EXPERIMENT 6

**OBJECTIVE** : WAP to train and evaluate a Recurrent Neural Network using PyTorch Library to predict the next value in a sample time series dataset.

**DESCRIPTION OF MODEL**

* Model Type: Recurrent Neural Network (RNN)
  + Input Size: 4 (Open, High, Low, Volume)
  + Sequence Length: 25
  + Hidden Layer: 128
  + Output Layer: 1 (predicted closing price)
* Hyperparameters :
  + Epochs-70
  + Learning Rate-0.001
  + Optimizer-Adam
  + Loss Function-MSE (Mean Squared Error)
  + Time Step Window - 25
  + Random Seed – 42

**CODE:**

import pandas as pd

import numpy as np

import torch

import torch.nn as nn

import torch.optim as optim

import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score

from sklearn.model\_selection import train\_test\_split

import random

# Set seeds

torch.manual\_seed(42)

np.random.seed(42)

random.seed(42)

torch.backends.cudnn.deterministic = True

torch.backends.cudnn.benchmark = False

# Load dataset

file\_path = "HistoricalQuotes.csv"

df = pd.read\_csv(file\_path)

# Clean column names

df.columns = df.columns.str.strip().str.lower()

print("Columns:", df.columns.tolist())

# Check required columns

required\_columns = ['date', 'open', 'high', 'low', 'close', 'volume']

missing\_cols = [col for col in required\_columns if col not in df.columns]

if missing\_cols:

    raise KeyError(f"Missing columns in CSV: {missing\_cols}")

# Convert date and clean numeric columns

df['date'] = pd.to\_datetime(df['date'])

for col in ['open', 'high', 'low', 'close', 'volume']:

    df[col] = df[col].astype(str).str.replace(r'[\$,]', '', regex=True).astype(float)

# Sort and index

df.sort\_values('date', inplace=True)

df.set\_index('date', inplace=True)

# Feature/target setup

features = ['open', 'high', 'low', 'volume']

target\_col = ['close']

scaler\_X = MinMaxScaler()

scaler\_y = MinMaxScaler()

X\_scaled = scaler\_X.fit\_transform(df[features])

y\_scaled = scaler\_y.fit\_transform(df[target\_col])

# Create sequences

def create\_sequences(X, y, time\_step):

    X\_seq, y\_seq = [], []

    for i in range(len(X) - time\_step):

        X\_seq.append(X[i:i + time\_step])

        y\_seq.append(y[i + time\_step])

    return np.array(X\_seq), np.array(y\_seq)

time\_step = 25

X\_seq, y\_seq = create\_sequences(X\_scaled, y\_scaled, time\_step)

X\_seq = torch.tensor(X\_seq, dtype=torch.float32)

y\_seq = torch.tensor(y\_seq, dtype=torch.float32)

# Train-test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_seq, y\_seq, test\_size=0.2, shuffle=False)

# Define RNN

class RNN(nn.Module):

    def \_\_init\_\_(self, input\_size, hidden\_size, output\_size):

        super(RNN, self).\_\_init\_\_()

        self.rnn = nn.RNN(input\_size, hidden\_size, batch\_first=True)

        self.fc = nn.Linear(hidden\_size, output\_size)

    def forward(self, x):

        out, \_ = self.rnn(x)

        return self.fc(out[:, -1, :])

input\_size = len(features)

hidden\_size = 128

output\_size = 1

model = RNN(input\_size, hidden\_size, output\_size)

criterion = nn.MSELoss()

optimizer = optim.Adam(model.parameters(), lr=0.001)

train\_losses, train\_accuracy\_scores, train\_r2\_scores = [], [], []

# Train loop

epochs = 70

for epoch in range(epochs):

    model.train()

    optimizer.zero\_grad()

    output = model(X\_train)

    loss = criterion(output, y\_train)

    loss.backward()

    optimizer.step()

    model.eval()

    with torch.no\_grad():

        train\_pred = model(X\_train).detach().numpy()

        train\_true = y\_train.detach().numpy()

        train\_pred\_inv = scaler\_y.inverse\_transform(train\_pred)

        train\_true\_inv = scaler\_y.inverse\_transform(train\_true)

        train\_mse = mean\_squared\_error(train\_true\_inv, train\_pred\_inv)

        train\_mae = mean\_absolute\_error(train\_true\_inv, train\_pred\_inv)

        train\_r2 = r2\_score(train\_true\_inv, train\_pred\_inv)

        train\_accuracy = 100 \* (1 - train\_mae / np.mean(train\_true\_inv))

        train\_losses.append(train\_mse)

        train\_accuracy\_scores.append(train\_accuracy)

        train\_r2\_scores.append(train\_r2)

        print(f"Epoch {epoch+1}/{epochs} | Train MSE: {train\_mse:.2f} | Train MAE: {train\_mae:.2f} | "

              f"Train Accuracy: {train\_accuracy:.2f}% | Train R²: {train\_r2:.4f}")

