S. B. JAIN INSTITUTE OF TECHNOLOGY, MANAGEMENT & RESEARCH, NAGPUR.

Practical No. 11

Aim: Perform and implement LSTM (Long Short-Term Memory) on IMDb dataset.

Name of Student : Shrutika Pradeep Bagdi

Roll No : CS22130

Semester/Year : VIIth Sem / IVth Year

Academic Session : 2025-2026 [ODD]

Date of Performance : _____

Date of Submission : _____

AIM: Perform and implement LSTM (Long Short-Term Memory) on IMDb dataset.

OBJECTIVE/EXPECTED LEARNING OUTCOME:

The objectives and expected learning outcome of this practical are:

- To understand the working principle of Long Short-Term Memory (LSTM) networks and their role in handling sequential data.
- To learn how to preprocess textual data for sentiment analysis using the IMDb dataset.
- To implement an LSTM-based model for text classification using deep learning frameworks such as TensorFlow or Keras.
- To analyze the performance of the LSTM model in predicting sentiment (positive or negative) from movie reviews.

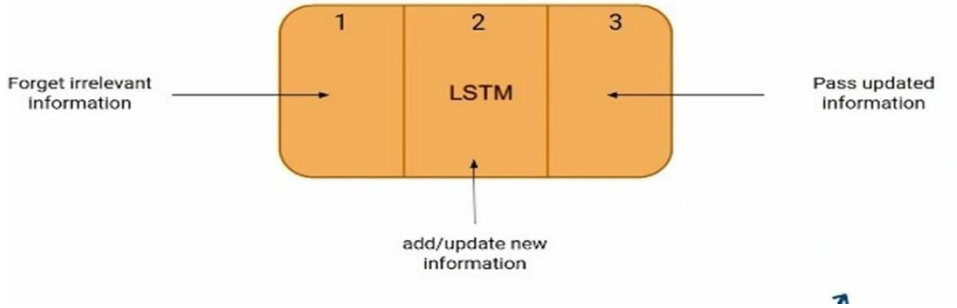
THEORY:

Long Short-Term Memory Networks or LSTM in deep learning, is a sequential neural network that allows information to persist. It is a special type of Recurrent Neural Network which is capable of handling the vanishing gradient problem faced by RNN. LSTM was designed by Hochreiter and Schmidhuber that resolves the problem caused by traditional rnns and machine learning algorithms

LSTM (Long Short-Term Memory) is a recurrent neural network (RNN) architecture widely used in Deep Learning. It excels at capturing long-term dependencies, making it ideal for sequence prediction tasks. Unlike traditional neural networks, LSTM incorporates feedback connections, allowing it to process entire sequences of data, not just individual data points. This makes it highly effective in understanding and predicting patterns in sequential data like time series, text, and speech. LSTM has become a powerful tool in artificial intelligence and deep learning, enabling breakthroughs in various fields by uncovering valuable insights from sequential data.

LSTM Architecture:

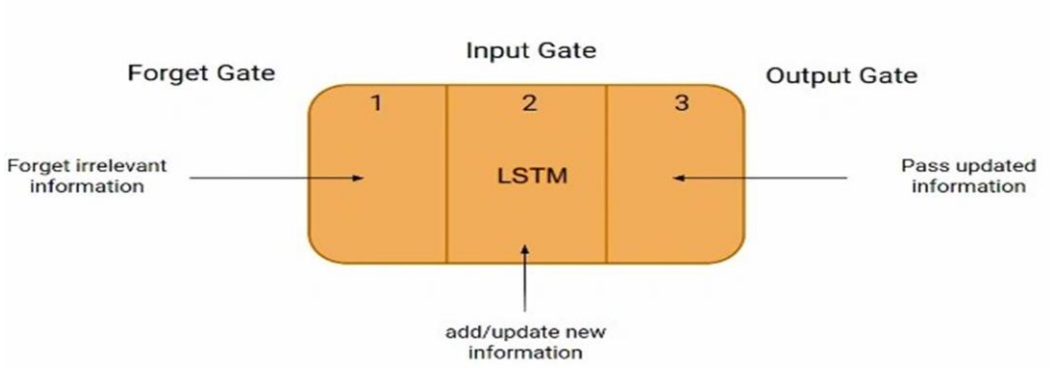
In the introduction to long short-term memory, we learned that it resolves the vanishing gradient problem faced by RNN, so now, in this section, we will see how it resolves this problem by learning the architecture of the LSTM. At a high level, LSTM works very much like an RNN cell. Here is the internal functioning of the **LSTM network**. The LSTM network architecture consists of three parts, as shown in the image below, and each part performs an individual function.



