**Experiment No.: 8 Title: LSTM**

**Name: Aarya Tiwari Batch: TY DL-4 Roll No.: 16010421119 Experiment No.: 8**

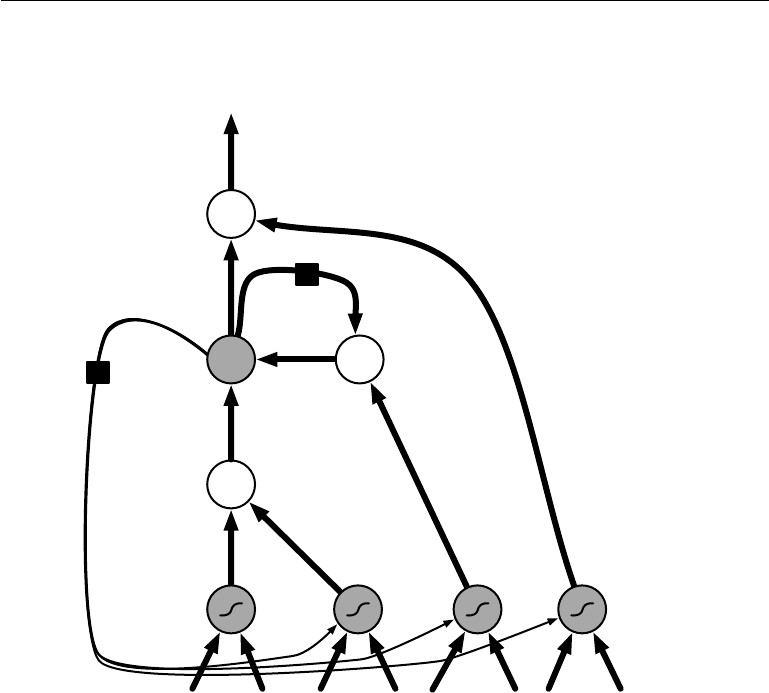
**Aim:** To implement LSTM network

**Resources needed: Python/Matlab**

**Theory:**

The most effective sequence models used in practical applications are called gated RNNs. These include the long short-term memory and networks based on the gated recurrent unit. Long Short Term Memory Networks is an advanced RNN, a sequential network, that allows information to persist. It is capable of handling the vanishing gradient problem faced by RNN. A recurrent neural network is also known as RNN is used for persistent memory.

Let’s say while watching a video you remember the previous scene or while reading a book you know what happened in the earlier chapter. Similarly, RNNs work, they remember the previous information and use it for processing the current input. The shortcoming of RNN is, they cannot remember Long term dependencies due to vanishing gradient. LSTMs are explicitly designed to avoid long-term dependency problems.



output

Self-loop

Forget

gate

state

Input

gate

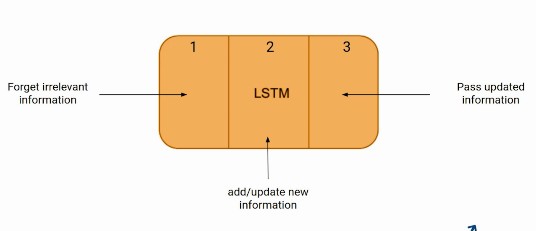
Output gate

input

Fig-1: Block diagram of the LSTM recurrent network “cell.”

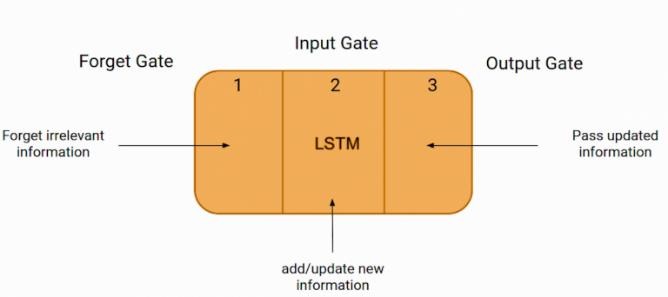
The clever idea of introducing self-loops to produce paths where the gradient can flow for long durations is a core contribution of the initial long short-term memory (LSTM) model (Hochreiter and Schmidhuber, 1997).

At a high-level LSTM works very much like an RNN cell. Here is the internal functioning of the LSTM network. The LSTM consists of three parts, as shown in the image below and each part performs an individual function.



