Date of Performance :

Date of Submission :

EXPERIMENT NUMBER: 7

Aim: Design and implement LSTM for Sentiment Analysis

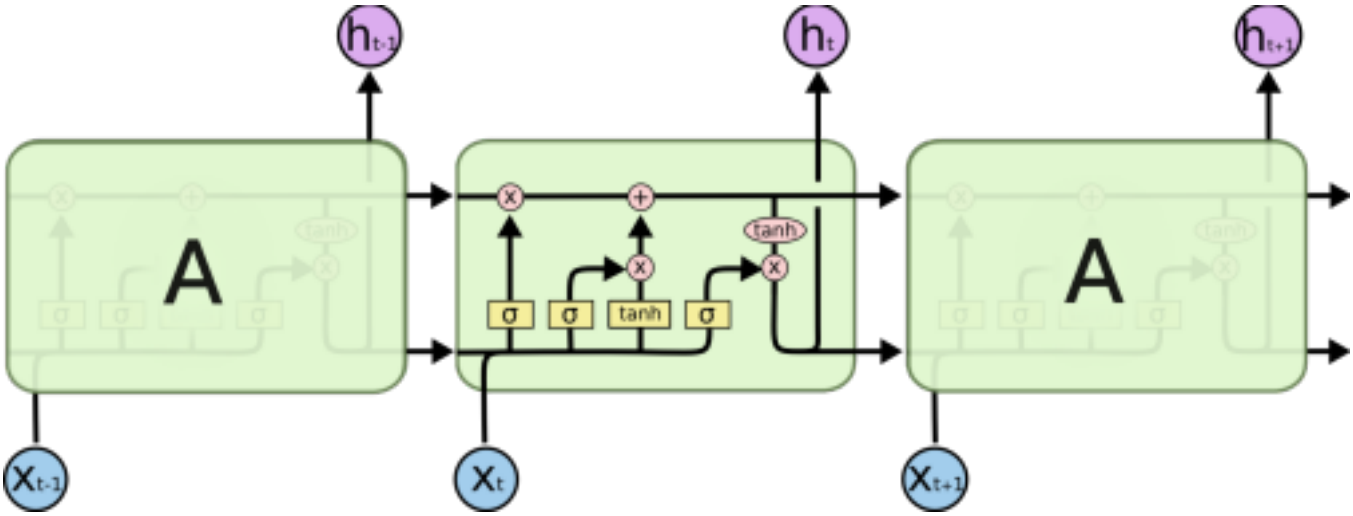
Objective:

Students will be designed appropriate DNN model for supervised, unsupervised and sequence learning applications.

Software Used : Python

Theory:

- Long Short Term Memory networks (LSTM)



LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn.LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.
LSTM has three gates:
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.

Input Gate: The input gate is used to quantify the importance of the new information carried by the input.

Output Gate: Its value will also lie between 0 and 1 because of this sigmoid function. If you need to

take the output of the current timestamp, just apply the SoftMax activation on hidden state Ht.

Algorithm:

1) download `IMDB Dataset from keras`
2) Split data in training and testing part.
3) Save image parameters to the constants that we will use later for data re-shaping and for model traning.
4) Normalized data
5) Add convolutional layer
6) Add max pooling layer
7) Add convolutional layer
8) Add max pooling layer
9) Add Flatten layer
10) Add dropout layer and fully connected layer
11) Train model and find accuracy from confusion matrix

Program:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings
# warnings.filterwarnings("ignore")
from google.colab import drive
drive.mount("/content/gdrive")
df = pd.read_csv("/content/gdrive/My Drive/Colab
Notebooks/IMDB Dataset.csv")
df.head(10)
print("Summary statistics of numerical features : \n",
df.describe())
print("==================================================
======================") print("\nTotal number of
reviews: ",len(df))
print("==================================================
======================") print("\nTotal number of
Sentiments: ", len(list(set(df['sentiment']))))
df['sentiment'] = np.where(df['sentiment'] ==
"positive", 1, 0) df
df = df.sample(frac=0.1, random_state=0) #uncomment to
```

