Gradient Descent

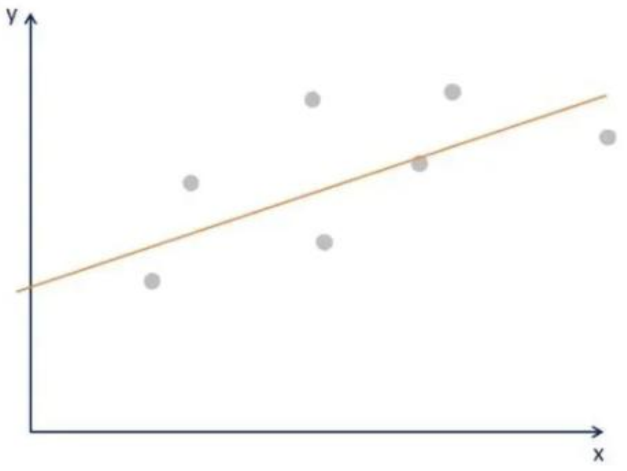
Chaithanya K.S

Roll No. 22

Linear Regression

Linear regression model. Geometrical representation

$$\hat{y}_i = b_0 + b_1 x_i$$



Linear regression is a statistical modeling technique used to model the relationship between a **dependent variable** (also called the target or response variable) and one or more **independent variables** (also called predictors or features).

It assumes that the relationship between the variables is linear, meaning that the change in the dependent variable is directly proportional to the change in the independent variables.

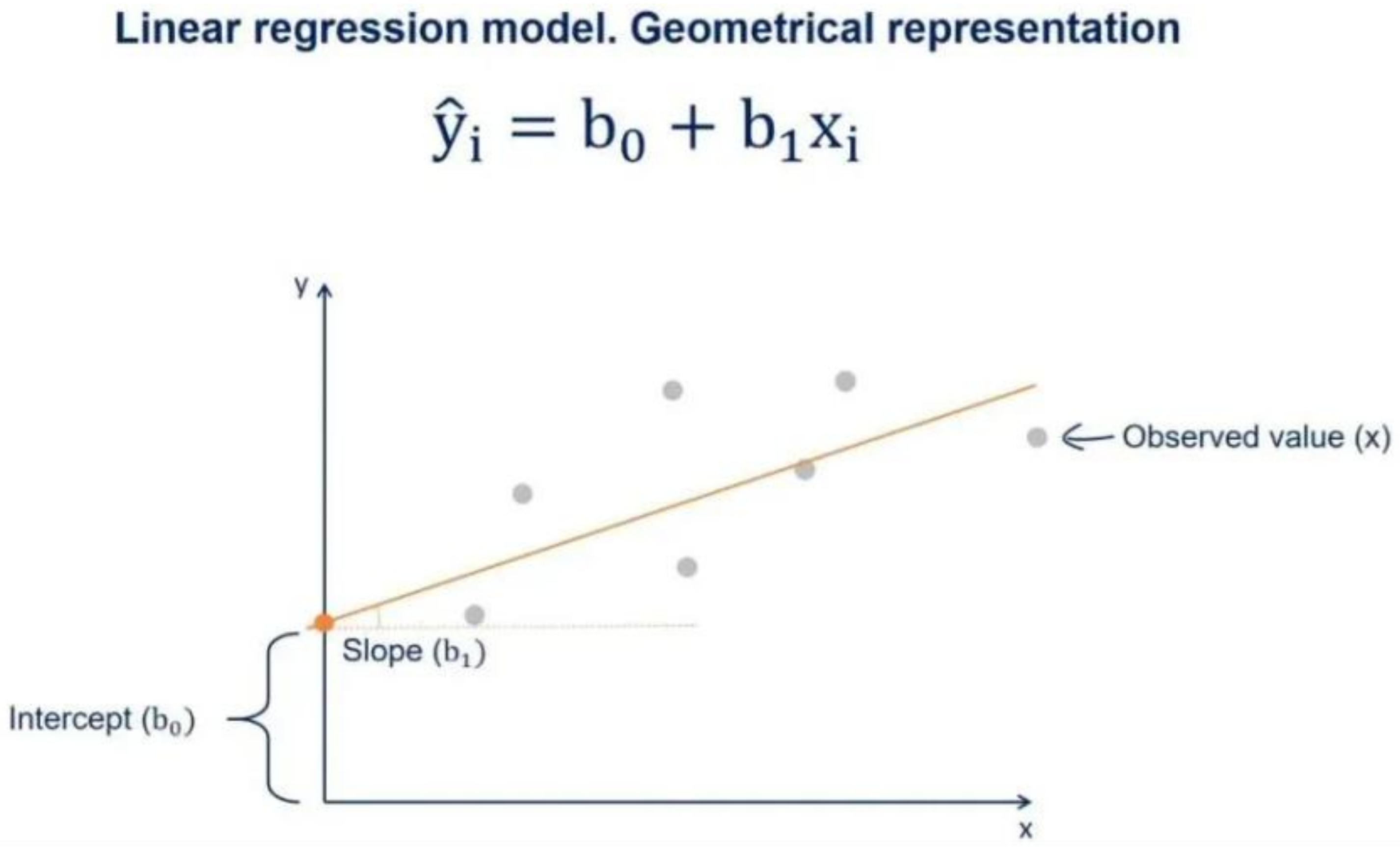
Linear Regression is based on the concept of a linear function,