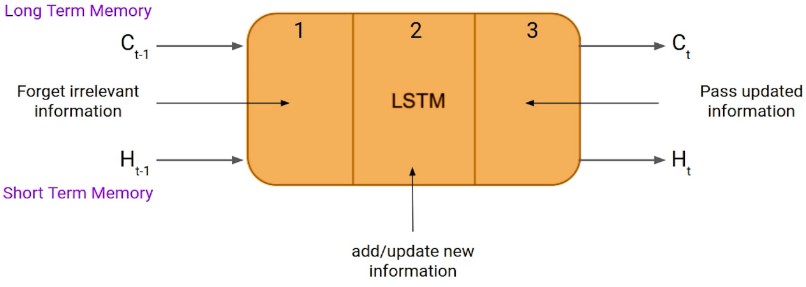
The first part chooses whether the information coming from the previous timestamp is to be remembered or is irrelevant and can be forgotten. In the second part, the cell tries to learn new information from the input to this cell. At last, in the third part, the cell passes the updated information from the current timestamp to the next timestamp.

These three parts of an LSTM cell are known as gates. The first part is called Forget gate, the second part is known as the Input gate and the last one is the Output gate.

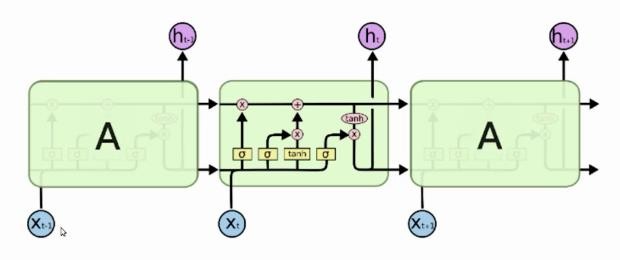


LSTM gates- Just like a simple RNN, an LSTM also has a hidden state where H(t-1) represents the hidden state of the previous timestamp and Ht is the hidden state of the current timestamp. In addition to that LSTM also have a cell state represented by C(t-1) and C(t) for previous and current timestamp respectively.

Here the hidden state is known as Short term memory and the cell state is known as Long term memory. Refer to the following image.



This is the More intuitive diagram of the LSTM network.



The LSTM has been found extremely successful in many applications, such as unconstrained handwriting recognition (Graves et al., 2009), speech recognition (Graves et al., 2013; Graves and Jaitly, 2014), handwriting generation (Graves, 2013), machine translation (Sutskever et al., 2014), image captioning (Kiros et al., 2014b; Vinyals et al., 2014b; Xu et al., 2015) and parsing (Vinyals et al., 2014a).

**Activity:**

* Load any time series dataset.
* Pre-process and visualize the dataset.
* Form the Training and Testing Data.
* Develop and train LSTM model.
* Plot the predictions for training and testing data.
* Comment on the output.

**Program:**

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)

# 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')

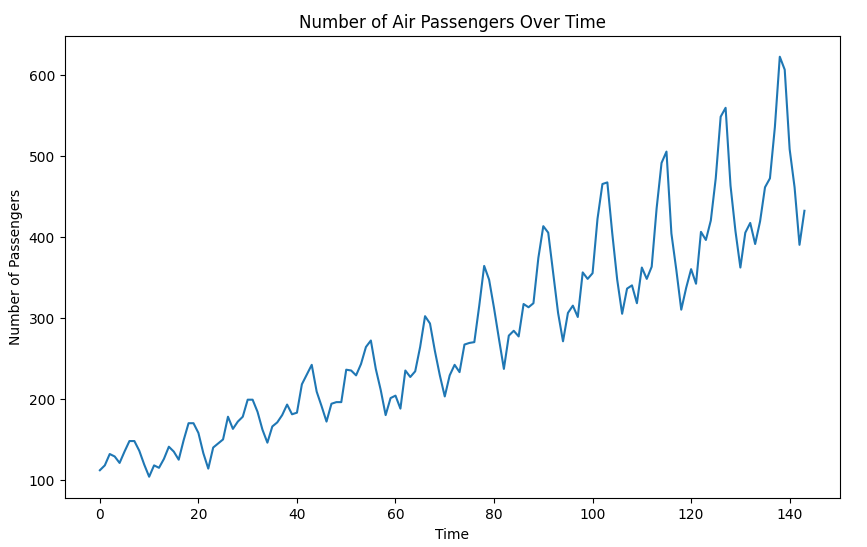
plt.xlabel('Time')

