**Department of Electronics & Communication Engineering**

Bharati Vidyapeeth

(Deemed to be University)

College of Engineering,

Pune – 411043

**Academic Year: 2022-23**

**Project Based Learning Report**

on

**“Implementation of Gradient Descent”**

Submitted in the partial fulfillment of the requirements

For the Project based learning in (FUZZY LOGIC NUERAL NETWORK AND GENETIC ALGORITHM)

in

Electronics & Communication Engineering

By

**PRN Name of the Student**

**2014111122 AKHIL SUBHASH**

**2014111121 SURYANSH SRIVASTAVA**

**2014111109 RAGHVENDRA SINGH**

Under the guidance of Course In-charge

**Prof**

Department of Electronics & Communication Engineering

Bharati Vidyapeeth

(Deemed to be University)

College of Engineering,

Pune – 4110043

**Academic Year: 2022-23**

**Bharati Vidyapeeth**

**(Deemed to be University)**

**College of Engineering,**

**Pune – 411043**

**DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING**

**CERTIFICATE**

Certified that the Project Based Learning report entitled, “**Implementation of Gradient Descent in Python** ”

is work done by

**PRN Name of the Student**

**2014111122 AKHIL SUBHASH**

**2014111121 SURYANSH SRIVASTAVA**

**2014111109 RAGHVENDRA SINGH**

in partial fulfillment of the requirements for the award of credits for Project Based Learning (PBL) in **Information Theory And Coding** of Bachelor of Technology Semester V, in ELECTRONICS AND COMMUNICATION ENGINEERING

**Date:**

**Prof. V.Kaduskar Dr. Arundhati A .Shinde**

**Course In-charge Professor & Head**

**INDEX**

|  |  |  |
| --- | --- | --- |
| **SERIAL NO.** | **CONTENTS** | **PAGE NO.** |
| 1 | INTRODUCTION | 5 |
| 2 | DEFINITION | 6 |
| 3 | METHODOLOGY | 8 |
| 4 | RESULT AND CONCLUSION | 13 |
| 5 | APPENDIX | 16 |
|  |  |  |

**INTRODUCTION**

**Implemention a Gradient descent in Python to find a local minimum**

[Gradient Descent](https://www.geeksforgeeks.org/gradient-descent-algorithm-and-its-variants/) is an iterative algorithm that is used to minimize a function by finding the optimal parameters. Gradient Descent can be applied to any dimension function i.e. 1-D, 2-D, 3-D. In this article, we will be working on finding global minima for parabolic function (2-D) and will be implementing gradient descent in python to find the optimal parameters for the linear regression equation (1-D). Before diving into the implementation part, let us make sure the set of parameters required to implement the gradient descent algorithm. To implement a gradient descent algorithm, we require a cost function that needs to be minimized, the number of iterations, a learning rate to determine the step size at each iteration while moving towards the minimum, partial derivates for weight & bias to update the parameters at each iteration, and a prediction function.

Gradient descent is an optimization algorithm which is commonly-used to train [machine learning](https://www.ibm.com/cloud/learn/machine-learning) models and [neural networks](https://www.ibm.com/cloud/learn/neural-networks).  Training data helps these models learn over time, and the cost function within gradient descent specifically acts as a barometer, gauging its accuracy with each iteration of parameter updates. Until the function is close to or equal to zero, the model will continue to adjust its parameters to yield the smallest possible error. Once machine learning models are optimized for accuracy, they can be powerful tools for artificial intelligence (AI) and computer science applications.

