Name : Ajay Shitkar
Roll No : 58
Experiment No. 7
Design and implement LSTM model for sentiment analysis.
Date of Performance: 28/03/24
Date of Submission: 04/04/24

Aim: To design and implement LSTM model for handwriting recognition, speech recognition, machine translation, speech activity detection, robot control, video games, time series forecasting etc.
Objective: To design deep learning models for supervised, unsupervised and sequence learning

Theory:

Long Short-Term Memory is an improved version of recurrent neural network designed by Hochreiter & Schmid Huber. **LSTM** is well-suited for sequence prediction tasks and excels in capturing long-term dependencies. Its applications extend to tasks involving time series and sequences. LSTM's strength lies in its ability to grasp the order dependence crucial for solving intricate problems, such as machine translation and speech recognition. The article provides an in-depth introduction to LSTM, covering the LSTM model, architecture, working principles, and the critical role they play in various applications.
A traditional RNN has a single hidden state that is passed through time, which can make it difficult for the network to learn long-term dependencies. LSTMs address this problem by introducing a memory cell, which is a container that can hold information for an extended period. LSTM networks are capable of learning long-term dependencies in sequential data, which makes them well-suited for tasks such as language translation, speech recognition, and time series forecasting. LSTMs can also be used in combination with other neural network architectures, such as Convolutional Neural Networks (CNNs) for image and video analysis.
The memory cell is controlled by three gates: the input gate, the forget gate, and the output gate. These gates decide what information to add to, remove from, and output from the memory cell. The input gate controls what information is added to the memory cell. The forget gate controls what information is removed from the memory cell. And the output gate controls what information is output from the memory cell. This allows LSTM networks to selectively retain or discard information as it flows through the network, which allows them to learn long-term dependencies.
Bidirectional LSTM
Bidirectional LSTM (Bi LSTM/ BLSTM) is recurrent neural network (RNN) that is able to process sequential data in both forward and backward directions. This allows Bi LSTM to learn longer-range dependencies in sequential data than traditional LSTMs, which can only process sequential data in one direction.
Sentiment Analysis: the process of computationally identifying and categorizing opinions expressed in a piece of text, especially in order to determine whether the writer's attitude towards a particular topic, product, etc. is positive, negative, or neutral.

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import nltk
nltk.download('punkt')
from nltk.tokenize import word_tokenize
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Embedding,LSTM, Dense,Dropout
from sklearn.preprocessing import LabelEncoder
import csv
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

```
imdb = pd.read_csv('/content/IMDB Dataset.csv',engine="python")
imdb.head()
```

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. <br /><br />The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive

