

**Modern Education Society's  
College of Engineering, Pune-01**

<b>NAME OF STUDENT:</b>	<b>CLASS:</b>
<b>SEMESTER/YEAR:</b>	<b>ROLL NO:</b>
<b>DATE OF PERFORMANCE:</b>	<b>DATE OF SUBMISSION:</b>
<b>EXAMINED BY:</b>	<b>EXPERIMENT NO:</b>

**TITLE:** ASSIGNMENT ON GRADIENT DESCENT ALGORITHM TO FIND THE LOCAL MINIMA OF A FUNCTION.

**Problem Statement:**

Implement Gradient Descent Algorithm to find the local minima of a function.

For example, find the local minima of the function  $y=(x+3)^2$  starting from the point  $x=2$ .

**Objectives:**

- Understand how the Gradient descent algorithm works and optimize model performance.

**Pre-requisites:**

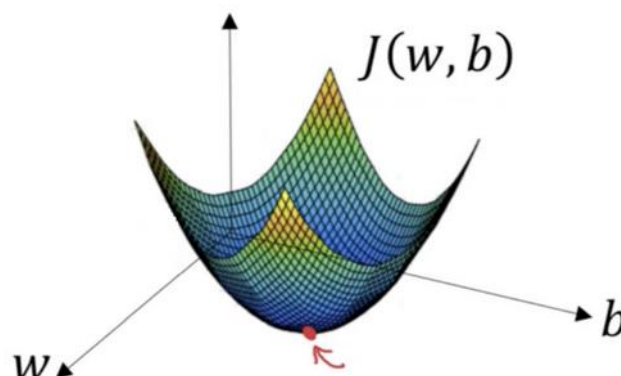
1. Knowledge of python programming.
2. Knowledge of Data Pre-processing.
3. Knowledge of cost functions.

**Description:**

Gradient Descent is an optimization algorithm for finding a local minimum of a differentiable function. Gradient descent in machine learning is simply used to find the values of a function's parameters (coefficients) that minimize a cost function as far as possible. Gradient descent is an optimization algorithm that's used when training a machine learning model. It's based on a convex function and tweaks its parameters iteratively to minimize a given function to its local minimum.

In machine learning, a gradient is a derivative of a function that has more than one input variable. Known as the slope of a function in mathematical terms, the gradient simply measures the change in all weights with regard to the change in error.

Imagine you have a machine learning problem and want to train your algorithm with gradient descent to minimize your cost-function  $J(w, b)$  and reach its local minimum by tweaking its parameters ( $w$  and  $b$ ). The image below shows the horizontal axes representing the parameters ( $w$  and  $b$ ), while the cost function  $J(w, b)$  is represented on the vertical axes. Gradient descent is a convex function.



We know we want to find the values of  $w$  and  $b$  that correspond to the minimum of the cost function (marked with the red arrow). To start finding the right values we initialize  $w$  and  $b$  with some random numbers. Gradient descent then starts at that point (somewhere around the top of our illustration), and it takes one step after another in the steepest downside direction (i.e., from the top to the bottom of the illustration) until it reaches the point where the cost function is as small as possible.

### Function requirements

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

- differentiable
- convex

### Gradient Descent Algorithm

Gradient Descent Algorithm iteratively calculates the next point using gradient at the current position, scales it (by a learning rate) and subtracts obtained value from the current position (makes a step). It subtracts the value because we want to minimise the function (to maximise it would be adding). This process can be written as:

$$p_{n+1} = p_n - \eta \nabla f(p_n)$$

There's an important parameter  $\eta$  which scales the gradient and thus controls the step size. In machine learning, it is called **learning rate** and have a strong influence on performance.

- The smaller learning rate the longer GD converges, or may reach maximum iteration before reaching the optimum point
- If learning rate is too big the algorithm may not converge to the optimal point (jump around) or even to diverge completely.

Gradient Descent method's steps are:

1. choose a starting point (initialisation)
2. calculate gradient at this point
3. make a scaled step in the opposite direction to the gradient (objective: minimise)
4. repeat points 2 and 3 until one of the criteria is met:
  - maximum number of iterations reached
  - step size is smaller than the tolerance (due to scaling or a small gradient).