Till now we have seen the parameters required for gradient descent. Now let us map the parameters with the gradient descent algorithm and work on an example to better understand gradient descent. Let us consider a parabolic equation y=4x2. By looking at the equation we can identify that the parabolic function is minimum at x = 0 i.e. at x=0, y=0. Therefore x=0 is the local minima of the parabolic function y=4x2. Now let us see the algorithm for gradient descent and how we can obtain the local minima by applying gradient descent

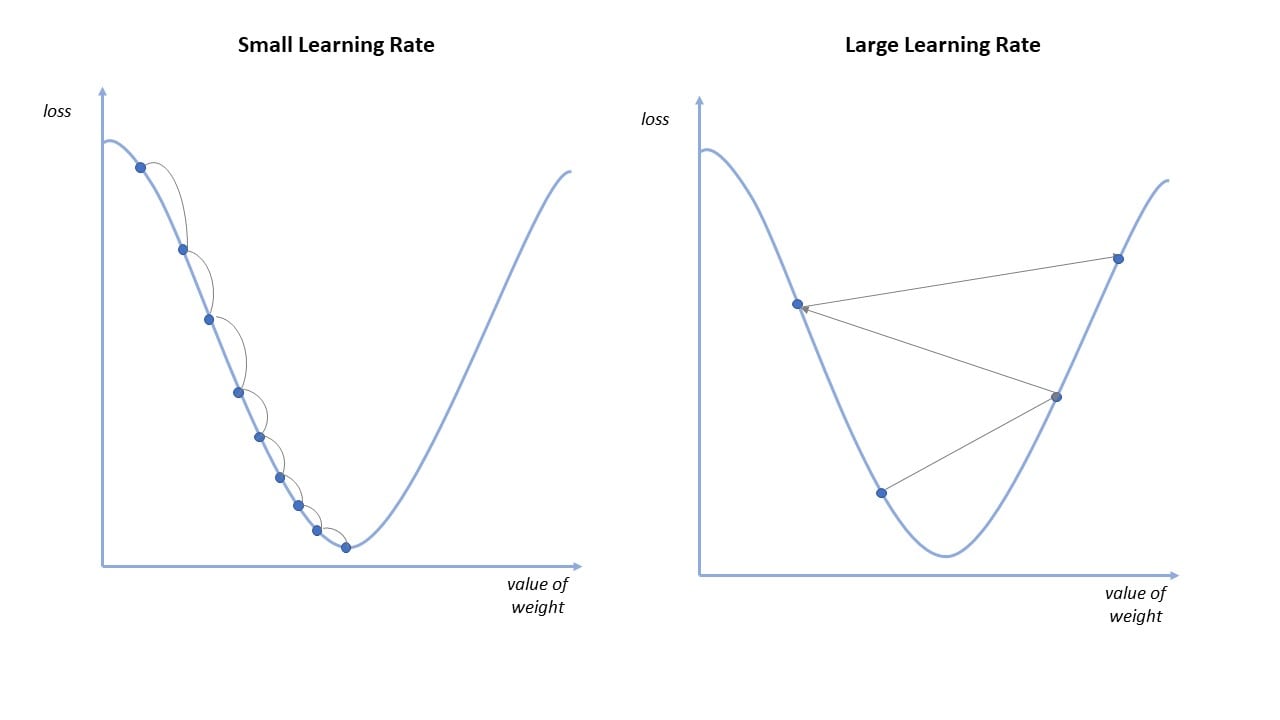
**1**

**DEFINITION**

**How does gradient descent work?**

Before we dive into gradient descent, it may help to review some concepts from linear regression. You may recall the following formula for the slope of a line, which is y = mx + b, where m represents the slope and b is the intercept on the y-axis.

