Assignment 4:
Recurrent neural network (RNN) Use the Google stock prices dataset and design a time series analysis and prediction system using RNN.

Objective

The objective of this assignment is to design and implement a Recurrent Neural Network (RNN) for time series forecasting using historical Google (GOOG) stock prices. The model should learn from past stock data to predict future closing prices accurately.

Theory

Recurrent Neural Networks (RNNs) are designed for sequential data where previous inputs influence future outputs. Unlike traditional feedforward neural networks, RNNs maintain a hidden state that gets updated with each input step, giving them a form of memory. This makes them ideal for time series tasks like stock prediction. In this project, a simple RNN is trained on 60-day windows of Google's historical closing prices to predict the price on the 61st day. The model is trained using normalized data and evaluated using Mean Squared Error (MSE).

Workflow

1. Data Collection: Google stock data is retrieved using the yfinance library (2018–2024).
2. Preprocessing:
    - Use only the 'Close' column.
    - Normalize values with MinMaxScaler.
    - Create input sequences (60 previous days → 1 target day).
3. Model Design:
    - Use a SimpleRNN layer with 50 units and a Dense output layer.
4. Training:
    - Use 80% of data for training, 20% for testing.
    - Train the model using Adam optimizer and MSE loss.
5. Prediction & Visualization:
    - Predict on test data.
    - Plot predicted prices vs real prices.

Source Code:

```
import yfinance as yf
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense

# Download Google stock data (GOOG) for the last 5 years
print("Downloading stock data...")
data = yf.download('GOOG', start='2018-01-01', end='2024-01-01')
data_close = data[['Close']]

# Normalize the data
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(data_close)

# Create sequences
def create_sequences(data, seq_length):
    x, y = [], []
    for i in range(seq_length, len(data)):
        x.append(data[i-seq_length:i, 0])
        y.append(data[i, 0])
    return np.array(x), np.array(y)

sequence_length = 60
X, y = create_sequences(scaled_data, sequence_length)

# Reshape for RNN input
X = np.reshape(X, (X.shape[0], X.shape[1], 1))

# Train-Test Split
train_size = int(len(X) * 0.8)
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]

# Build the RNN model
model = Sequential([
    SimpleRNN(units=50, return_sequences=False, input_shape=(X_train.shape[1], 1)),
    Dense(1)
])

model.compile(optimizer='adam', loss='mean_squared_error')
model.summary()

# Train the model
print("Training the model...")
history = model.fit(X_train, y_train, epochs=20, batch_size=32, validation_data=(X_test, y_test))

# Predict
```

```
print("Predicting stock prices...")
predicted_stock_price = model.predict(X_test)
predicted_stock_price = scaler.inverse_transform(predicted_stock_price.reshape(-1, 1))
real_stock_price = scaler.inverse_transform(y_test.reshape(-1, 1))

# Plotting results
plt.figure(figsize=(12,6))
plt.plot(real_stock_price, color='red', label='Real Google Stock Price')
plt.plot(predicted_stock_price, color='blue', label='Predicted Google Stock Price')
plt.title('Google Stock Price Prediction using RNN')
plt.xlabel('Time')
plt.ylabel('Google Stock Price')
plt.legend()
plt.tight_layout()
plt.show()
```

Output:

Results

- The model successfully learned patterns from the past data.
- The predicted stock prices closely followed the actual prices with minor deviations.
- The line graph showed a strong alignment between real and predicted prices, demonstrating the model's ability to generalize patterns over time.

Conclusion

This assignment demonstrates the effectiveness of Recurrent Neural Networks in time series forecasting, especially for financial data. Although simple RNNs may not capture long-term dependencies as well as LSTMs or GRUs, they still offer valuable insights for short-term forecasting. Further improvements could include using more advanced RNN variants, tuning hyperparameters, or incorporating additional features like volume or technical indicators.