Parul® University
VADODARA, GUJARAT

NAAC A++
ACCREDITED UNIVERSITY

FACULTY OF ENGINEERING AND TECHNOLOGY

BACHELOR OF TECHNOLOGY

Pattern Recognition Laboratory
(303105482)

SEMESTER VII
Department of
Artificial Intelligence & Data Science
(A.I & D.S)

योगः कर्मसु कौशलम्

PARUL UNIVERSITY

Laboratory Manual

CERTIFICATE

This is to certify that

*Ms.**Gopi Kota** with Enrollment No.* **2303031247002** has *successfully completed his laboratory experiments in the **Pattern Recognition Lab (303105482)** from the department of* **Artificial Intelligence & Data Science (A.I & D.S)** *during the academic year **2025-2026.***

Date of Submission …… **Staff In charge ……**

Head of Department……

TABLE OF CONTENTS

PRACTICAL: 01

Aim: Implementation of Gradient Descent

Objective:

- To understand and implement the Gradient Descent optimization algorithm.
- To apply Gradient Descent in the context of Linear Regression to minimize the loss function.
- To observe how the algorithm iteratively updates model parameters to achieve better predictions

Description (Theory in brief):

- **Gradient Descent Algorithm**
  - Gradient Descent is an iterative optimization algorithm used to minimize a cost (or loss) function by updating the parameters (weights) of a model in the direction of the negative gradient.
  - It is widely used in machine learning, deep learning, and other fields where functions are complex and cannot be minimized analytically.

- **Purpose**
  - In supervised learning (e.g., linear regression), we try to find the best parameters (weights and bias) that minimize the difference between actual and predicted outputs. This difference is measured by a loss function, and gradient descent helps find the values of parameters that minimize this loss.

How Gradient Descent Works

1. **Start with Initial Parameters**
   - Begin with some initial guess of the parameters (weights), usually 0 or random small values.

2. **Start with Initial Parameters**
   - Calculate how well the current parameters perform using a loss function (e.g., Mean Squared Error in regression).

$$J(w,b) = \frac{1}{2m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2$$

Where: **m**: number of data points,
yi: actual value,

3. **Calculate the Gradient**
   - Compute the gradient (partial derivatives) of the loss function with respect to each parameter.
   - These gradients tell us the direction and rate of change of the loss with respect to the parameters

For linear regression:

$$\frac{\partial J}{\partial w} = -\frac{2}{m}\sum_{i=1}^{m} x_i(y_i - \hat{y}_i)$$

$$\frac{\partial J}{\partial b} = -\frac{2}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)$$

4. **Update the Parameters**
    - Adjust the parameters in the opposite direction of the gradient (to minimize the loss).

$$w := w - \alpha \cdot \frac{\partial J}{\partial w}$$

$$b := b - \alpha \cdot \frac{\partial J}{\partial b}$$

Where:

- α is the learning rate: a small positive value that controls how big the steps are.

5. **Repeat Until Convergence**
    - Repeat steps 2–4 for a number of iterations or until the parameters converge (i.e., updates are very small, or loss stops changing).

Types of Gradient Descent

1. **Batch Gradient Descent**
    - Uses all training data to compute the gradient.
    - More accurate but slower with large datasets.
2. **Stochastic Gradient Descent (SGD)**
    - Uses only one data point at each iteration.
    - Faster but more noisy; good for very large datasets.
3. **Stochastic Gradient Descent (SGD)**
    - Uses a small batch of data at each step.
    - A compromise between speed and stability; widely used in deep learning.

Advantages of Gradient Descent

- Works for large and complex datasets.
- Efficient with vectorized implementations.
- Backbone of most machine learning and deep learning algorithms.

Implementation:

Step 1: Import Required Libraries

- Import necessary Python libraries such as numpy for numerical operations and matplotlib for plotting (optional for visualization).

```
# Step 1: Import Required Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Step 2: Prepare the Dataset

- Create or load the input features (X) and target values (y).

```
# Step 2: Prepare the Dataset
data=pd.read_csv('/content/data.csv')
print(data.head())
```

```
   32.502345269453031  31.70700584656992
0           53.426804          68.777596
1           61.530358          62.562382
2           47.475640          71.546632
3           59.813208          87.230925
4           55.142188          78.211518
```

Step 3: Initialize Parameters

- Initialize model parameters (weights and bias) and define the learning rate and number of iterations.