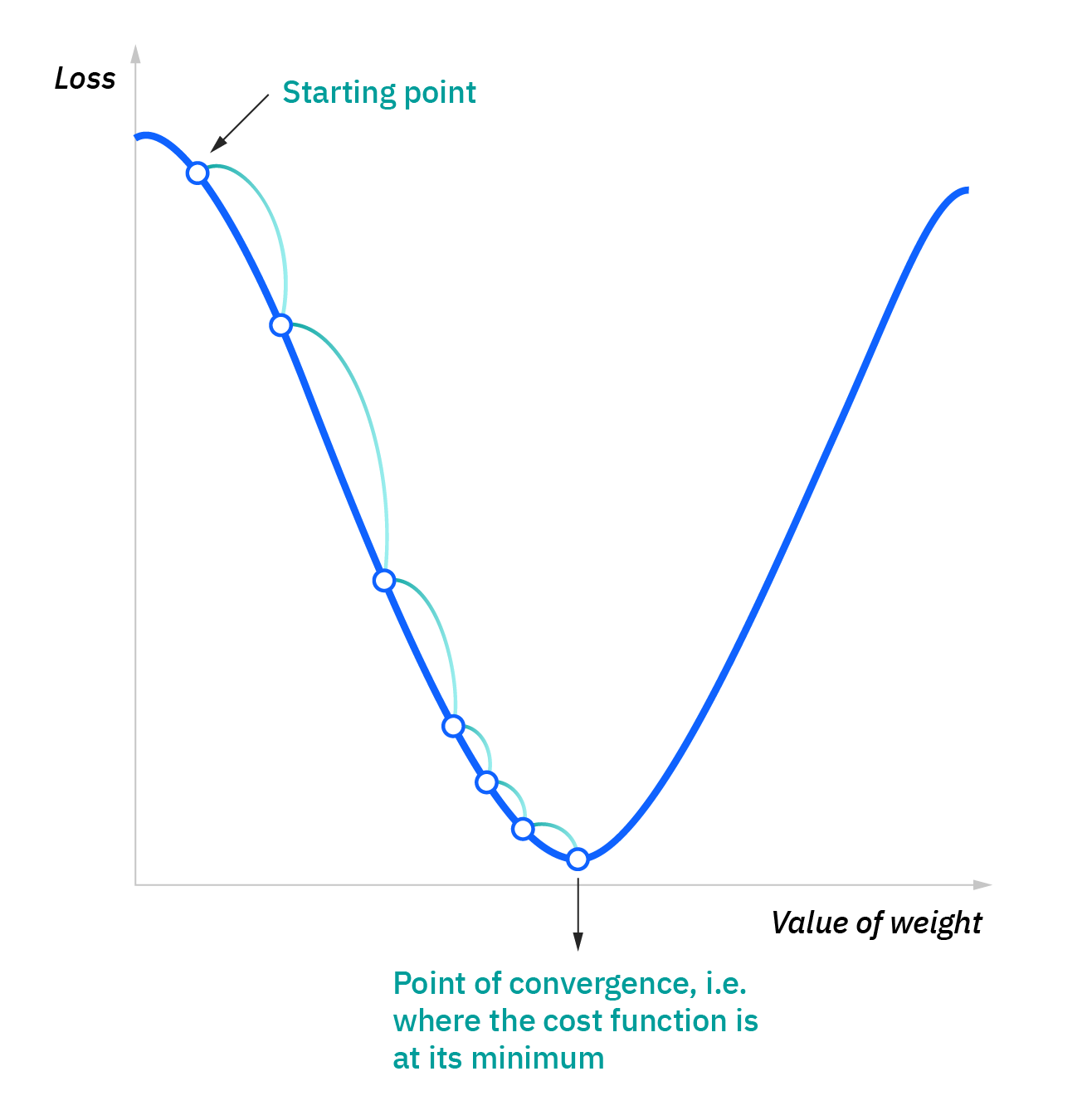
You may also recall plotting a scatterplot in statistics and finding the line of best fit, which required calculating the error between the actual output and the predicted output (y-hat) using the mean squared error formula. The gradient descent algorithm behaves similarly, but it is based on a convex function, such as the one below:



2

The starting point is just an arbitrary point for us to evaluate the performance. From that starting point, we will find the derivative (or slope), and from there, we can use a tangent line to observe the steepness of the slope. The slope will inform the updates to the parameters—i.e. the weights and bias. The slope at the starting point will be steeper, but as new parameters are generated, the steepness should gradually reduce until it reaches the lowest point on the curve, known as the point of convergence.

