

|  |  |
| --- | --- |
| Semester | B.E. Semester VII |
| Subject | Deep Learning |
| Subject Professor In- charge | Dr. Nayana Mahajan |
| Laboratory | M201B |

|  |  |  |  |
| --- | --- | --- | --- |
| Student Name | Harsh Jain | Division | B |
| Roll Number | 22108B0054 | Batch | 4 |
| Grade and Subject  Teacher’s Signature |  |  | |

|  |  |
| --- | --- |
| Experiment Number | 2 |
| Experiment Title | To train and evaluate a single-layer feedforward neural network on a real- world binary classification dataset using Stochastic Gradient Descent (SGD) and Momentum-based Gradient Descent (Momentum GD) as optimization techniques. The objective is to compare and analyze the performance of both optimizers in terms of:   * Convergence Rate: How quickly the training loss decreases over epochs. * Training Speed: The computational efficiency and time taken during training. * Classification Accuracy: The predictive performance on unseen test data.   This study aims to highlight the impact of optimization strategy on neural  network training effectiveness, particularly in low-complexity models such as single-layer networks. |
| Resources / Apparatus Required | Software: Google Colab |
| Algorithm | 1. **Load Dataset**: 2. **Create Binary Target**: 3. **One-Hot Encode Categorical Features**: 4. **Prepare Features and Labels**: 5. **Normalize Features**: 6. **Split Data into Training and Test Sets**: 7. **Define Activation and Loss Functions**: 8. **Initialize Weights and Bias**: 9. **Train the Model**: 10. **Evaluate the Model**: |
| Program code | # Using Sigmoid Activation Function  import pandas as pd  from sklearn.preprocessing import StandardScaler from sklearn.model\_selection import train\_test\_split |



from sklearn.metrics import accuracy\_score import time

import numpy as np

import matplotlib.pyplot as plt

# Load the dataset

df = pd.read\_csv('House\_Price\_Prediction\_Dataset.csv')

# Step 1: Create a binary Price\_Label based on the median price

median\_price = df['Price'].median()

df['Price\_Label'] = (df['Price'] > median\_price).astype(int)

# Step 2: One-Hot Encode the categorical variables (Location, Condition, Garage)

df = pd.get\_dummies(df, columns=['Location', 'Condition', 'Garage'], drop\_first=True)

# Step 3: Prepare the data (Features and Labels)

X = df.drop(['Price', 'Price\_Label', 'Id'], axis=1).values #

Features

y = df['Price\_Label'].values # Binary target (high price vs low price)

# Step 4: Normalize the features scaler = StandardScaler()

X = scaler.fit\_transform(X)

# Step 5: Split the data into training and test sets X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Sigmoid activation function def sigmoid(z):

return 1 / (1 + np.exp(-z))

# Derivative of sigmoid function def sigmoid\_derivative(a):

return a \* (1 - a)

# Binary cross-entropy loss function

def binary\_cross\_entropy(y\_true, y\_pred): epsilon = 1e-8



return -np.mean(y\_true \* np.log(y\_pred + epsilon) + (1 - y\_true) \* np.log(1 - y\_pred + epsilon))

# Initialize weights and bias

def initialize\_weights(n\_features):

W = np.random.randn(n\_features, 1) \* 0.01

b = 0 return W, b

# Training function with chosen optimizer

def train(X, y, optimizer='sgd', lr=0.01, epochs=100, beta=0.9):

n\_samples, n\_features = X.shape

W, b = initialize\_weights(n\_features) loss\_history = []

velocity\_W = np.zeros\_like(W) velocity\_b = 0

start\_time = time.time()

for epoch in range(epochs): for i in range(n\_samples):

xi = X[i].reshape(1, -1) yi = y[i].reshape(1, 1) # Forward pass

z = np.dot(xi, W) + b a = sigmoid(z)

# Compute gradients dz = a - yi

dW = np.dot(xi.T, dz) db = dz

# Update parameters

if optimizer == 'sgd':

W -= lr \* dW b -= lr \* db

elif optimizer == 'momentum':

velocity\_W = beta \* velocity\_W + (1 - beta) \*

dW

velocity\_b = beta \* velocity\_b + (1 - beta) \*

db

W -= lr \* velocity\_W b -= lr \* velocity\_b



# Compute loss at the end of each epoch z\_full = np.dot(X, W) + b

a\_full = sigmoid(z\_full)

loss = binary\_cross\_entropy(y, a\_full) loss\_history.append(loss)

training\_time = time.time() - start\_time return W, b, loss\_history, training\_time

# Prediction function def predict(X, W, b):

z = np.dot(X, W) + b a = sigmoid(z)

return (a > 0.5).astype(int)

# Train using SGD

W\_sgd, b\_sgd, loss\_sgd, time\_sgd = train(X\_train, y\_train, optimizer='sgd', lr=0.01, epochs=100)

# Train using Momentum GD

W\_mom, b\_mom, loss\_mom, time\_mom = train(X\_train, y\_train, optimizer='momentum', lr=0.01, epochs=100, beta=0.9)

# Predictions

y\_pred\_sgd = predict(X\_test, W\_sgd, b\_sgd) y\_pred\_mom = predict(X\_test, W\_mom, b\_mom)

# Accuracy

acc\_sgd = accuracy\_score(y\_test, y\_pred\_sgd) acc\_mom = accuracy\_score(y\_test, y\_pred\_mom)

# Output results

print("Sigmoid Activation Function is being used") print(f"SGD Accuracy: {acc\_sgd\*100:.2f}% | Training Time:

{time\_sgd:.4f} sec")

print(f"Momentum GD Accuracy: {acc\_mom\*100:.2f}% | Training Time: {time\_mom:.4f} sec")

# Plot loss

plt.plot(loss\_sgd, label='SGD Loss') plt.plot(loss\_mom, label='Momentum Loss') plt.xlabel('Epochs')

plt.ylabel('Loss')



|  |  |
| --- | --- |
|  | plt.title('Loss Convergence Comparison') plt.legend()  plt.grid(True) plt.show()  # Output the best optimizer based on the results if acc\_mom > acc\_sgd and time\_mom < time\_sgd:  print("\nMomentum GD is the better optimizer in terms of both accuracy and training time.")  elif acc\_mom < acc\_sgd and time\_mom > time\_sgd: print("\nSGD is the better optimizer in terms of both  accuracy and training time.") else:  print("\nThe optimizers have a mixed performance, and the choice depends on your priorities (accuracy vs. training time).") |
| Output | # Output For Sigmoid Activation Function |
| Conclusion | In our analysis, we utilized a **House Price Prediction Dataset** obtained from Kaggle. We tested two activation functions: **Sigmoid** and **ReLU**.  The results revealed that, for both activation functions, **Momentum SGD** outperformed **SGD** in terms of accuracy. However, **SGD** demonstrated faster training times compared to **Momentum SGD**. |