y = b0 + b1x

where

- y is the dependent variable,
- x is the independent variable,

- b1 is the slope of the line and
- b0 is the y- intercept.



Linear regression model. Geometrical representation
$\hat{y}_i = b_0 + b_1 x_i$
y
Observed value (x)
Slope (b_1)
Intercept (b_0)
x

The Equation of the Line

In linear regression, our prediction function is represented by the equation of a straight line:

$$y = wx + b$$

w (Weight)

Controls the **slope** of the line. It indicates how much the output (Salary) changes for a unit increase in the input (GPA). A higher 'w' means a steeper line and a stronger positive relationship.

b (Bias)

Represents the **y-intercept** of the line. This is the predicted output value when all input features are zero (e.g., minimum base salary regardless of GPA).

Scaling Up: Adding More Features

Real-world scenarios often involve multiple factors. What if we also consider a person's **Age** when predicting salary?

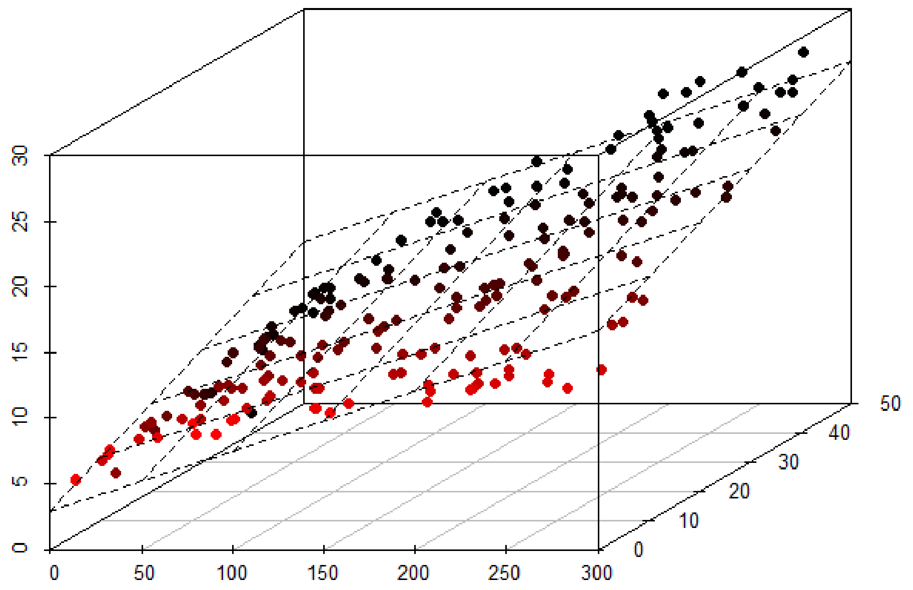
$$y = w_1 \cdot GPA + w_2 \cdot Age + b$$