Similar to finding the line of best fit in linear regression, the goal of gradient descent is to minimize the cost function, or the error between predicted and actual y. In order to do this, it requires two data points—a direction and a learning rate. These factors determine the partial derivative calculations of future iterations, allowing it to gradually arrive at the local or global minimum (i.e. point of convergence). More detail on these components can be found below:



3

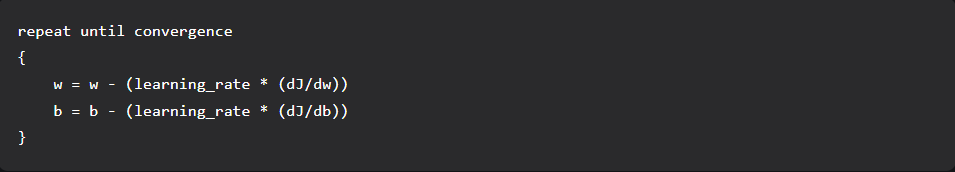
Learning rate (also referred to as step size or the alpha) is the size of the steps that are taken to reach the minimum. This is typically a small value, and it is evaluated and updated based on the behavior of the cost function. High learning rates result in larger steps but risks overshooting the minimum. Conversely, a low learning rate has small step sizes. While it has the advantage of more precision, the number of iterations compromises overall efficiency as this takes more time and computations to reach the minimum.

The cost (or loss) function measures the difference, or error, between actual y and predicted y at its current position. This improves the machine learning model's efficacy by providing feedback to the model so that it can adjust the parameters to minimize the error and find the local or global minimum. It continuously iterates, moving along the direction of steepest descent (or the negative gradient) until the cost function is close to or at zero. At this point, the model will stop learning. Additionally, while the terms, cost function and loss function, are considered synonymous, there is a slight difference between them. It’s worth noting that a loss function refers to the error of one training example, while a cost function calculates the average error across an entire training set.

**Methodology**

**Algorithm for Gradient Descent**

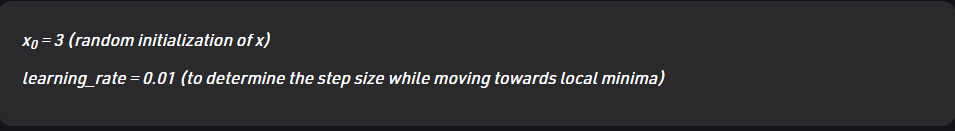
Steps should be made in proportion to the negative of the function gradient (move away from the gradient) at the current point to find local minima. Gradient Ascent is the procedure for approaching a local maximum of a function by taking steps proportional to the positive of the gradient (moving towards the gradient).



4

**Step 1:**

Initializing all the necessary parameters and deriving the gradient function for the parabolic equation 4x2. The derivate of x2 is 2x, so the derivative of the parabolic equation 4x2 will be 8x.

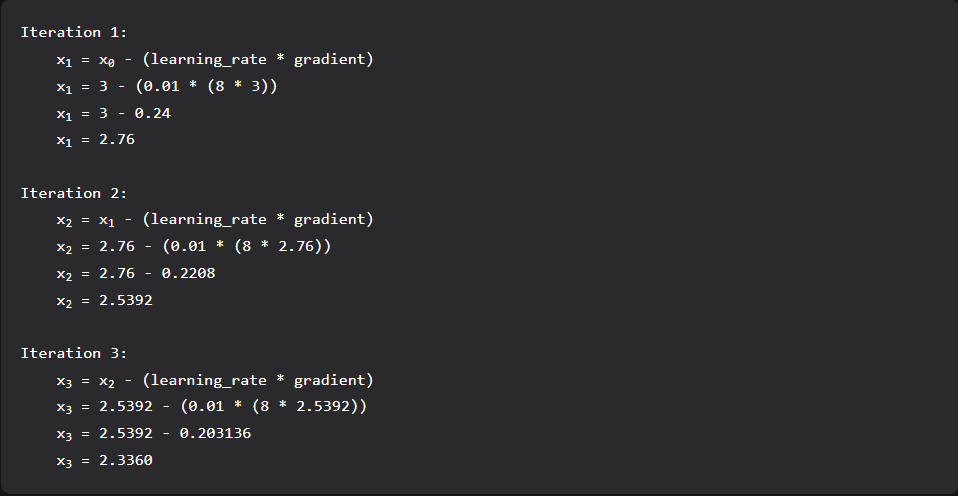


 gradient =    (Calculating the gradient function)

**Step 2:** Let us perform 3 iterations of gradient descent:

For each iteration keep on updating the value of x based on the gradient descent formula.