```
use full set of data
# Drop missing values
df.dropna(inplace=True)
df
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =
train_test_split(df['review'], df['sentiment'], \
test_size=0.1,
random_state=0)
print('Load %d training examples and %d validation
examples. \n' %(X_train.shape[0],X_test.shape[0]))
print('Show a review in the training set : \n',
X_train.iloc[10])
X_train,y_train
def cleanText(raw_text, remove_stopwords=False,
stemming=False, split_text=False, \
):
'''
Convert a raw review to a cleaned review
'''
text = BeautifulSoup(raw_text,
'html.parser').get_text()
letters_only = re.sub("[^a-zA-Z]", " ", text)
words = letters_only.lower().split()
if remove_stopwords:
stops = set(stopwords.words("english"))
words = [w for w in words if not w in stops]
if stemming==True:
stemmer = SnowballStemmer('english')
words = [stemmer.stem(w) for w in words]
if split_text==True:
return (words)
return( " ".join(words))
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from nltk.stem import SnowballStemmer,
```

```
WordNetLemmatizer
from nltk import sent_tokenize, word_tokenize, pos_tag
from bs4 import BeautifulSoup
import logging
from wordcloud import WordCloud
from gensim.models import word2vec
from gensim.models import Word2Vec
from gensim.models.keyedvectors import KeyedVectors
X_train_cleaned = []
X_test_cleaned = []
for d in X_train:
X_train_cleaned.append(cleanText(d))
print('Show a cleaned review in the training set : \n',
X_train_cleaned[10])
for d in X_test:
X_test_cleaned.append(cleanText(d))
from sklearn.feature_extraction.text import
CountVectorizer,TfidfVectorizer from
sklearn.naive_bayes import BernoulliNB, MultinomialNB
countVect = CountVectorizer()
X_train_countVect =
countVect.fit_transform(X_train_cleaned) print("Number
of features : %d \n"
%len(countVect.get_feature_names_out())) #6378
print("Show some feature names : \n",
countVect.get_feature_names_out()[::1000])
# Train MultinomialNB classifier
mnb = MultinomialNB()
mnb.fit(X_train_countVect, y_train)
import pickle
pickle.dump(countVect,open('countVect_imdb.pkl','wb'))
from sklearn import metrics
from sklearn.metrics import
accuracy_score,roc_auc_score
def modelEvaluation(predictions):
'''
Print model evaluation to predicted result
'''
```

```
print ("\nAccuracy on validation set:
{:.4f}".format(accuracy_score(y_test, predictions)))
print("\nAUC score :
{:.4f}".format(roc_auc_score(y_test, predictions)))
print("\nClassification report : \n",
metrics.classification_report(y_test, predictions))
print("\nConfusion Matrix : \n",
metrics.confusion_matrix(y_test, predictions))
predictions =
mnb.predict(countVect.transform(X_test_cleaned))
modelEvaluation(predictions)
import pickle
pickle.dump(mnb,open('Naive_Bayes_model_imdb.pkl','wb')
)
from sklearn.linear_model import LogisticRegression
tfidf = TfidfVectorizer(min_df=5) #minimum document
frequency of 5 X_train_tfidf =
tfidf.fit_transform(X_train)
print("Number of features : %d \n"
%len(tfidf.get_feature_names_out())) #1722 print("Show
some feature names : \n",
tfidf.get_feature_names_out()[::1000])
# Logistic Regression
lr = LogisticRegression()
lr.fit(X_train_tfidf, y_train)
feature_names = np.array(tfidf.get_feature_names_out())
sorted_coef_index = lr.coef_[0].argsort()
print('\nTop 10 features with smallest coefficients
:\n{}\n'.format(feature_names[sorted_coef_index[:10]]))
print('Top 10 features with largest coefficients :
\n{}'.format(feature_names[sorted_coef_index[:-11:-1]])
)
predictions =
lr.predict(tfidf.transform(X_test_cleaned))
modelEvaluation(predictions)
from sklearn.model_selection import GridSearchCV
from sklearn import metrics
from sklearn.metrics import roc_auc_score,
```

```
accuracy_score from sklearn.pipeline import Pipeline
estimators = [("tfidf", TfidfVectorizer()), ("lr",
LogisticRegression())] model = Pipeline(estimators)
params = {"lr__C":[0.1, 1, 10],
"tfidf__min_df": [1, 3],
"tfidf__max_features": [1000, None],
"tfidf__ngram_range": [(1,1), (1,2)],
"tfidf__stop_words": [None, "english"]}
grid = GridSearchCV(estimator=model, param_grid=params,
scoring="accuracy", n_jobs=-1)
grid.fit(X_train_cleaned, y_train)
print("The best paramenter set is : \n",
grid.best_params_)
# Evaluate on the validaton set
predictions = grid.predict(X_test_cleaned)
modelEvaluation(predictions)
import nltk
nltk.download('punkt')
tokenizer =
nltk.data.load('tokenizers/punkt/english.pickle') def
parseSent(review, tokenizer, remove_stopwords=False):
raw_sentences = tokenizer.tokenize(review.strip())
sentences = []
for raw_sentence in raw_sentences:
if len(raw_sentence) > 0:
sentences.append(cleanText(raw_sentence,
remove_stopwords, split_text=True))
return sentences
# Parse each review in the training set into sentences
sentences = []
for review in X_train_cleaned:
sentences += parseSent(review,
tokenizer,remove_stopwords=False)
print('%d parsed sentence in the training set\n'
%len(sentences)) print('Show a parsed sentence in the
training set : \n', sentences[10]) from wordcloud
import WordCloud
from gensim.models import word2vec
```