The Logic Behind LSTM:

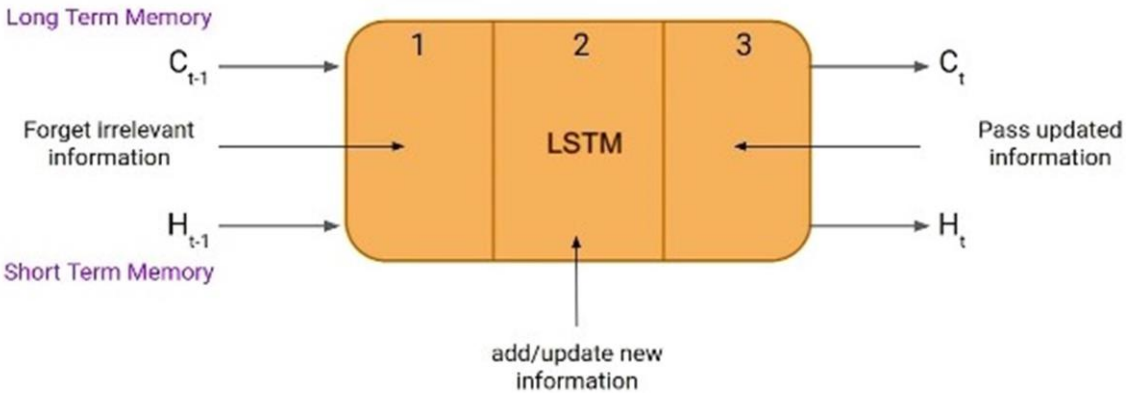
The first part chooses whether the information coming from the previous timestamp is to be remembered or is irrelevant and can be forgotten. In the second part, the cell tries to learn new information from the input to this cell. At last, in the third part, the cell passes the updated information from the current timestamp to the next timestamp. This one cycle of LSTM is considered a single-time step.

These three parts of an LSTM unit are known as gates. They control the flow of information in and out of the memory cell or lstm cell. The first gate is called Forget gate, the second gate is known as the Input gate, and the last one is the Output gate. An LSTM unit that consists of these three gates and a memory cell or lstm cell can be considered as a layer of neurons in traditional feedforward neural network, with each neuron having a hidden layer and a current state.



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 has a cell state represented by C(t-1) and C(t) for the previous and current timestamps, 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.



It is interesting to note that the cell state carries the information along with all the timestamps.

LSTM

Bob is a nice person. Dan on the other hand is evil.

Example of LTSM Working:

Let's take an example to understand how LSTM works. Here we have two sentences separated by a full stop. The first sentence is "Bob is a nice person," and the second sentence is "Dan, on the Other hand, is evil". It is very clear, in the first sentence, we are talking about Bob, and as soon as we encounter the full stop(.), we started talking about Dan.
As we move from the first sentence to the second sentence, our network should realize that we are no more talking about Bob. Now our subject is Dan. Here, the Forget gate of the network allows it to forget about it. Let's understand the roles played by these gates in LSTM architecture.

Forget Gate

In a cell of the LSTM neural network, the first step is to decide whether we should keep the information from the previous time step or forget it. Here is the equation for forget gate.

