Assignment 9

Name: Vaishnavi Gosavi
Roll No: 71
Class: TY CSAI-A
Batch: 2

Title: Implementation of RNN model for Stock Price Prediction

Description:-

- **Recurrent Neural Networks (RNN):**
  An RNN is a type of neural network that processes sequential data by maintaining a hidden state (memory) that captures information about previous time steps. Each neuron in the RNN not only receives input from the current time step but also from its previous hidden state. This enables RNNs to learn temporal dependencies.

- **Key Features of PCA**
  - *Input layer:* Takes in the stock price features (e.g., Open, High, Low, Close, Volume).
  - *Hidden layer:* Maintains a hidden state that is updated at each time step.
  - *Output layer:* Predicts the stock price at the next time step.
  - *Activation functions:* Typically tanh or ReLU is used in hidden layers.

- **Implementation of RNN**

  1. **Import Libraries and Load Data**
  - Import required libraries (NumPy, Pandas, scikit-learn, TensorFlow/Keras, Matplotlib).
  - Load your historical stock data.

  2. **Data Preprocessing**
  - *Data Cleaning:* Handle missing values if any.
  - *Normalization:* Scale the features (for example, using MinMaxScaler).
  - *Sequence Preparation:*
    - Define a look-back period (window size) to form input sequences.
    - Split data into input sequences (features) and targets (next value).
  - *Train-Test Split:* Divide your data into training and testing sets.

  3. **Building the RNN Model**
  - Initialize a Sequential model.
  - Add one or more RNN layers (e.g., SimpleRNN, or LSTM/GRU if you wish to handle long-term dependencies).
  - Add a Dense layer as the output layer for predicting future stock prices.

4. **Compile the Model**
   - Select a regression loss function (Mean Squared Error, MSE).
   - Choose an optimizer (commonly Adam).

5. **Train the Model**
   - Fit the model using the training data over several epochs with a given batch size.
   - Validate using a subset of data if desired.

6. **Make Predictions and Evaluate the Model**
   - Use the trained model to predict values on the test set.
   - Inverse scale the data to recover original stock prices.
   - Calculate evaluation metrics (e.g., RMSE).
   - Plot actual vs. predicted stock prices.

- ## Algorithm

```
BEGIN
  LOAD stock_data from source
  HANDLE missing_values in stock_data
  NORMALIZE stock_data using scaling method
  FOR each index from look_back to len(stock_data)
    x_sequence = stock_data[index-look_back:index]
    y_target = stock_data[index]  // or next day value
    ADD x_sequence and y_target to dataset
  SPLIT dataset into train and test sets

  INITIALIZE Sequential model
  ADD RNN layer (e.g., SimpleRNN or LSTM) with units and activation function
  ADD Dense layer to output prediction

  COMPILE model with loss = MSE and optimizer = Adam
  TRAIN model on train data for defined epochs with a batch size

  PREDICT on test data using trained model
  INVERSE scale predicted and true values to original scale
  CALCULATE RMSE between predicted and true values

  PLOT true values vs predicted values
END
```

```python
import yfinance as yf
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

df = pd.read_csv('/content/Google_Stock_Price_Train.csv')

import pandas as pd
import matplotlib.pyplot as plt


plt.figure(figsize=(12, 6))
plt.plot(df['Date'], df['Open'], label='Open Price')
plt.xlabel('Date')
plt.ylabel('Stock Price')
plt.title('Google Stock Price')
plt.legend()
plt.grid(True)
plt.show()
```


```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
```

```
data = df['Open'].values.reshape(-1, 1)


scaler = MinMaxScaler(feature_range=(0, 1))
data = scaler.fit_transform(data)


train_size = int(len(data) * 0.8)
train_data, test_data = data[0:train_size, :],
data[train_size:len(data), :]

def create_dataset(dataset, look_back=60):
    X, Y = [], []
    for i in range(len(dataset) - look_back - 1):
        a = dataset[i:(i + look_back), 0]
        X.append(a)
        Y.append(dataset[i + look_back, 0])
    return np.array(X), np.array(Y)

look_back = 60
X_train, Y_train = create_dataset(train_data, look_back)
X_test, Y_test = create_dataset(test_data, look_back)

X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))

model = Sequential()
model.add(SimpleRNN(units=50, return_sequences=True,
input_shape=(X_train.shape[1], 1)))
model.add(SimpleRNN(units=50))
model.add(Dense(1))
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/
rnn.py:200: UserWarning: Do not pass an `input_shape`/`input_dim`
argument to a layer. When using Sequential models, prefer using an
`Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

