Batch: HO-DL-1

Roll Number: 16010421073 **Experiment Number: 8**

Name: Keyur Patel

Title of the Experiment: LSTM

Program and results:

Importing libraries

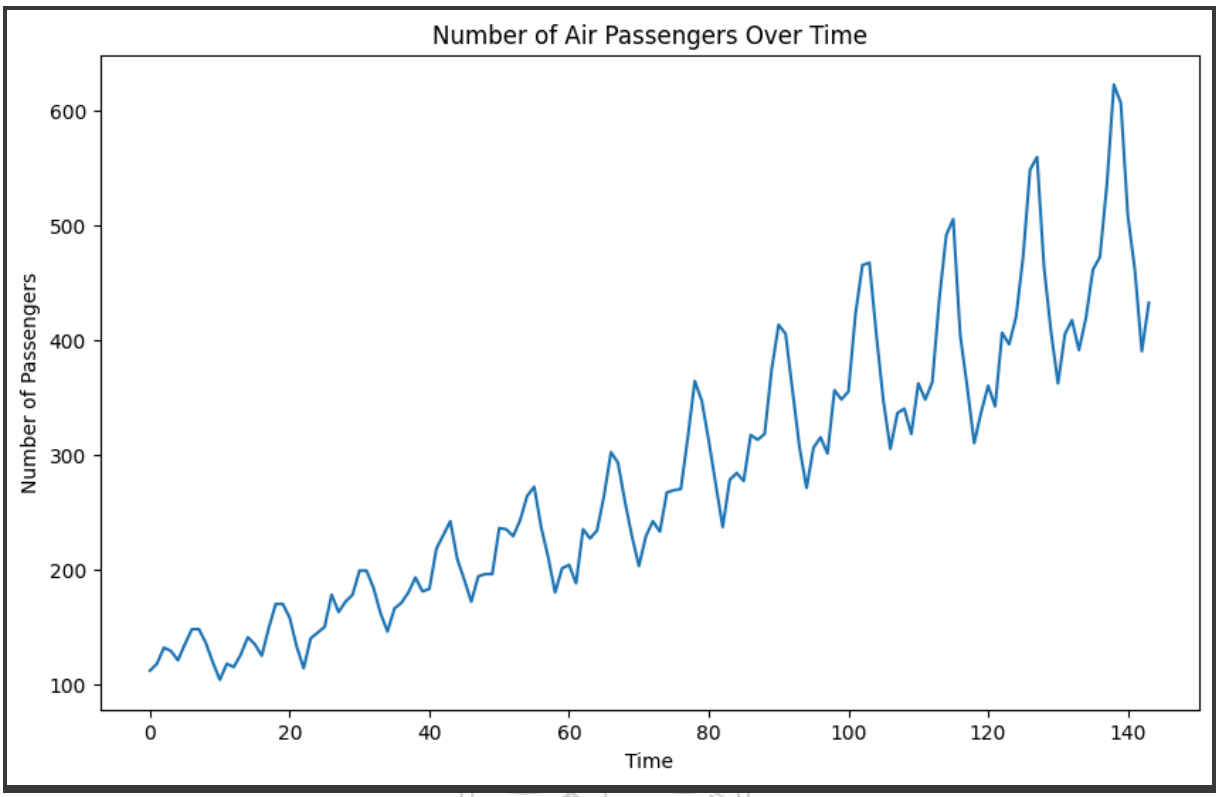
```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.preprocessing import MinMaxScaler

from sklearn.model_selection import train_test_split

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense
```

Load the Air Passengers dataset

```
data = sns.load_dataset("flights")
```

Pre-process and visualize the dataset

```
plt.figure(figsize=(10, 6))

plt.plot(data['passengers'])

plt.title('Number of Air Passengers Over Time')

plt.xlabel('Time')

plt.ylabel('Number of Passengers')

plt.show()
```



Normalize the data using Min-Max scaling

```
scaler = MinMaxScaler(feature_range=(0, 1))

scaled_data =
scaler.fit_transform(data['passengers'].values.reshape(-1, 1))
```

Form the Training and Testing Data

```
train_size = int(len(scaled_data) * 0.8)

test_size = len(scaled_data) - train_size

train_data, test_data = scaled_data[0:train_size],
scaled_data[train_size:len(scaled_data)]


look_back = 5
```