Forget Gate:

- $f_t = \sigma (x_t * U_f + H_{t-1} * W_f)$

Let's try to understand the equation, here

- Xt: input to the current timestamp.
- Uf: weight associated with the input
- Ht-1: The hidden state of the previous timestamp
- Wf: It is the weight matrix associated with the hidden state

Later, a sigmoid function is applied to it. That will make ft a number between 0 and 1. This ft is later multiplied with the cell state of the previous timestamp, as shown below.

$C_{t-1} * f_t = 0$ ...if $f_t = 0$ (forget everything)

$C_{t-1} * f_t = C_{t-1}$ ...if $f_t = 1$ (forget nothing)

Input Gate

Let's take another example.

"Bob knows swimming. He told me over the phone that he had served the navy for four long years." So, in both these sentences, we are talking about Bob. However, both give different kinds of information about Bob. In the first sentence, we get the information that he knows swimming. Whereas the second sentence tells, he uses the phone and served in the navy for four years.

Now just think about it, based on the context given in the first sentence, which information in the second sentence is critical? First, he used the phone to tell, or he served in the navy. In this context, it doesn't matter whether he used the phone or any other medium of communication to pass on the information. The fact that he was in the navy is important information, and this is something we want our model to remember for future computation. This is the task of the Input gate.
The input gate is used to quantify the importance of the new information carried by the input. Here is the equation of the input gate

Input Gate:

- $i_t = \sigma(x_t * U_i + H_{t-1} * W_i)$

Here,

- Xt: Input at the current timestamp t
- Ui: weight matrix of input
- Ht-1: A hidden state at the previous timestamp
- Wi: Weight matrix of input associated with hidden state

Again we have applied the sigmoid function over it. As a result, the value of I at timestamp t will be between 0 and 1.

New Information

- $N_t = \tanh(x_t * U_c + H_{t-1} * W_c)$ (new information)

Now the new information that needed to be passed to the cell state is a function of a hidden state at the previous timestamp t-1 and input x at timestamp t. The activation function here is tanh. Due to the tanh function, the value of new information will be between -1 and 1. If the value of Nt is negative, the information is subtracted from the cell state, and if the value is positive, the information is added to the cell state at the current timestamp.

However, the Nt won't be added directly to the cell state. Here comes the updated equation:

$$C_t = f_t * C_{t-1} + i_t * N_t \text{ (updating cell state)}$$

Here, Ct-1 is the cell state at the current timestamp, and the others are the values we have calculated previously.

Output Gate

Now consider this sentence.

"Bob single-handedly fought the enemy and died for his country. For his contributions, brave_____."

During this task, we have to complete the second sentence. Now, the minute we see the word brave, we know that we are talking about a person. In the sentence, only Bob is brave, we can not say the enemy is brave, or the country is brave. So based on the current expectation, we have to give a relevant word to fill in the blank. That word is our output, and this is the function of our Output gate. Here is the equation of the Output gate, which is pretty similar to the two previous gates.

Output Gate:

- $o_t = \sigma(x_t * U_o + H_{t-1} * W_o)$

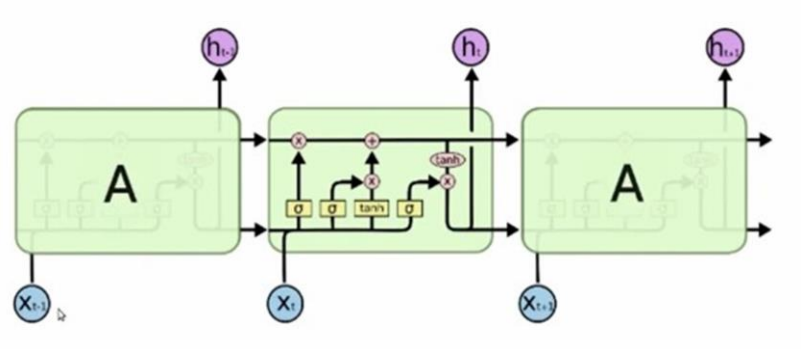
Its value will also lie between 0 and 1 because of this sigmoid function. Now to calculate the current hidden state, we will use Ot and tanh of the updated cell state. As shown below.