Next steps: Generate code with imdb View recommended plots

```
imdb.sentiment.value_counts()
```

```
sentiment
positive    25000
negative    25000
Name: count, dtype: int64
```

```
text=imdb['review'][10]
print(text)
print("---------------------")
print(word_tokenize(text))
```

```
or /><br />At first it was very odd and pretty funny but as the movie progressed I didn't find the jokes or oddness funr

iess', 'of', 'everything', 'rather', 'than', 'actual', 'punchlines.', '<', 'br', '/', '>', '<', 'br', '/', '>', 'At', 'f
```

```
corpus=[]
for text in imdb['review']:
  words=[word.lower() for word in word_tokenize(text)]
  corpus.append(words)
```

```
num_words=len(corpus)
print(num_words)
```

```
50000
```

```
imdb.shape
```

```
(50000, 2)
```

```
train_size=int(imdb.shape[0]*0.8)
X_train=imdb.review[:train_size]
y_train=imdb.sentiment[:train_size]

X_test=imdb.review[train_size:]
y_test=imdb.sentiment[train_size:]
```

```
tokenizer=Tokenizer(num_words)
tokenizer.fit_on_texts(X_train)
X_train=tokenizer.texts_to_sequences(X_train)
X_train=pad_sequences(X_train,maxlen=128, truncating='post',padding='post')
```

```
X_train[0],len(X_train[0])
```

```
(array([   27,     4,     1,    80,  2102,    45,  1073,    12,   100,
          147,    39,   316,  2968,   409,   459,    26,  3173,    33,
           23,   200,    14,    11,     6,   614,    48,   606,    16,
           68,     7,     7,     1,    87,   148,    12,  3256,    68,
           41,  2968,    13,    92,  5626,     2, 16202,   134,     4,
          569,    60,   271,     8,   200,    36,     1,   673,   139,
         1712,    68,    11,     6,    21,     3,   118,    15,     1,
         7870,  2257,    38, 11540,    11,   118,  2495,    54,  5662,
           16,  5182,     5,  1438,   377,    38,   569,    92,     6,
         3730,     8,     1,   360,   353,     4,     1,   673,     7,
            7,     9,     6,   431,  2968,    14,    12,     6,     1,
        11736,   356,     5,     1, 14689,  6526,  2594,  1087,     9,
         2661,  1432,    20, 22583,   534,    32,  4795,  2451,     4,
            1,  1193,   117,    29,     1,  6893,    25,  2874, 12191,
            2,   392], dtype=int32),
 128)
```

```
X_test=tokenizer.texts_to_sequences(X_test)
X_test=pad_sequences(X_test,maxlen=128,truncating='post',padding='post')
```

```
X_test[0],len(X_test[0])
```

```
(array([   87,   122,    10,   180,     5,   132,    12,    10,  7131,
         3717,    20,     1,  1001,  2285,     2,    10,   255,     1,
           17,  2431,    10,  1311,     5,   103,     1,   222,  6349,
            4,     3,    19,    11,    17,   974,     3,   351,     5,
          215,  1011,   415,     9,    13,   215,  1380,    56,   235,
          402,   300,     4,   316,    23,   257,    19,   961,    12,
        22250,    12,    33,    66,    61,   212,    53,    16,    11,
          113,    13,   497,     2,     1,   102,    70,  5358,    15,
            1,    88,   172,     1,   473,   824,     8,     1,    64,
           67,    54,    49,  2406,    30,    29,    33,    90,    40,
        35787,    83,    46,   438,     4,     3,    74,   220,     2,
           10,   115,    21,    63,    12,    30,    29,   268,    10,
         1059,   137,    10,    78,    21,   119,    28,    13,     1,
           88,   175,     5,   728,  3423,   108,     8,     1,    17,
           10,   115], dtype=int32),
 128)
```

```
print(X_train.shape,y_train.shape)
print(X_test.shape,y_test.shape)
```

```
(40000, 128) (40000,)
(10000, 128) (10000,)
```

```
le=LabelEncoder()
y_train=le.fit_transform(y_train)
y_test=le.transform(y_test)
```

```
model=Sequential()
model.add(Embedding(input_dim=num_words,output_dim=100,input_length=128,trainable=True))
model.add(LSTM(100,dropout=0.1,return_sequences=True))
model.add(LSTM(100,dropout=0.1))
model.add(Dense(1,activation='sigmoid'))
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
```