5

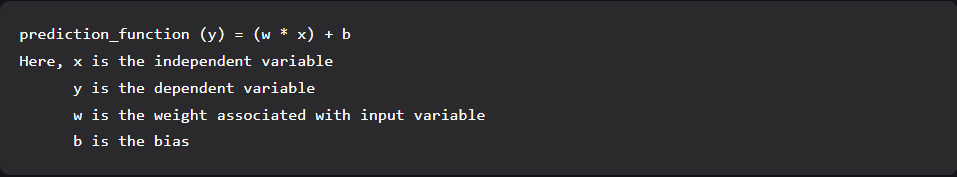
From the above three iterations of gradient descent, we can notice that the value of x is decreasing iteration by iteration and will slowly converge to 0 (local minima) by running the gradient descent for more iterations. Now you might have a question, for how many iterations we should run gradient descent?

We can set a stopping threshold i.e. when the difference between the previous and the present value of x becomes less than the stopping threshold we stop the iterations. When it comes to the implementation of gradient descent for machine learning algorithms and deep learning algorithms we try to minimize the cost function in the algorithms using gradient descent. Now that we are clear with the gradient descent’s internal working, let us look into the python implementation of gradient descent where we will be minimizing the cost function of the linear regression algorithm and finding the best fit line. In our case the parameters are below mentioned:

**Step 3:- Prediction Function**

The prediction function for the linear regression algorithm is a linear equation given by y= wx + b.

6



**Step 4:-Cost Function**

The cost function is used to calculate the loss based on the predictions made. In linear regression, we use mean squared error to calculate the loss. [Mean Squared Error](https://www.geeksforgeeks.org/python-mean-squared-error/) is the sum of the squared differences between the actual and predicted values.

Cost Function (J) = Here, n is the number of samples

Partial Derivatives (Gradients)

Calculating the partial derivates for weight and bias using the cost function.

We get:

**Step 5:-Parameter Updation**

7

Updating the weight and bias by subtracting the multiplication of learning rates and their respective gradients.

**Python Implementation for Gradient Descent**

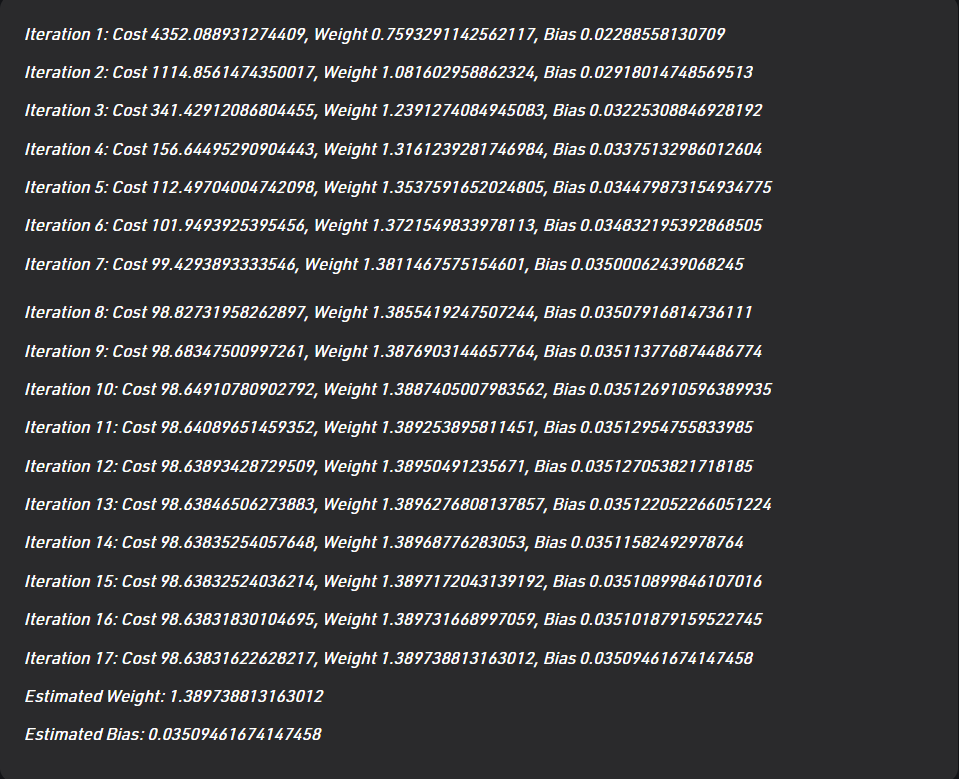
In the implementation part, we will be writing two functions, one will be the cost functions that take the actual output and the predicted output as input and returns the loss, the second will be the actual gradient descent function which takes the independent variable, target variable as input and finds the best fit line using gradient descent algorithm. The iterations, learning\_rate, and stopping threshold are the tuning parameters for the gradient descent algorithm and can be tuned by the user. In the main function, we will be initializing linearly related random data and applying the gradient descent algorithm on the data to find the best fit line. The optimal weight and bias found by using the gradient descent algorithm are later used to plot the best fit line in the main function. The iterations specify the number of times the update of parameters must be done, the stopping threshold is the minimum change of loss between two successive iterations to stop the gradient descent algorithm.