$$H_t = o_t * \tanh(C_t)$$

It turns out that the hidden state is a function of Long term memory (Ct) and the current output. If you need to take the output of the current timestamp, just apply the SoftMax activation on hidden state Ht.

$$\text{Output} = \text{Softmax}(H_t)$$

Here the token with the maximum score in the output is the prediction.
This is the More intuitive diagram of the LSTM network.



Advantages:

1. Handles long-term dependencies effectively compared to simple RNNs.
2. Reduces the problem of vanishing and exploding gradients.
3. Suitable for sequential data like text, audio, and time series.
4. Can remember relevant information for long durations.
5. Provides better accuracy in natural language processing tasks such as sentiment analysis.

Disadvantages:

1. Computationally more expensive than simple RNNs due to complex architecture.
2. Requires large datasets and high processing power for training.
3. Training time is longer because of multiple gate mechanisms.
4. Difficult to interpret the internal workings (less explainable).
5. May still struggle with extremely long sequences compared to Transformers.

Applications:

1. **Sentiment Analysis:** Classifying text as positive or negative (e.g., IMDb movie reviews).
2. **Speech Recognition:** Understanding and transcribing spoken language.
3. **Text Generation:** Generating text sequences such as poetry, code, or chat responses.
4. **Machine Translation:** Translating text from one language to another.
5. **Stock Price Prediction:** Forecasting time-series data based on historical patterns.
6. **Music Composition:** Generating new melodies based on learned patterns.

PROGRAM CODE:

```
# Step 1: Import libraries
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models, datasets, preprocessing
# Step 2: Load IMDb dataset
# Keep only top 10,000 most frequent words
num_words = 10000
maxlen = 200 # Cut off reviews longer than 200 words
(x_train, y_train), (x_test, y_test)=datasets.imdb.load_data (num_words=num_words)
# Step 3: Pad sequences (make all reviews same length)
x_train = preprocessing.sequence.pad_sequences (x_train, maxlen=maxlen)
x_test =  preprocessing.sequence.pad_sequences (x_test, maxlen=maxlen)
print("Training data shape:", x_train.shape)
print("Test data shape:", x_test.shape)
model = models. Sequential([
  layers.Embedding (input_dim=num_words, output_dim=128, input_length=maxlen),
  layers.LSTM(128, dropout=0.2, recurrent_dropout=0.2),
  layers.Dense (1, activation='sigmoid')
])
# Step 5: Compile model
model.compile(loss='binary_crossentropy',
    optimizer='adam',
    metrics=['accuracy'])
model.summary()
# Step 6: Train model
history = model.fit(x_train, y_train,
                batch_size=64,
                epochs=15,
                validation_split=0.2,
                verbose = 1)
#Step 7: Evaluate model
```

```
loss, accuracy = model.evaluate(x_test, y_test, verbose=1)
print("Test loss:", loss)
print("Test accuracy:", accuracy)
```

OUTPUT (SCREENSHOT):

Step 1: Import libraries

```
[ ]
#Shrutika Pradeep Bagdi_CS22130
# Step 1: Import libraries
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models, datasets, preprocessing
```

Step 2: Load IMDb dataset

```
[ ]
#Shrutika Pradeep Bagdi_CS22130
# Step 2: Load IMDb dataset
# Keep only top 10,000 most frequent words
num_words = 10000
maxlen = 200 # Cut off reviews longer than 200 words
(x_train, y_train), (x_test, y_test)=datasets.imdb.load_data (num_words=num_words)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

Step 3: Pad sequences (make all reviews same length)