# Plot training loss

plt.figure(figsize=(8, 5))

plt.plot(train\_losses, label="Train MSE Loss", color='blue')

plt.title("Training Loss Curve (MSE)")

plt.xlabel("Epoch")

plt.ylabel("MSE Loss")

plt.grid(True)

plt.legend()

plt.show()

# Plot accuracy

plt.figure(figsize=(8, 5))

plt.plot(train\_accuracy\_scores, label="Train Accuracy", color='green')

plt.title("Training Accuracy (MAE-based)")

plt.xlabel("Epoch")

plt.ylabel("Accuracy (%)")

plt.grid(True)

plt.legend()

plt.show()

# Evaluate on test set

model.eval()

with torch.no\_grad():

    y\_pred\_test = model(X\_test).detach().numpy()

    y\_test\_true = y\_test.detach().numpy()

    y\_pred\_test\_inv = scaler\_y.inverse\_transform(y\_pred\_test)

    y\_test\_true\_inv = scaler\_y.inverse\_transform(y\_test\_true)

    test\_mse = mean\_squared\_error(y\_test\_true\_inv, y\_pred\_test\_inv)

    test\_mae = mean\_absolute\_error(y\_test\_true\_inv, y\_pred\_test\_inv)

    test\_r2 = r2\_score(y\_test\_true\_inv, y\_pred\_test\_inv)

    test\_accuracy = 100 \* (1 - test\_mae / np.mean(y\_test\_true\_inv))

    print(f"\nTest MSE: {test\_mse:.2f} | Test MAE: {test\_mae:.2f} | "

          f"Test Accuracy: {test\_accuracy:.2f}% | Test R²: {test\_r2:.4f}")

    # Plot predictions

    plt.figure(figsize=(10, 5))

    plt.plot(y\_test\_true\_inv, label='Actual', marker='o')

    plt.plot(y\_pred\_test\_inv, label='Predicted', marker='x')

    plt.title("Actual vs Predicted Stock Price (Test Set)")

    plt.xlabel("Time Step")

    plt.ylabel("Stock Price")

    plt.legend()

    plt.grid(True)

    plt.show()

# Predict next value

last\_input = torch.tensor(X\_scaled[-time\_step:].reshape(1, time\_step, len(features)), dtype=torch.float32)

with torch.no\_grad():

    next\_pred\_scaled = model(last\_input).item()

    next\_price = scaler\_y.inverse\_transform([[next\_pred\_scaled]])[0, 0]

    print(f"\nPredicted next stock price: ${next\_price:.2f}")

**DESCRIPTION OF CODE**

I have used Apple stock price dataset .

* 1. **Dataset Preprocessing:**
* Loaded and cleaned historical stock data (from CSV)
* Removes symbols $ and , , parsed dates, sorted chronologically. Scales features to range [0,1] using MinMaxScaler (ensures better convergence).
  1. **Sequence Creation:**
* Features Used: ['Open', 'High', 'Low', 'Volume']
* Target Variable: 'Close'
* Creates sequences of 25 time steps for input to the RNN.
* Each sequence is paired with the target closing price of the next day(26th) .
  1. **Model Construction:**
* Defined a custom RNN class with PyTorch’s built-in nn.RNN .
* input\_size: 4, features given at each time step ( Open, High, Low, Volume).
* hidden\_size: 128 , neurons in the RNN's "memory" layer.
* output\_size: predicting 1 value, "Close" price.
* Model takes stock data for the last 25 days.
* The RNN reads that time sequence step by step and builds an understanding of the pattern(hidden state).
* The final hidden state is given to a linear layer.
* The linear layer predicts closing price for the next day.
  1. **Training Loop:**
* Model is trained on training data , backpropagation updates weights.
* Metrics are stored and printed.
* Calculated Train MSE, MAE, Accuracy, and R² after each epoch.
* Accuracy defined via MAE: Accuracy = 100 - (MAE / mean\_target) \* 100 .
  1. **Evaluation:**
* Tested on unseen data (test set = last 20%) .
* Inverted scaling to get real prices .
* Plotted predicted vs actual stock prices and loss and accuracy curves .
* Predicted the next stock price .

**PERFORMANCE EVALUATION**

* Performance of model has been evaluated using mean\_squared\_error, mean\_absolute\_error, r2\_score metrics .
* Accuracy has been calculated according to mae and loss has been calculated using mse.
* Loss and accuracy curves have been plotted.
* Actual and predicted values have been plotted .
* Train Accuracy: 97.80% , Test Accuracy: 96.97% , Train R²: 0.9940 , Test R²: 0.9609 .

**MY COMMENTS**

* Model has achieved Train Accuracy: 97.80% , Test Accuracy: 96.97%. Accuracy has been calculated using mae and r2 score as it is a regression problem .
* Accuracy improved by changing the number of hidden neurons from 64 to 128 .
* To keep accuracy same after re-running same python script , random seed has been set.
* To improve accuracy different hidden sizes, layers,time step and dropout can be used , also LSTM/GRU can be used instead of basic RNN