```
# Step 3: Initialize Parameters
m = 0
c = 0
alpha = 0.0001
epochs = 1000
n = float(len(X))
```

Step 4: Implement the Gradient Descent Loop

```
# Step 4: Apply Gradient Descent
for i in range(epochs):
    y_pred = m * X + c
    error = y_pred - y
    cost = (1/n) * sum(error ** 2)
    md = (2/n) * sum(X * error)
    cd = (2/n) * sum(error)
    m = m - alpha * md
    c = c - alpha * cd
```

- For a given number of iterations (epochs), update the weights and bias using the gradient of the loss function.

Step 5: Implement the Gradient Descent Loop

- Print the final optimized values of the slope and intercept.

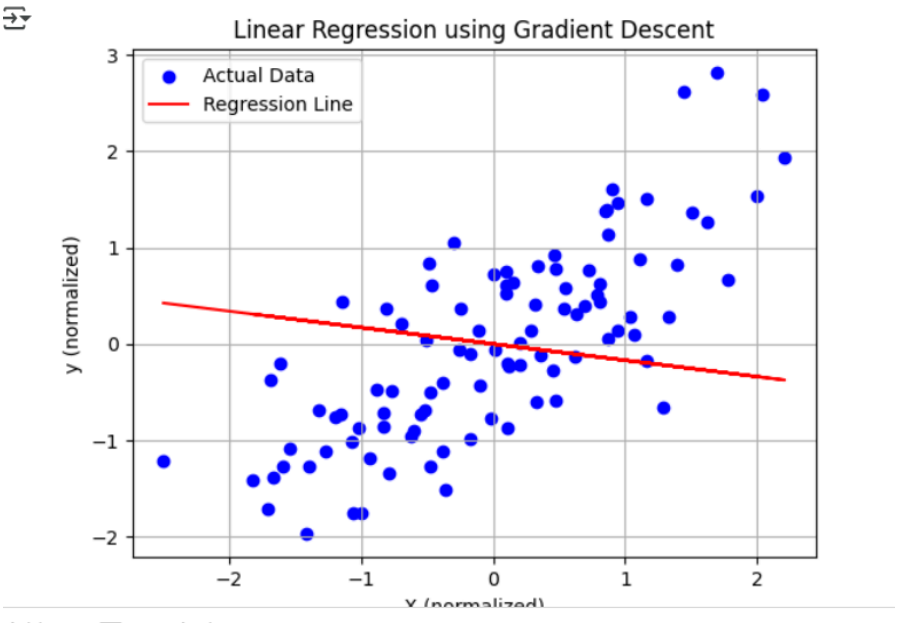
```
# Step 5: Display Final Parameters
print("Optimized slope (m):", m)
print("Optimized intercept (c):", c)
```

```
Optimized slope (m): -0.16963430979471195
Optimized intercept (c): -9.45748529995234e-17
```

Step 6: Visualize the Result

- Plot the actual data points and the regression line to visualize the fit.

