rows. The batches may be of 5 rows, or anything else. You can decide the number of batches. So, after running one batch, the weights or intercept and coefficient will be updated. If in our dataset have 300 rows and if we decide the batch size is 30 then update will be happened 10 times.