Multi-dimensional Relationships

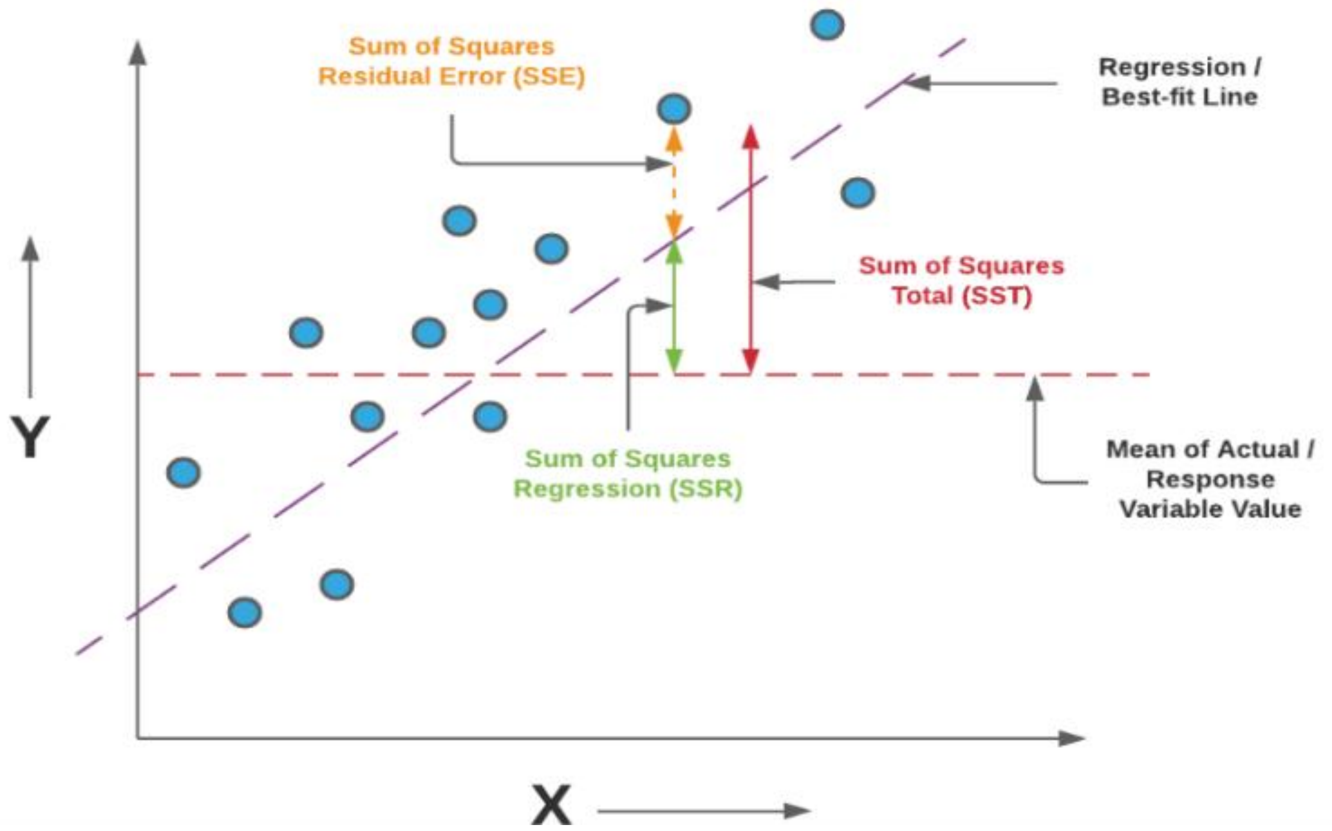
Now, salary prediction depends on **two weights** (w_1 for GPA, w_2 for Age) and one bias. This creates a plane in a 3D space, or a hyperplane in higher dimensions.