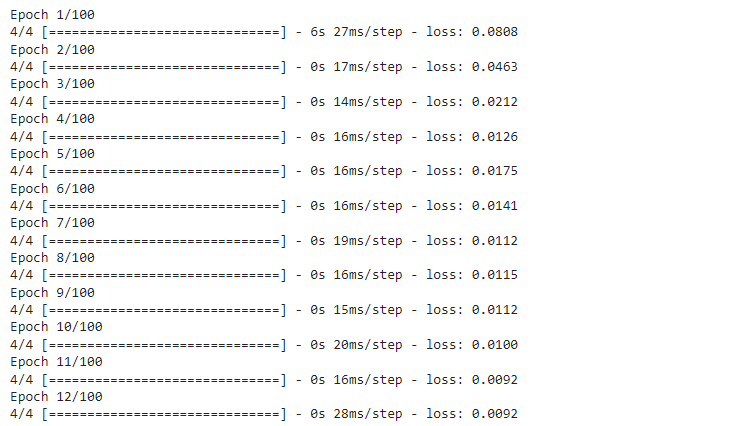
plt.ylabel('Number of Passengers')

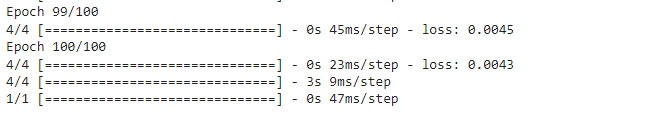
plt.legend()

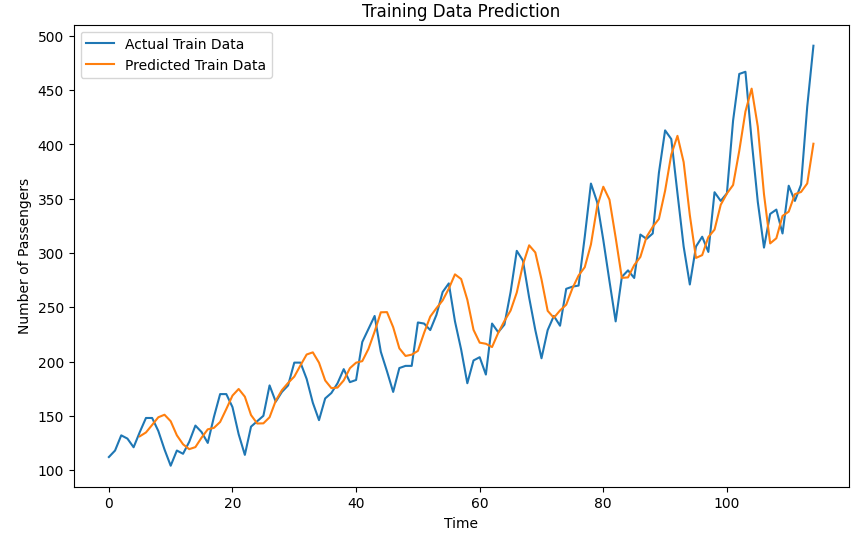
plt.show()