```
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(X_train, Y_train, epochs=10, batch_size=32)
```

```
Epoch 1/10
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 28ms/step - loss: 0.0547
Epoch 2/10
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - loss: 0.0018
Epoch 3/10
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - loss: 9.4069e-04
Epoch 4/10
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - loss: 0.0012
Epoch 5/10
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - loss: 0.0011
```

```
Epoch 6/10
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - loss: 0.0010
Epoch 7/10
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - loss: 9.4552e-04
Epoch 8/10
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - loss: 8.8362e-04
Epoch 9/10
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - loss: 0.0011
Epoch 10/10
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - loss: 8.9328e-04

<keras.src.callbacks.history.History at 0x7b118b1de190>
```

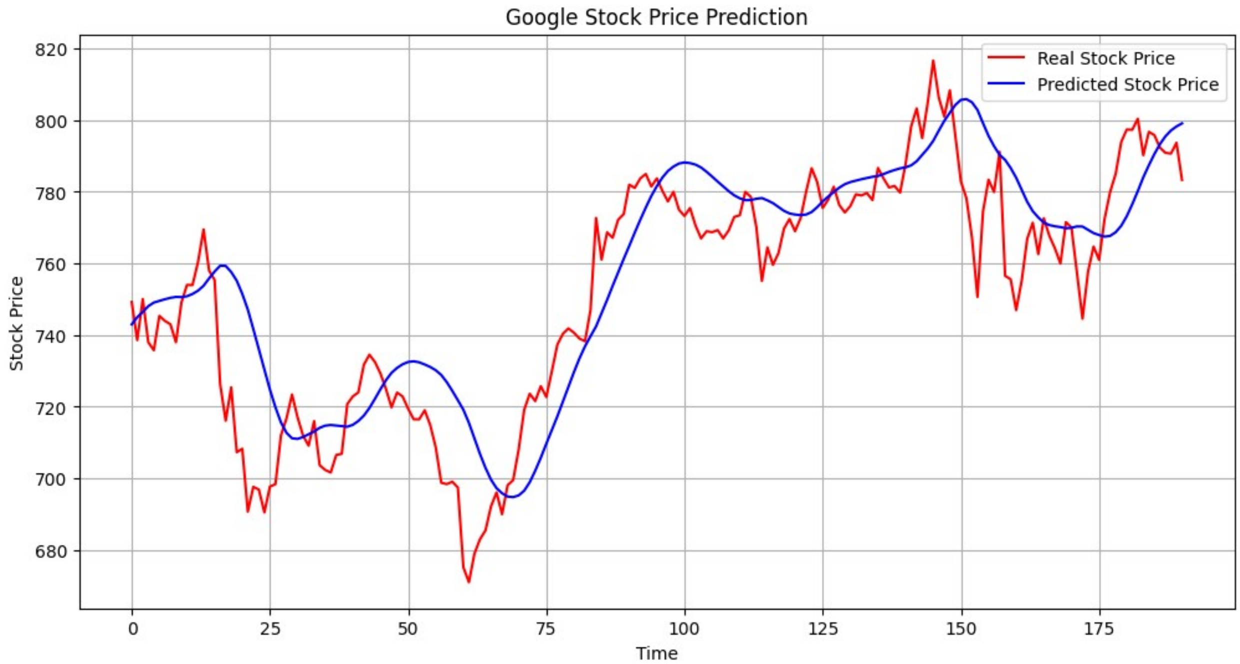
```
predictions = model.predict(X_test)
```

```
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step
```

```
predictions = scaler.inverse_transform(predictions)
Y_test = scaler.inverse_transform([Y_test])
```

```
import matplotlib.pyplot as plt


plt.figure(figsize=(12, 6))
plt.plot(Y_test[0], color='red', label='Real Stock Price')
plt.plot(predictions, color='blue', label='Predicted Stock Price')
plt.title('Google Stock Price Prediction')
plt.xlabel('Time')
plt.ylabel('Stock Price')
plt.legend()
plt.grid(True)
plt.show()
```




```
rmse = np.sqrt(np.mean((predictions - Y_test)**2))
print('RMSE:', rmse)

RMSE: 45.84129635805651
```