Real-life Insights

A high GPA **and** older age could lead to a higher salary. The bias (b) still represents a base income even with zero GPA or age, reflecting a minimum possible value.



How to know if a line is best fitting?



Error = $y_{actual} - y_{predicted}$
(Difference between actual and predicted value)

<u>Mean Square Error (MSE)</u>
Average of sum of all (errors)^2

Cost Function

a metric that we need to minimise to find the optimal solution

cost function for linear regression:

the cost function for linear regression is the mean squared error (MSE), which measures the average squared difference between predicted and actual target values. The objective is to find the coefficient that minimises this MSE and yield the best-fitting line or hyperplane for the given dataset

How Do We Fit the "Best" Line?

If we were to try guessing lines, it would be an endless task.

Random Guess

We start by randomly picking initial values for our weight (w) and bias (b), drawing a line through the data.

Measure Error

We quantify how "wrong" our line is by calculating the difference between our predicted salaries and the actual salaries for all data points. This is called the **Loss Function** (e.g., Mean Squared Error).

Adjust and Repeat?

We could try another random line, measure the error again, and hope it's better. But there are **infinite possible lines**! This trial-and-error approach is highly inefficient and impractical.

The Need for a Systematic Approach: Gradient Descent

We can't rely on random guesses. We need an intelligent method to systematically reduce the error and find the optimal line. This is where **Gradient Descent** comes in.

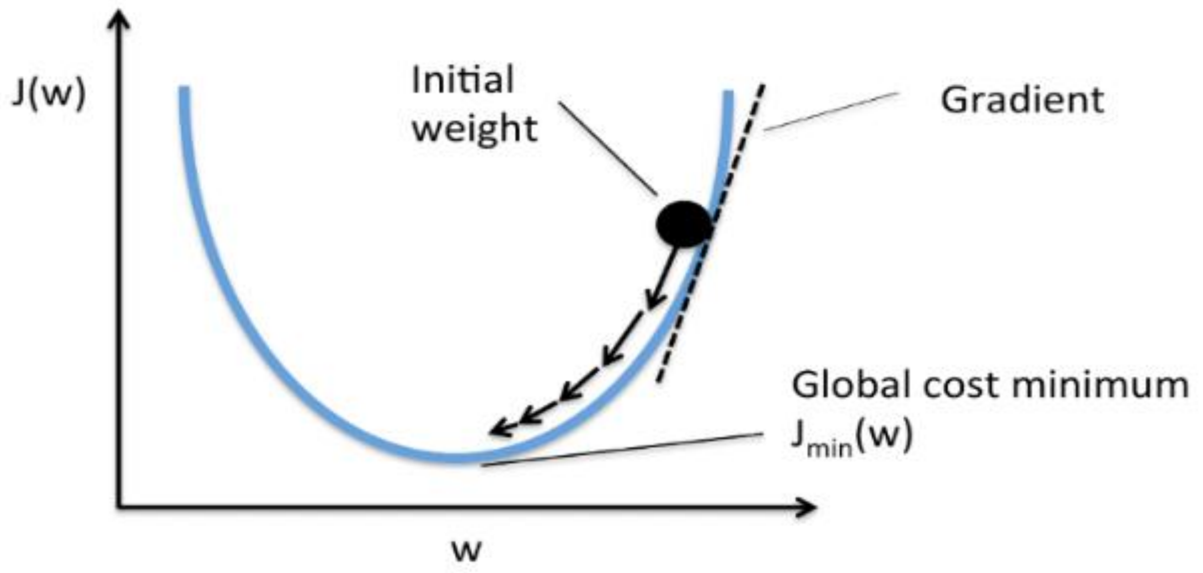
- **Systematic Search:** Gradient Descent provides a structured way to navigate the "error surface" (a visualization of all possible errors for different w and b combinations).
- **Intelligent Reduction:** Instead of random adjustments, it guides our w and b values in the direction that most effectively minimizes the loss.