**RESULT AND CONCLUSION**

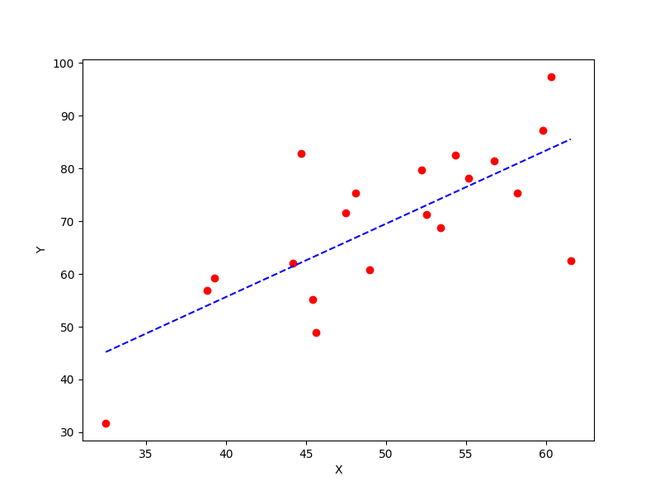
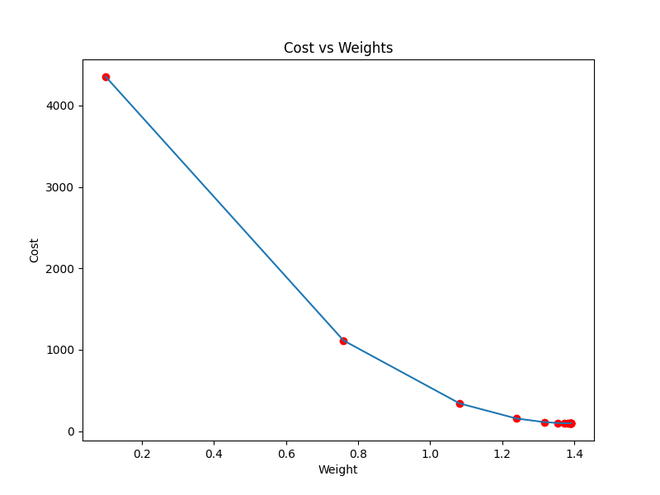
The project helped us understand the concepts of Gradient Descent Algorithm how the code influences the Machine Learning algo. Hence by this algorithm we learned how the Gradient Descent minimizes cost function.