```
from gensim.models.keyedvectors import KeyedVectors
num_features = 300 #embedding dimension
min_word_count = 10
num_workers = 4
context = 10
downsampling = 1e-3
print("Training Word2Vec model ...\n")
w2v = Word2Vec(sentences, workers=num_workers,
vector_size=num_features, min_count = min_word_count,\
window = context, sample = downsampling)
w2v.init_sims(replace=True)
w2v.save("w2v_300features_10minwordcounts_10context")
#save trained word2vec model
#print("Number of words in the vocabulary list : %d
\n"
%len(w2v.wv.index_to_word)) #4016
#print("Show first 10 words in the vocalbulary list
vocabulary list: \n", w2v.wv.index_to_word[0:10])
def makeFeatureVec(review, model, num_features):
'''
Transform a review to a feature vector by averaging
feature vectors of words appeared in that review and in
the volcabulary list created
'''
featureVec = np.zeros((num_features,),dtype="float32")
nwords = 0.
index2word_set = set(model.wv.index2word) #index2word
is the volcabulary list of the Word2Vec model
isZeroVec = True
for word in review:
if word in index2word_set:
nwords = nwords + 1.
featureVec = np.add(featureVec, model[word])
isZeroVec = False
if isZeroVec == False:
featureVec = np.divide(featureVec, nwords)
return featureVec
def getAvgFeatureVecs(reviews, model, num_features):
```

```
'''
Transform all reviews to feature vectors using
makeFeatureVec() '''
counter = 0
reviewFeatureVecs =
np.zeros((len(reviews),num_features),dtype="float32")
for review in reviews:
reviewFeatureVecs[counter] = makeFeatureVec(review,
model,num_features) counter = counter + 1
return reviewFeatureVecs
import nltk
nltk.download('stopwords')
X_train_cleaned = []
for review in X_train:
X_train_cleaned.append(cleanText(review,
remove_stopwords=True, split_text=True))
trainVector = getAvgFeatureVecs(X_train_cleaned, w2v,
num_features) print("Training set : %d feature vectors
with %d dimensions" %trainVector.shape)
# Get feature vectors for validation set
X_test_cleaned = []
for review in X_test:
X_test_cleaned.append(cleanText(review,
remove_stopwords=True, split_text=True))
testVector = getAvgFeatureVecs(X_test_cleaned, w2v,
num_features) print("Validation set : %d feature
vectors with %d dimensions" %testVector.shape)
```

LSTM

```
#Step 1 : Prepare X_train and X_test to 2D tensor.

#Step 2 : Train a simple LSTM (embeddign layer => LSTM layer =>
dense layer). #Step 3 : Compile and fit the model using log loss
function and ADAM optimizer. !pip install matplotlib-venn

!apt-get -qq install -y libfluidsynth1

!apt-get -qq install -y libarchive-dev && pip install -U libarchive
import libarchive
```

```
!apt-get -qq install -y graphviz && pip install pydot

import pydot

!pip install cartopy

import cartopy

from keras.preprocessing import sequence

from keras.utils import np_utils

from keras.models import Sequential

from keras.layers.core import Dense, Dropout, Activation, Lambda
##from keras.layers.embeddings import Embedding

from tensorflow.keras.layers import Embedding

from keras.layers import LSTM

from keras.layers import SimpleRNN

from keras.layers import GRU

#from keras.layers.recurrent import LSTM, SimpleRNN, GRU

from keras.preprocessing.text import Tokenizer

from collections import defaultdict

from keras.layers.convolutional import Convolution1D

from keras import backend as K

##from keras.layers.embeddings import Embedding

top_words = 40000

maxlen = 200

batch_size = 62

nb_classes = 4

nb_epoch = 6

from keras.utils import pad_sequences

# Vectorize X_train and X_test to 2D tensor
```

```
tokenizer = Tokenizer(nb_words=top_words) #only consider top 20000
words in the corpse

tokenizer.fit_on_texts(X_train)