It's the core optimization algorithm for many machine learning models.

Gradient descent

Gradient descent is an optimization algorithm used in various machine learning models, including linear regression. It aims to find the optimal values for the model's coefficients by minimizing the cost function.

In linear regression, the goal is to create a line or hyperplane that best fits the data by minimizing the error between the predicted values and the actual target values. The cost function quantifies this error.



Understanding the "Gradient" and "Descent"

Gradient Descent is named for its two core concepts:

Gradient

In mathematics, the gradient of a function points in the direction of the **steepest increase**. For our error function, the gradient tells us how to change w and b to make the error go **up** fastest.

Descent

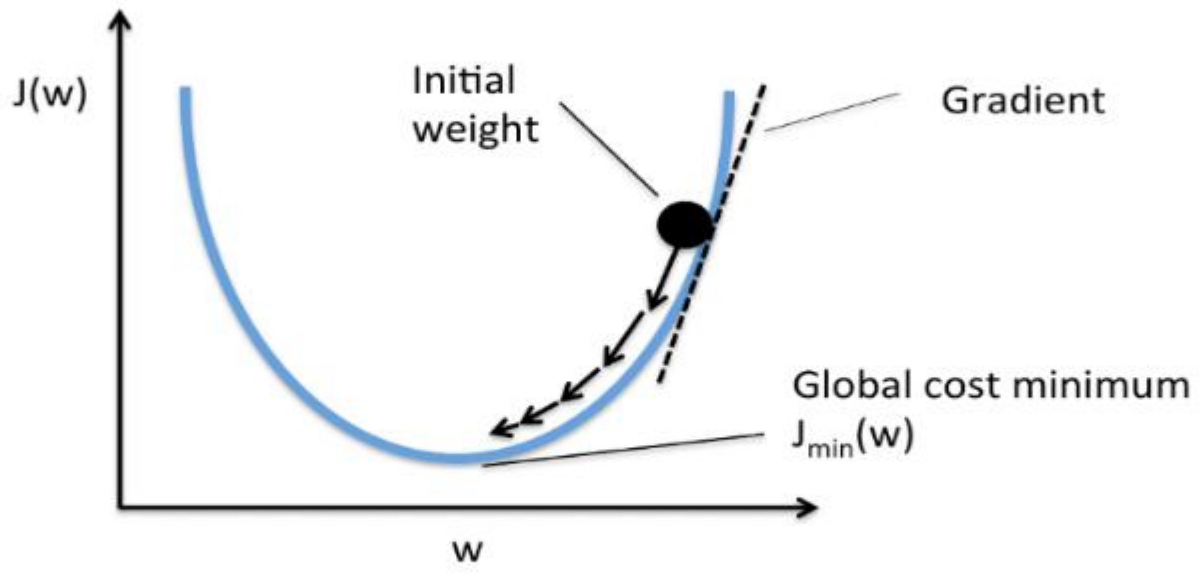
Since our goal is to **minimize** the error, we move in the **opposite direction** of the gradient. This ensures we are always taking steps that lead us downhill towards the lowest point on the error surface.

$$\theta = \theta - \eta \nabla_\theta L(\theta)$$

This formula describes how we update our parameters (weights and bias, represented by θ). We subtract a fraction (η, the learning rate) of the gradient of the Loss function ($\nabla_\theta L(\theta)$).

Steps of Gradient descent

1. Define the cost function
2. Initialize coefficients
3. Calculate the gradient:
   Compute the partial derivatives of the cost function with respect to each coefficient. The gradient indicates the direction and magnitude of the steepest increase of the cost function. It helps in determining how much and in what direction the coefficients should be updated.