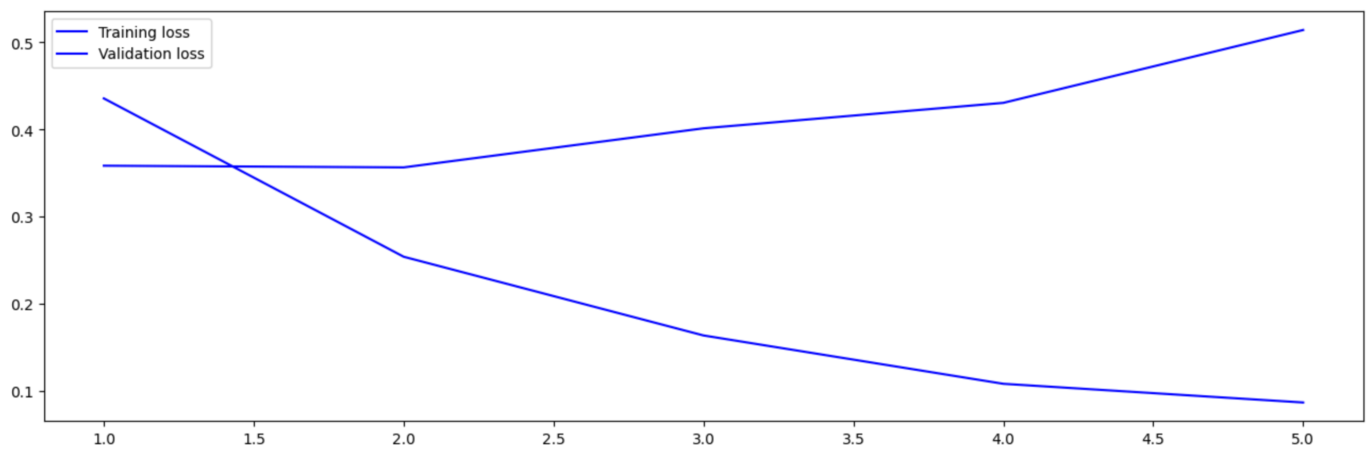
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 128, 100)          5000000

 lstm (LSTM)                 (None, 128, 100)          80400

 lstm_1 (LSTM)               (None, 100)               80400

 dense (Dense)               (None, 1)                 101

=================================================================
Total params: 5160901 (19.69 MB)
Trainable params: 5160901 (19.69 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
history=model.fit(X_train,y_train,epochs=5,batch_size=64,validation_data=(X_test,y_test))
```

```
Epoch 1/5
625/625 [==============================] - 370s 585ms/step - loss: 0.4357 - accuracy: 0.7961 - val_loss: 0.3585 - val_
Epoch 2/5
625/625 [==============================] - 380s 609ms/step - loss: 0.2540 - accuracy: 0.9050 - val_loss: 0.3566 - val_
Epoch 3/5
625/625 [==============================] - 373s 597ms/step - loss: 0.1637 - accuracy: 0.9417 - val_loss: 0.4014 - val_
Epoch 4/5
625/625 [==============================] - 370s 592ms/step - loss: 0.1082 - accuracy: 0.9636 - val_loss: 0.4306 - val_
Epoch 5/5
625/625 [==============================] - 377s 603ms/step - loss: 0.0868 - accuracy: 0.9714 - val_loss: 0.5143 - val_
```

```
plt.figure(figsize=(16,5))
epochs=range(1,len(history.history['accuracy'])+1)
plt.plot(epochs,history.history['loss'],'b',label='Training loss')
plt.plot(epochs,history.history['val_loss'],'b',label='Validation loss')
plt.legend()
plt.show()
```



```
#validation_sentence=['this movie was not good at all. it had some good parts like the acting was pretty good but story wa
validation_sentence=['i can watch the movie forever just because of beuty of cinematography']
validation_sentence_tokened=tokenizer.texts_to_sequences(validation_sentence)
val_sent_pad=pad_sequences(validation_sentence_tokened,maxlen=128,truncating='post',padding='post')
print(validation_sentence[0])
print(model.predict(val_sent_pad)[0])
```

```
i can watch the movie forever just because of beuty of cinematography
1/1 [==============================] - 1s 949ms/step
[0.72242147]
```

Conclusion: In this exploration of LSTM models across various applications including handwriting recognition, speech recognition, machine translation, speech activity detection, robot control, video games, and time series forecasting, it becomes evident that LSTMs exhibit remarkable versatility and efficacy. Their ability to capture long-term dependencies in sequential data makes them well-suited for tasks requiring context understanding and pattern recognition. Through appropriate model architecture design, hyperparameter tuning, and data preprocessing, LSTMs can achieve impressive performance across diverse domains. Their deployment in real-world scenarios underscores their significance in advancing technology across multiple fields, promising enhanced automation, efficiency, and user experience. As research continues to evolve, leveraging LSTMs will likely remain a cornerstone in tackling complex sequential data analysis challenges, driving innovation and transformation across a myriad of applications.