```
[ ]
#Shrutika Pradeep Bagdi_CS22130
# Step 3: Pad sequences (make all reviews same length)
x_train = preprocessing.sequence.pad_sequences (x_train, maxlen=maxlen)
x_test =  preprocessing.sequence.pad_sequences (x_test, maxlen=maxlen)
print("Training data shape:", x_train.shape)
print("Test data shape:", x_test.shape)
```

```
Training data shape: (25000, 200)
Test data shape: (25000, 200)
```

Step 4: Build LSTM Model

```
[ ]
#Shrutika Pradeep Bagdi_CS22130
model = models. Sequential([
  layers.Embedding (input_dim=num_words, output_dim=128, input_length=maxlen),
  layers.LSTM(128, dropout=0.2, recurrent_dropout=0.2),
  layers.Dense (1, activation='sigmoid')
])
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/embedding.py:97: UserWarning:
  warnings.warn(
```

Step 5: Compile model

```
[ ]
#Shrutika Pradeep Bagdi_CS22130
# Step 5: Compile model
model.compile(loss='binary_crossentropy',
    optimizer='adam',
    metrics=['accuracy'])
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	?	0 (unbuilt)
lstm (LSTM)	?	0 (unbuilt)
dense (Dense)	?	0 (unbuilt)

Total params: 0 (0.00 B)
Trainable params: 0 (0.00 B)
Non-trainable params: 0 (0.00 B)

Step 6: Train model

```
#Shrutika Pradeep Bagdi_CS22130
# Step 6: Train model
history = model.fit(x_train, y_train,
                    batch_size=64,
                    epochs = 5,
                    validation_split=0.2,
                    verbose = 1)
```

```
Epoch 1/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 210s 658ms/step - accuracy: 0.6890 - loss: 0.5663 - val_accuracy: 0.8018 - val_loss: 0.4347
Epoch 2/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 254s 634ms/step - accuracy: 0.8567 - loss: 0.3438 - val_accuracy: 0.8508 - val_loss: 0.3552
Epoch 3/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 200s 628ms/step - accuracy: 0.8940 - loss: 0.2771 - val_accuracy: 0.8222 - val_loss: 0.3941
Epoch 4/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 197s 629ms/step - accuracy: 0.8969 - loss: 0.2708 - val_accuracy: 0.8612 - val_loss: 0.3564
Epoch 5/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 200s 640ms/step - accuracy: 0.9230 - loss: 0.2066 - val_accuracy: 0.8504 - val_loss: 0.3939
```

Step 7: Evaluate model

```
[ ]
#Shrutika Pradeep Bagdi_CS22130
#Step 7: Evaluate model
loss, accuracy = model.evaluate(x_test, y_test, verbose=1)
print("Test loss:", loss)
print("Test accuracy:", accuracy)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 66s 84ms/step - accuracy: 0.8504 - loss: 0.3997
Test loss: 0.4040710926055908
Test accuracy: 0.8499199748039246
```

CONCLUSION:

_ _

_ _

_ _

_ _

DISCUSSION AND VIVA VOCE:

1. Why was LSTM developed?
2. What problem does LSTM solve that traditional RNNs face?
3. What is the main difference between RNN and LSTM?
4. What are gates in LSTM? Why are they important?
5. Explain the function of the forget gate in LSTM.
6. What does the input gate do in LSTM?
7. What is the role of the output gate in LSTM?
8. How does LSTM prevent the vanishing gradient problem?
9. How does an LSTM differ from a GRU (Gated Recurrent Unit)?
10. How can overfitting be prevented in an LSTM model?
11. What are the advantages of using bidirectional LSTM?
12. How do you evaluate the performance of an LSTM model?

REFERENCE:

- https://www.analyticsvidhya.com/blog/2021/03/introduction-to-long-short-term-memory-lstm/
- https://www.mathworks.com/discovery/lstm.html
- https://d2l.ai/chapter_recurrent-modern/lstm.html