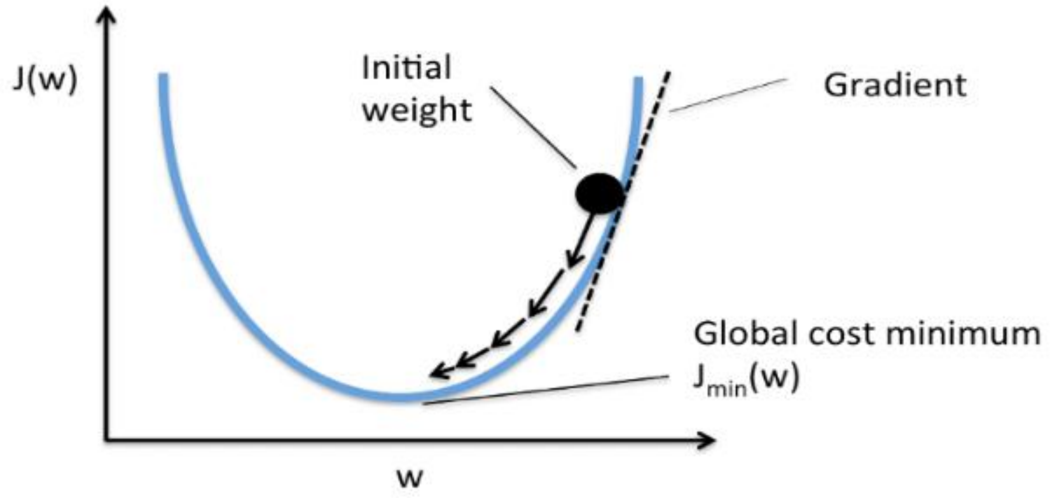
4. Update coefficients:

Adjust the coefficients in the direction opposite to the gradient using a learning rate (alpha).

The learning rate controls the step size in each iteration and prevents overshooting the minimum.

The update rule for each coefficient is as follows:

new_coefficient = old_coefficient - alpha * gradient



For example, for a simple linear regression with coefficients 'b0' (intercept) and 'b1' (slope), the updates would be:

new_b0 = old_b0 - alpha * d(MSE) / d(b0)
new_b1 = old_b1 - alpha * d(MSE) / d(b1)

5. Repeat:

Repeat steps 3 and 4 for a certain number of iterations or until the convergence criteria are met. The convergence criteria can be based on the difference between consecutive cost function values or the magnitude of the gradient.

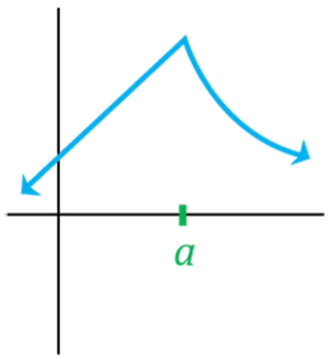
6. Final coefficients:

Once the algorithm converges, the final coefficients represent the best-fitting line or hyperplane that minimizes the cost function and provides the optimal model for the given dataset.

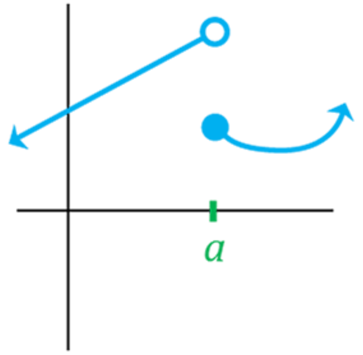
Gradient descent algorithm does not work for all functions. There are two specific requirements. A function has to be:

- differentiable
- convex

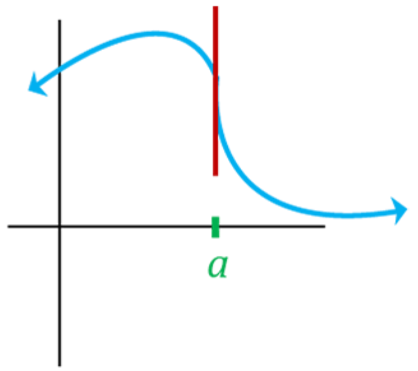
Differentiable : If a function is differentiable, it has a derivative for each point in its domain