-:OUTPUT:-

8



9



10

**Applications**

* IoT based precision crop suggestion
* To train machine learning modules that utilize neural networks
* Image processing

12

**Appendix A**

# Importing Libraries

**import** numpy as np

**import** matplotlib.pyplot as plt

**def** mean\_squared\_error(y\_true, y\_predicted):

# Calculating the loss or cost

cost **=** np.sum((y\_true**-**y\_predicted)**\*\***2) **/** len(y\_true)

**return** cost

# Gradient Descent Function

# Here iterations, learning\_rate, stopping\_threshold

# are hyperparameters that can be tuned

**def** gradient\_descent(x, y, iterations **=** 1000, learning\_rate **=** 0.0001,

stopping\_threshold **=** 1e**-**6):

# Initializing weight, bias, learning rate and iterations

current\_weight **=** 0.1

current\_bias **=** 0.01

iterations **=** iterations

learning\_rate **=** learning\_rate

n **=** float(len(x))

costs **=** []

weights **=** []

previous\_cost **=** None

# Estimation of optimal parameters

**for** i **in** range(iterations):

# Making predictions

y\_predicted **=** (current\_weight **\*** x) **+** current\_bias

# Calculationg the current cost

current\_cost **=** mean\_squared\_error(y, y\_predicted)

# If the change in cost is less than or equal to

# stopping\_threshold we stop the gradient descent

**if** previous\_cost **and** abs(previous\_cost**-**current\_cost)<**=**stopping\_threshold:

**break**

previous\_cost **=** current\_cost

costs.append(current\_cost)

weights.append(current\_weight)

# Calculating the gradients

weight\_derivative **=** **-**(2**/**n) **\*** sum(x **\*** (y**-**y\_predicted))

bias\_derivative **=** **-**(2**/**n) **\*** sum(y**-**y\_predicted)

# Updating weights and bias

current\_weight **=** current\_weight **-** (learning\_rate **\*** weight\_derivative)

current\_bias **=** current\_bias **-** (learning\_rate **\*** bias\_derivative)

# Printing the parameters for each 1000th iteration

**print**(f"Iteration {i**+**1}: Cost {current\_cost}, Weight \

{current\_weight}, Bias {current\_bias}")

# Visualizing the weights and cost at for all iterations

plt.figure(figsize **=** (8,6))

plt.plot(weights, costs)

plt.scatter(weights, costs, marker**=**'o', color**=**'red')

plt.title("Cost vs Weights")

plt.ylabel("Cost")

plt.xlabel("Weight")

plt.show()

**return** current\_weight, current\_bias

**def** main():

# Data

X **=** np.array([32.50234527, 53.42680403, 61.53035803, 47.47563963, 59.81320787,

55.14218841, 52.21179669, 39.29956669, 48.10504169, 52.55001444,

45.41973014, 54.35163488, 44.1640495 , 58.16847072, 56.72720806,

48.95588857, 44.68719623, 60.29732685, 45.61864377, 38.81681754])

Y **=** np.array([31.70700585, 68.77759598, 62.5623823 , 71.54663223, 87.23092513,

78.21151827, 79.64197305, 59.17148932, 75.3312423 , 71.30087989,

55.16567715, 82.47884676, 62.00892325, 75.39287043, 81.43619216,

60.72360244, 82.89250373, 97.37989686, 48.84715332, 56.87721319])

# Estimating weight and bias using gradient descent

estimated\_weight, eatimated\_bias **=** gradient\_descent(X, Y, iterations**=**2000)

**print**(f"Estimated Weight: {estimated\_weight}\nEstimated Bias: {eatimated\_bias}")

# Making predictions using estimated parameters

Y\_pred **=** estimated\_weight**\***X **+** eatimated\_bias

# Plotting the regression line

plt.figure(figsize **=** (8,6))

plt.scatter(X, Y, marker**=**'o', color**=**'red')

plt.plot([min(X), max(X)], [min(Y\_pred), max(Y\_pred)], color**=**'blue',markerfacecolor**=**'red',

markersize**=**10,linestyle**=**'dashed')

plt.xlabel("X")

plt.ylabel("Y")

plt.show()

**if** \_\_name\_\_**==**"\_\_main\_\_":

main()

**APPENDIX (B)**

Github link : <https://github.com/akhil111pro/Fuzzy-Logic-PBL.git>