```
# Step 6: Visualize the Regression Line (on normalized scale)
plt.scatter(X, y, color='blue', label='Actual Data')
plt.plot(X, m * X + c, color='red', label='Regression Line')
plt.title("Linear Regression using Gradient Descent")
plt.xlabel("X (normalized)")
plt.ylabel("y (normalized)")
plt.legend()
plt.grid(True)
plt.show()
```



Conclusion:

- In this experiment, we successfully implemented the Gradient Descent algorithm to optimize the parameters of a Linear Regression model. We observed how the algorithm iteratively adjusted the slope and intercept to minimize the mean squared error. This demonstrates how gradient descent is a foundational technique in machine learning, especially in training models where minimizing a loss function is essential for accurate predictions.

PRACTICAL: 02

Aim: Implementation of Linear regression using Gradient Descent Algorithm.

Objective:

- To understand the working of Linear Regression and Gradient Descent.
- To apply Gradient Descent to optimize the parameters (slope and intercept) of the Linear Regression model.
- To minimize the prediction error by iteratively updating model parameters.
- To evaluate model performance through visualization and loss reduction.

Description (Theory in brief):

- **Linear Regression**
  - Linear Regression is a supervised learning algorithm used for predicting continuous values. It establishes a linear relationship between the input variable(s) and the output variable. In the simplest case of univariate linear regression, we model the relationship between one input variable x and a continuous output y.
  - The goal is to fit a straight line through the data points that best describes the relationship between the variables.

The Mathematical Model

- In simple linear regression, we assume that the output variable y is a linear function of the input x:

$$= mx + c$$

is the predicted value of y,
is the slope of the line (also called the weight or coefficient),
is the intercept (bias)
is the input feature

- This line represents the hypothesis function h(x), which we want to learn from the data.

Cost Function: Mean Squared Error (MSE)

- To determine how well our hypothesis (regression line) fits the data, we use a cost function that measures the difference between the actual values y and the predicted values ŷ. The most commonly used cost function in linear regression is the Mean Squared Error (MSE):

$$J(m,c) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - (mx_i + c))^2$$

  Where:
  - n is the number of data points,
  - yi is the actual value,
  - ŷi is the predicted value from the current model.

- The objective is to minimize this cost function by finding the best values of mmm and ccc that reduce the prediction error.

Optimization Using Gradient Descent

- Gradient Descent is an iterative optimization algorithm used to find the minimum of a function — in this case, the MSE cost function.

- The idea is to start with initial values for the parameters m and c and iteratively update them in the direction that reduces the cost. This is done by computing the gradient (slope) of the cost function with respect to the parameters.

- Gradient Descent uses the partial derivatives of the cost function to update the parameters:

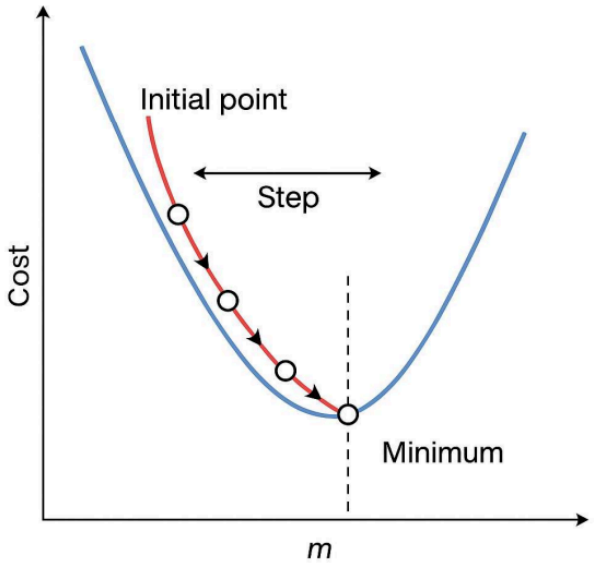
$$m := m - \alpha \cdot \frac{\partial J}{\partial m} \quad \text{and} \quad c := c - \alpha \cdot \frac{\partial J}{\partial c}$$

  Where:
  - α is the learning rate (step size),

  &

$$\frac{\partial J}{\partial m} \text{ and } \frac{\partial J}{\partial c}$$

are the gradients of the cost function.



Derivation of Gradients

- Let's compute the partial derivatives of the MSE cost function:

$$J(m,c) = \frac{1}{n}\sum_{i=1}^{n}(y_i - (mx_i + c))^2$$

- Take derivative w.r.t m:

$$\frac{\partial J}{\partial m} = -\frac{2}{n}\sum_{i=1}^{n} x_i(y_i - (mx_i + c))$$

- Take derivative w.r.t c:

$$\frac{\partial J}{\partial c} = -\frac{2}{n}\sum_{i=1}^{n}(y_i - (mx_i + c))$$

- These gradients indicate the direction and rate of change of the cost function. Gradient Descent updates the parameters in the opposite direction of these gradients to reduce the error.

Convergence and Learning Rate

- The α is crucial in determining the speed and stability of convergence.
- If α is too small, convergence is slow.
- If α is too large, the algorithm might overshoot the minimum or diverge.
- Typically, a good learning rate is found by experimentation or tuning

- Gradient Descent is repeated for a fixed number of epochs (iterations) or until the change in cost becomes negligible, indicating convergence.

Final Hypothesis

- After convergence, the final values of m and c define the best-fit line:

$$\hat{y} = mx + c$$

- This line can now be used for predicting new values or analyzing the linear relationship between the variable

Steps Of Implementation:

Step 1: Import Required Libraries

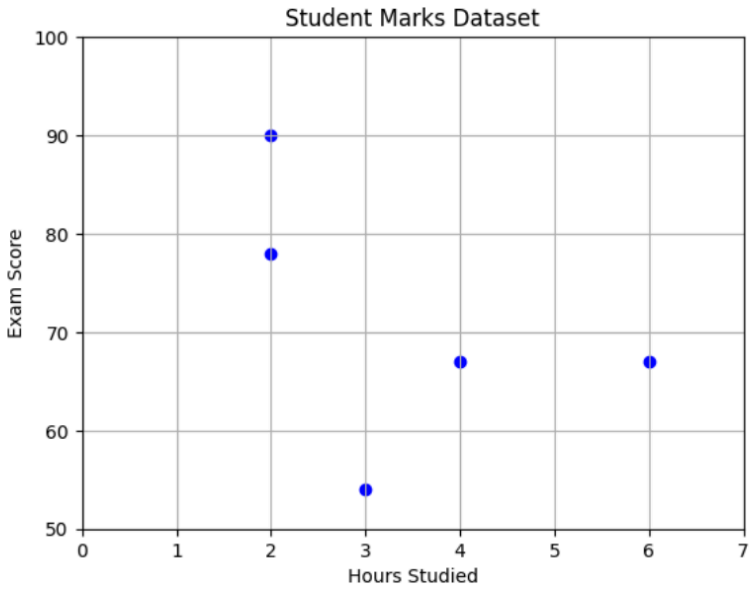
```
# Step 1: Import Required Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Step 2: Prepare the Dataset

```
# Step 2: Prepare the Dataset
# Load data from CSV file
df = pd.read_csv('/content/student.csv')
print(df.head())  # Preview the data

# Extract input (X) and output (Y)
X = df.iloc[:, 0].values  # Feature: Hours Studied
Y = df.iloc[:, 1].values  # Target: Exam Score

# Optional: Visualize the dataset
plt.scatter(X, Y, color='blue')
plt.xlabel("Hours Studied")
plt.ylabel("Exam Score")
plt.title("Student Marks Dataset")
plt.show()
```