Create the training and testing datasets with lookback

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

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))
```


Develop and train LSTM model

```
model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=(look_back, 1)))
model.add(LSTM(units=50))
model.add(Dense(units=1))
model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(X_train, Y_train, epochs=100, batch_size=32)
```

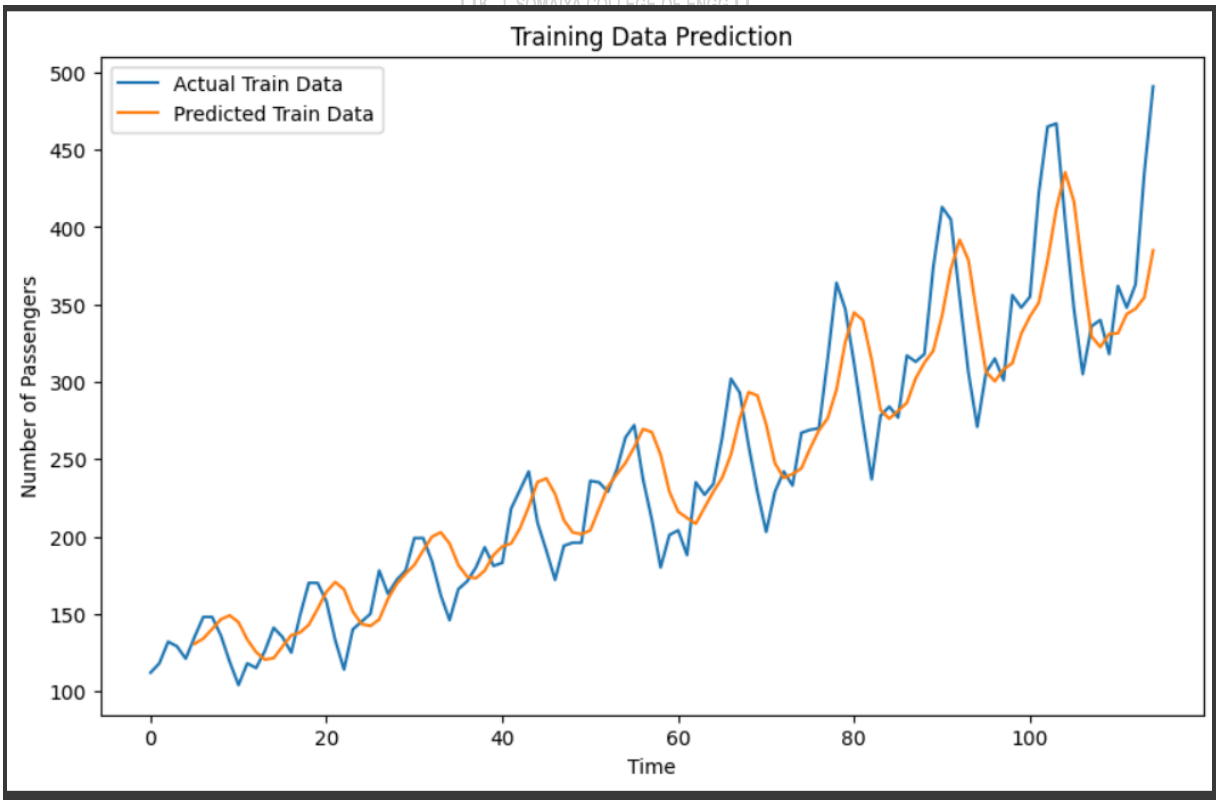
```
Epoch 1/100
4/4 [==============================] - 4s 9ms/step - loss: 0.0930
Epoch 2/100
4/4 [==============================] - 0s 8ms/step - loss: 0.0567
Epoch 3/100
4/4 [==============================] - 0s 8ms/step - loss: 0.0279
Epoch 4/100
4/4 [==============================] - 0s 7ms/step - loss: 0.0145
Epoch 5/100
4/4 [==============================] - 0s 7ms/step - loss: 0.0177
Epoch 6/100
4/4 [==============================] - 0s 6ms/step - loss: 0.0176
Epoch 7/100
4/4 [==============================] - 0s 7ms/step - loss: 0.0128
Epoch 8/100
4/4 [==============================] - 0s 10ms/step - loss: 0.0116
Epoch 9/100
4/4 [==============================] - 0s 7ms/step - loss: 0.0119
Epoch 10/100