Cusp / Corner



Discontinuous



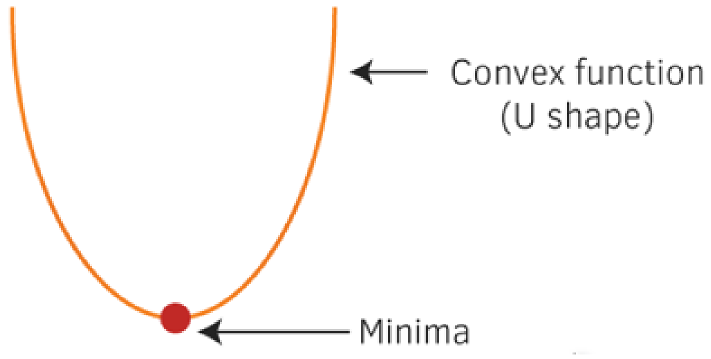
Vertical Tangent

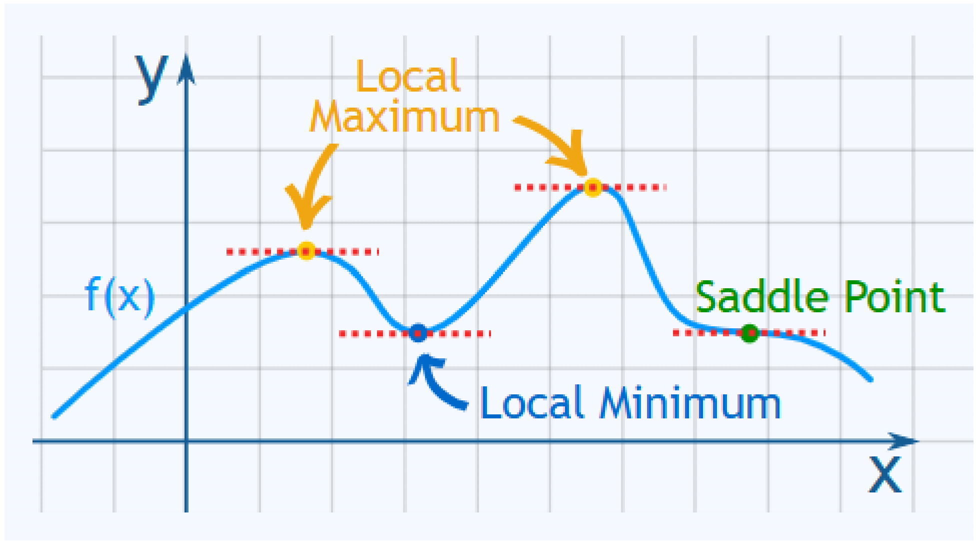
To check mathematically if a univariate function is convex is to calculate the second derivative and check if its value is always bigger than 0

$$\frac{d^2 f(x)}{dx^2} > 0$$

A convex function has only one minimum — the global minimum. No matter where you start, gradient descent will always lead to that lowest point.

❌ But if the function is not convex, it may have many local minima, and gradient descent might stop at the wrong one — not the best solution.





1.Batch Gradient Descent:

- This is a type of gradient descent which processes all the training examples for each iteration of gradient descent.
- But if the number of training examples is large, then batch gradient descent is computationally very expensive.
- If the number of training examples is large, then batch gradient descent is not preferred.
- Instead, we prefer to use stochastic gradient descent or mini-batch gradient descent

2. Stochastic Gradient Descent:

- This is a type of gradient descent which processes 1 training example per iteration.
- The parameters are being updated even after one iteration in which only a single example has been processed.
- Hence this is quite faster than batch gradient descent.
- But again, when the number of training examples is large, even then it processes only one example which can be additional overhead for the system as the number of iterations will be quite large.

3. Mini Batch gradient descent:

This is a type of gradient descent which works faster than both batch gradient descent and stochastic gradient descent. So even if the number of training examples is large, it is processed in batches of b training examples in one go. Thus, it works for larger training examples and that too with lesser number of iterations