Step 3: Initialize Parameters

```
# Step 3: Initialize Parameters
m = 0  # initial slope
c = 0  # initial intercept
alpha = 0.01  # learning rate
epochs = 1000  # number of iterations
n = float(len(X))  # number of samples
```

Step 4: Apply Gradient Descent

```
# Step 4: Apply Gradient Descent
for i in range(epochs):
    y_pred = m * X + c
    error = y_pred - Y
    cost = (1/n) * sum(error ** 2)
    md = (2/n) * sum(X * error)
    cd = (2/n) * sum(error)
    m = m - alpha * md
    c = c - alpha * cd
```

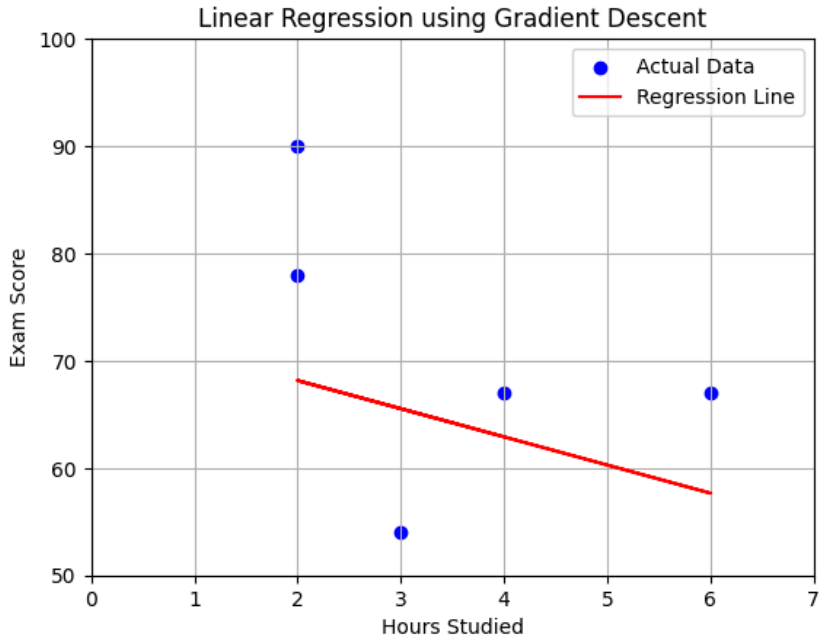
Step 5: Display Final Parameters

```
# Step 5: Display Final Parameters
print("Optimized slope (m):", m)
print("Optimized intercept (c):", c)
print("Final Cost (MSE):", cost)
```

Step 6: Visualize the Regression Line

```
plt.scatter(X, Y, color='blue', label='Actual Data')
plt.plot(X, m * X + c, color='red', label='Regression Line')
plt.xlabel("Hours Studied")
plt.ylabel("Exam Score")
plt.title("Linear Regression using Gradient Descent")
plt.xlim(0, 7)
plt.ylim(50, 100)
plt.legend()
plt.grid(True)
plt.show()
```

```
Optimized slope (m): -2.6237435275757788
Optimized intercept (c): 73.40438827181653
Final Cost (MSE): 277.4954734272738
```



- The red line will fit through the data points and show a strong linear trend — as the number of hours studied increases, the predicted score also increases proportionally.

Conclusion:

- In this experiment, we implemented **Linear Regression** using the **Gradient Descent algorithm**. We initialized parameters, calculated the prediction error using the mean squared error, and minimized this error by updating the parameters iteratively. The final model was able to fit a straight line that best represented the given data. This experiment demonstrated the fundamental concepts of regression and optimization used in many machine learning applications.