4/4 [==============================] - 0s 7ms/step - loss: 0.0109
Epoch 11/100
4/4 [==============================] - 0s 7ms/step - loss: 0.0098
Epoch 12/100
4/4 [==============================] - 0s 7ms/step - loss: 0.0093
Epoch 13/100
4/4 [==============================] - 0s 7ms/step - loss: 0.0093
Epoch 14/100
4/4 [==============================] - 0s 8ms/step - loss: 0.0087
Epoch 15/100
4/4 [==============================] - 0s 7ms/step - loss: 0.0083
Epoch 16/100
4/4 [==============================] - 0s 8ms/step - loss: 0.0087
Epoch 17/100
4/4 [==============================] - 0s 7ms/step - loss: 0.0086
Epoch 18/100
4/4 [==============================] - 0s 7ms/step - loss: 0.0083
Epoch 19/100
4/4 [==============================] - 0s 7ms/step - loss: 0.0085
Epoch 20/100
4/4 [==============================] - 0s 7ms/step - loss: 0.0082
Epoch 21/100
4/4 [==============================] - 0s 7ms/step - loss: 0.0082
Epoch 22/100
4/4 [==============================] - 0s 9ms/step - loss: 0.0083
Epoch 23/100
4/4 [==============================] - 0s 7ms/step - loss: 0.0082
Epoch 24/100
```

Plot the predictions for training and testing data

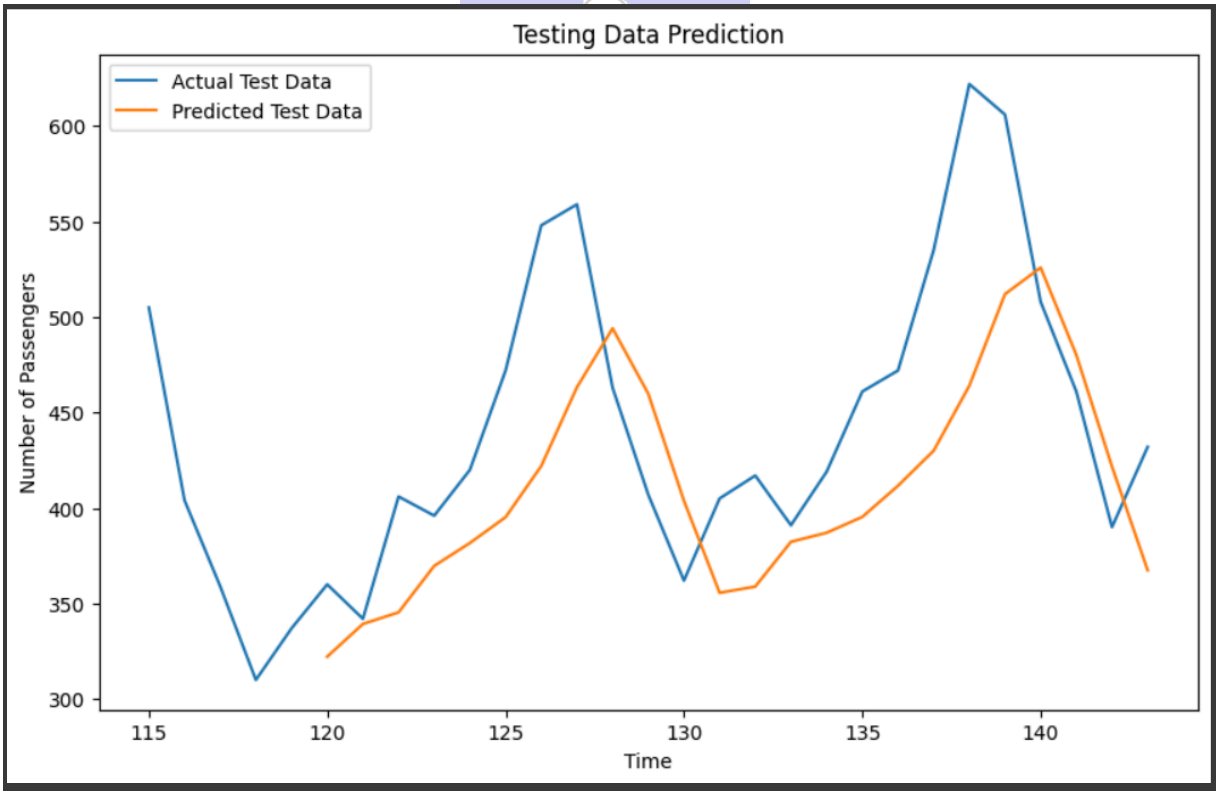
```
train_predictions = model.predict(X_train)

