**Practical 7**

**RNN for sequence analysis**

**7**

**Aim:** Demonstrate recurrent neural network that learns to perform sequence analysis for stock price.

**Description:**

**RNN**

1. **Sequential Data Processing**: Recurrent Neural Networks (RNNs) are designed for sequential data processing, making them suitable for tasks like time series prediction, speech recognition, and natural language processing.
2. **Recurrent Connections**: RNNs utilize recurrent connections to preserve information about previous states, enabling them to capture temporal dependencies in the data.
3. **Variable-Length Inputs**: RNNs can handle inputs of variable lengths, making them versatile for tasks where input sequences may vary in length.
4. **Vanishing Gradient Problem**: RNNs may suffer from the vanishing gradient problem, where gradients diminish exponentially over time, affecting the learning of long-range dependencies.
5. **LSTM and GRU**: To address the vanishing gradient problem, variants like Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU) were developed, which incorporate gating mechanisms to regulate information flow.
6. **Bidirectional RNNs**: Bidirectional RNNs process sequences in both forward and backward directions, enhancing their ability to capture context from both past and future inputs.
7. **Applications**: RNNs find applications in various fields such as natural language processing for sentiment analysis, machine translation, and speech recognition, as well as in finance for stock price prediction and in biology for sequence analysis.

**Code:**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

from keras.models import Sequential

from keras.layers import Dense, LSTM, Dropout

from sklearn.preprocessing import MinMaxScaler

# Read training dataset

dataset\_train = pd.read\_csv('Google\_Stock\_price\_train.csv')

training\_set = dataset\_train.iloc[:, 1:2].values

# Scale the training set

sc = MinMaxScaler(feature\_range=(0,1))

training\_set\_scaled = sc.fit\_transform(training\_set)

# Create X\_train and Y\_train

X\_train = []

Y\_train = []

for i in range(60, 1258):

X\_train.append(training\_set\_scaled[i-60:i, 0])

Y\_train.append(training\_set\_scaled[i, 0])

X\_train, Y\_train = np.array(X\_train), np.array(Y\_train)

# Reshape X\_train for LSTM

X\_train = np.reshape(X\_train, (X\_train.shape[0], X\_train.shape[1], 1))

# Build the LSTM model

regressor = Sequential()

regressor.add(LSTM(units=50, return\_sequences=True, input\_shape=(X\_train.shape[1], 1)))

regressor.add(Dropout(0.2))

regressor.add(LSTM(units=50, return\_sequences=True))

regressor.add(Dropout(0.2))

regressor.add(LSTM(units=50, return\_sequences=True))

regressor.add(Dropout(0.2))

regressor.add(LSTM(units=50))

regressor.add(Dropout(0.2))

regressor.add(Dense(units=1))

regressor.compile(optimizer='adam', loss='mean\_squared\_error')

# Train the model

regressor.fit(X\_train, Y\_train, epochs=100, batch\_size=32)

# Read test dataset

dataset\_test = pd.read\_csv('Google\_Stock\_price\_Test.csv')

real\_stock\_price = dataset\_test.iloc[:, 1:2].values

# Concatenate total dataset

dataset\_total = pd.concat((dataset\_train['Open'], dataset\_test['Open']), axis=0)

inputs = dataset\_total[len(dataset\_total)-len(dataset\_test)-60:].values

inputs = inputs.reshape(-1, 1)

inputs = sc.transform(inputs)

# Create X\_test

X\_test = []

for i in range(60, 80):

X\_test.append(inputs[i-60:i, 0])

X\_test = np.array(X\_test)

X\_test = np.reshape(X\_test, (X\_test.shape[0], X\_test.shape[1], 1))

# Predict stock prices

predicted\_stock\_price = regressor.predict(X\_test)

predicted\_stock\_price = sc.inverse\_transform(predicted\_stock\_price)

# Visualize results

plt.plot(real\_stock\_price, color='red', label='Real Google Stock Price')

plt.plot(predicted\_stock\_price, color='blue', label='Predicted Stock Price')

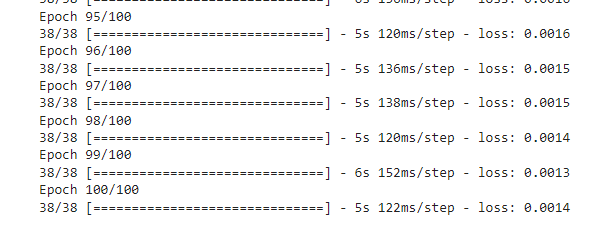
plt.xlabel('Time')

plt.ylabel('Google Stock Price')

plt.legend()

plt.show()

**Output:**



A graph showing a line and a line

Description automatically generated with medium confidence

**Learning:**

Data preprocessing involves scaling the training dataset using MinMaxScaler.

The LSTM model architecture comprises four layers with dropout regularization.

Model compilation uses the Adam optimizer and mean squared error loss.

Training the model on the training set for 100 epochs with a batch size of 32.

Finally, the model predicts stock prices on a test dataset, followed by inverse scaling and visualization of results using Matplotlib.