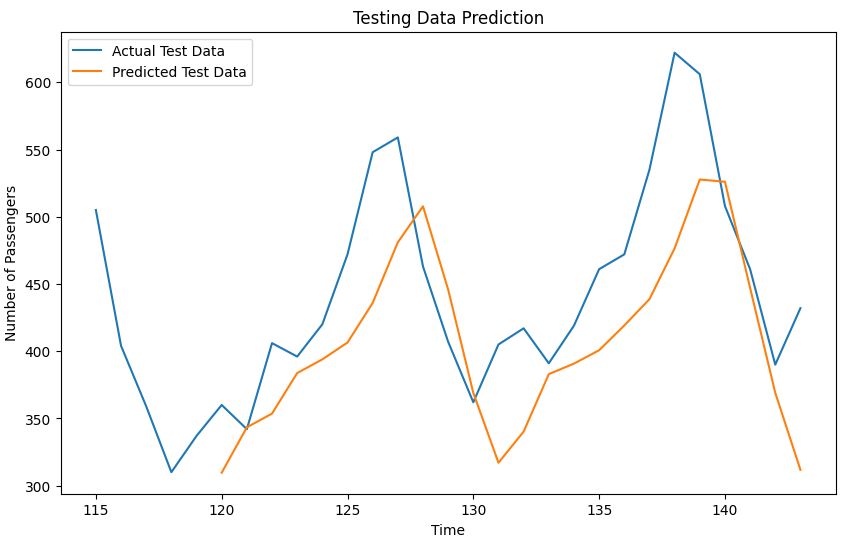
**Output:**

****

****

****

****

****

**Post Lab Question:**

1. **How does LSTM solve vanishing gradient challenge?**

LSTM (Long Short-Term Memory) networks solve the vanishing gradient problem by introducing a special architecture that allows them to selectively remember or forget information over time. The vanishing gradient problem occurs when the gradients during backpropagation become exponentially small as they propagate backward through many layers of a neural network, leading to slow or no learning in earlier layers. In traditional recurrent neural networks (RNNs), this problem arises because the gradients depend on the repeated multiplication of many small weights in the network, which can cause the gradients to vanish or explode over time. LSTMs solve this problem by using a gated cell state that can selectively remember or forget information over time. The cell state acts as a "highway" that information can flow through without being modified, allowing the gradients to propagate through the network more easily. The gate controls the information flow by deciding which information to keep and which information to discard. The three gates used in LSTM are the input gate, forget gate, and output gate. The input gate decides which information to store in the cell state, the forget gate decides which information to remove from the cell state, and the output gate decides which information to output from the cell state. The gates are controlled by sigmoid and tanh activation functions, allowing the network to learn when to add or remove information from the cell state. By selectively remembering or forgetting information over time, LSTMs can effectively address the vanishing gradient problem and learn long-term dependencies in the data.

1. **What are the practical applications of LSTM?**

LSTM (Long Short-Term Memory) networks have several practical applications in various fields, including:

1. Natural Language Processing (NLP): LSTMs are commonly used in NLP tasks such as language translation, sentiment analysis, and speech recognition. They are effective in processing variable-length sequences of text and can capture the context of words and phrases in a sentence.

2. Time Series Forecasting: LSTMs are widely used for time series forecasting tasks such as stock price prediction, weather forecasting, and energy demand prediction. They can learn long-term dependencies in the time series data and capture the underlying trends and seasonal patterns.

3. Image and Video Analysis: LSTMs can also be used in image and video analysis tasks such as object recognition, video classification, and action recognition. They can process variable-length sequences of images or video frames and capture the temporal dependencies between them.

4. Speech Recognition: LSTMs have been used for speech recognition tasks such as speech to-text conversion and speaker identification. They can handle variable-length audio sequences and capture the temporal dependencies between speech features.

5. Anomaly Detection: LSTMs can be used for anomaly detection in various applications such as fraud detection, intrusion detection, and equipment failure prediction. They can learn the normal patterns in the data and detect anomalies that deviate from the normal behaviour.

**CO4: Understand the essentials of Recurrent and Recursive Nets**

**Conclusion:**

We learnt about LSTM and implemented it in google colab

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of faculty in-charge with date**

**References:**

**Books/ Journals/ Websites:**

1. Josh Patterson and Adam Gibson, “Deep Learning A Practitioner’s Approach”, O’Reilly Media 2017
2. **http**[**s://www**](http://www.ibm.com/cloud/learn/recurrent-neural-networks)**.ibm**[**.com/cloud/learn/recurrent-neural-networks**](http://www.ibm.com/cloud/learn/recurrent-neural-networks)
3. [**https://searchenterpriseai.techtarget.com/definition/recurrent-neural-networks**](https://searchenterpriseai.techtarget.com/definition/recurrent-neural-networks) **http**[**s://www**](http://www.sciencedirect.com/science/article/pii/B9780128161760000260)**.sc**[**ience**](http://www.sciencedirect.com/science/article/pii/B9780128161760000260)**d**[**irect.com/science/article/pii/B9780128161760000260**](http://www.sciencedirect.com/science/article/pii/B9780128161760000260)