test_predictions = model.predict(X_test)
```

```
train_predictions = scaler.inverse_transform(train_predictions)

test_predictions = scaler.inverse_transform(test_predictions)
```

```
plt.figure(figsize=(10, 6))

plt.plot(data.index[:train_size], data['passengers'][:train_size],
label='Actual Train Data')

plt.plot(data.index[look_back:train_size], train_predictions,
label='Predicted Train Data')

plt.title('Training Data Prediction')

plt.xlabel('Time')

plt.ylabel('Number of Passengers')

plt.legend()

plt.show()
```



```
plt.figure(figsize=(10, 6))

plt.plot(data.index[train_size:], data['passengers'][train_size:],
label='Actual Test Data')

plt.plot(data.index[train_size+look_back:], test_predictions,
label='Predicted Test Data')

plt.title('Testing Data Prediction')

plt.xlabel('Time')

plt.ylabel('Number of Passengers')

plt.legend()

plt.show()
```



Post Lab Question- Answers (If Any):

1. How does LSTM solve vanishing gradient challenge?

Ans:

- **LSTMs (Long Short-Term Memory networks)** solve the vanishing gradient problem through the use of a more sophisticated architecture compared to traditional recurrent neural networks (RNNs).

- The vanishing gradient problem occurs when gradients become increasingly smaller as they propagate backward through many time steps in RNNs, making it difficult for the model to learn long-term dependencies.
- LSTMs address this issue with the help of specialized units called memory cells, which contain various components such as input, forget, and output gates.
- The key to addressing the vanishing gradient problem lies in the LSTM's gating mechanisms. These gates regulate the flow of information within the LSTM cell, allowing it to selectively retain or forget information over time.
- The input gate controls how much new information is stored in the memory cell, the forget gate determines which information should be discarded from the memory cell, and the output gate regulates the information flow from the memory cell to the output.
- By carefully controlling the flow of gradients through these gating mechanisms, LSTMs can effectively learn and retain important information over long sequences, mitigating the vanishing gradient problem and enabling them to capture long-term dependencies in sequential data.

2. What are the practical applications of LSTM?

Ans: LSTMs have a wide range of practical applications across various domains due to their ability to model sequential data and capture long-term dependencies. Some common applications of LSTMs include:

a. Natural Language Processing (NLP): LSTMs are widely used in tasks such as language modeling, machine translation, sentiment analysis, and text generation. They excel in processing and understanding sequential data like sentences or documents.

b. Speech Recognition: LSTMs are utilized in automatic speech recognition systems to convert spoken language into text. They can effectively model the temporal dependencies present in speech signals, making them suitable for this task.

c. Time Series Forecasting: LSTMs are well-suited for time series forecasting tasks, such as predicting stock prices, weather patterns, energy consumption, and demand forecasting. They can capture complex patterns and dependencies in the sequential data, making them valuable for predictive modeling.

d. Gesture Recognition: LSTMs can be applied to gesture recognition tasks, where sequential data from sensors (such as accelerometers or gyroscopes) are used to recognize hand gestures or body movements. This application finds use in various fields like human-computer interaction and virtual reality.

e. Healthcare: LSTMs are utilized in healthcare applications such as predicting patient outcomes, disease diagnosis, and monitoring physiological signals like electrocardiograms (ECG) or electroencephalograms (EEG). They can analyze sequential medical data and assist in making clinical decisions.

CO4: Underhand the essentials of Recurrent and Recursive Nets.

Conclusion: Thus we successfully implemented LSTM for airline passenger dataset.