**This function takes 5 parameters:**

1. **starting point** - in our case, we define it manually but in practice, it is often a random initialisation

2. **gradient function** - has to be specified before-hand
3. **learning rate** - scaling factor for step sizes
4. maximum number of iterations
5. tolerance to conditionally stop the algorithm (in this case a default value is 0.01)

## Example 1 — a quadratic function

Let's take a simple quadratic function defined as:

$$f(x) = x^2 - 4x + 1$$

Because it is an univariate function a gradient function is:

$$\frac{df(x)}{dx} = 2x - 4$$

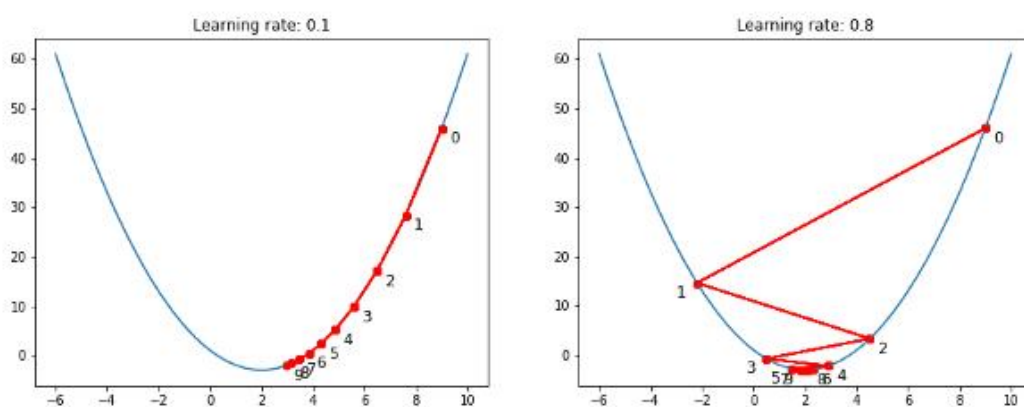
or this function, by taking a learning rate of 0.1 and starting point at  $x=9$  we can easily calculate each step by hand. Let's do it for the first 3 steps:

$$x_1 = 9 - 0.1 \cdot (2 \cdot 9 - 4) = 7.6$$

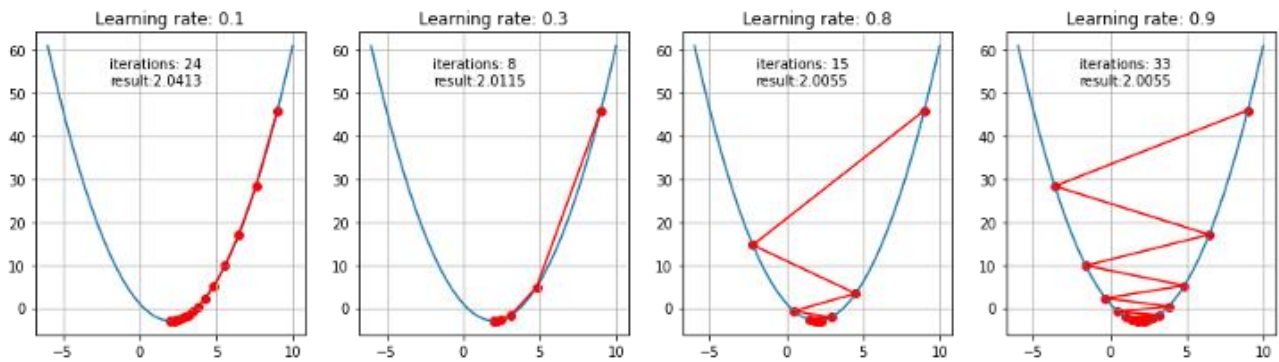
$$x_2 = 7.6 - 0.1 \cdot (2 \cdot 7.6 - 4) = 6.48$$

$$x_3 = 6.48 - 0.1 \cdot (2 \cdot 6.48 - 4) = 5.584$$

The animation below shows steps taken by the GD algorithm for learning rates of 0.1 and 0.8. As you see, for the smaller learning rate, as the algorithm approaches the minimum the steps are getting gradually smaller. For a bigger learning rate, it is jumping from one side to another before converging.



Trajectories, number of iterations and the final converged result (within tolerance) for various learning rates are shown below:



Questions:

1. Compare the Mini-batch Gradient Descent, Stochastic Gradient Descent, and Batch Gradient Descent.
2. Explain how does the Gradient descent work in Linear Regression
3. Does Gradient Descent always converge to an optimum?