# tokenizer.word_index #access word-to-index dictionary of trained
tokenizer

sequences_train = tokenizer.texts_to_sequences(X_train)

sequences_test = tokenizer.texts_to_sequences(X_test)

X_train_seq = pad_sequences(sequences_train, maxlen=maxlen) X_test_seq
= pad_sequences(sequences_test, maxlen=maxlen)

# one-hot encoding of y_train and y_test

y_train_seq = np_utils.to_categorical(y_train, nb_classes) y_test_seq
= np_utils.to_categorical(y_test, nb_classes)

print('X_train shape:', X_train_seq.shape)

print("=======================================")

print('X_test shape:', X_test_seq.shape)

print("=======================================")

print('y_train shape:', y_train_seq.shape)

print("=======================================")

print('y_test shape:', y_test_seq.shape)

print("=======================================")

model1 = Sequential()

model1.add(Embedding(top_words, 128))

model1.add(Dropout(0.2))

model1.add(LSTM(128, dropout=0.2, recurrent_dropout=0.2))
model1.add(Dense(nb_classes))

model1.add(Activation('softmax'))

model1.summary()
```

```
model1.compile(loss='binary_crossentropy',

optimizer='adam',

metrics=['accuracy'])

model1.fit(X_train_seq, y_train_seq, batch_size=batch_size,
epochs=nb_epoch, verbose=1)

# Model evluation

score = model1.evaluate(X_test_seq, y_test_seq, batch_size=batch_size)



print('Test loss : {:.4f}'.format(score[0]))

print('Test accuracy : {:.4f}'.format(score[1]))

len(X_train_seq),len(y_train_seq)

print("Size of weight matrix in the embedding layer : ", \

model1.layers[0].get_weights()[0].shape)

# get weight matrix of the hidden layer

print("Size of weight matrix in the hidden layer : ", \

model1.layers[0].get_weights()[0].shape)

# get weight matrix of the output layer

print("Size of weight matrix in the output layer : ", \

model1.layers[2].get_weights()[0].shape)

import pickle

pickle.dump(model1,open('model1.pkl','wb'))

v2swe = Word2Vec.load("w2v_300features_10minwordcounts_10context")

embedding_matrix = w2v.wv.vectors

print("Shape of embedding matrix : ", embedding_matrix.shape)
top_words = embedding_matrix.shape[0] #4016

maxlen = 300
```

```
batch_size = 62

nb_classes = 4

nb_epoch = 7

# Vectorize X_train and X_test to 2D tensor

tokenizer = Tokenizer(nb_words=top_words) #only consider top 20000
words in the corpse

tokenizer.fit_on_texts(X_train)

# tokenizer.word_index #access word-to-index dictionary of trained
tokenizer sequences_train = tokenizer.texts_to_sequences(X_train)



sequences_test = tokenizer.texts_to_sequences(X_test)

X_train_seq1 = pad_sequences(sequences_train, maxlen=maxlen)
X_test_seq1 = pad_sequences(sequences_test, maxlen=maxlen)

# one-hot encoding of y_train and y_test

y_train_seq1 = np_utils.to_categorical(y_train, nb_classes)
y_test_seq1 = np_utils.to_categorical(y_test, nb_classes)

print('X_train shape:', X_train_seq1.shape)

print("=========================================") print('X_test
shape:', X_test_seq1.shape)

print("=========================================") print('y_train
shape:', y_train_seq1.shape)

print("=========================================") print('y_test
shape:', y_test_seq1.shape)

print("=========================================")
len(X_train_seq1),len(y_train_seq1)

embedding_layer = Embedding(embedding_matrix.shape[0], #4016
embedding_matrix.shape[1], #300

weights=[embedding_matrix])
```

```
model2 = Sequential()

model2.add(embedding_layer)

model2.add(LSTM(128, dropout=0.2, recurrent_dropout=0.2))
model2.add(Dense(nb_classes))

model2.add(Activation('softmax'))

model2.summary()

model2.compile(loss='binary_crossentropy',

optimizer='adam',

metrics=['accuracy'])

model2.fit(X_train_seq1, y_train_seq1, batch_size=batch_size,
epochs=nb_epoch, verbose=1)

# Model evaluation

score = model2.evaluate(X_test_seq1, y_test_seq1,
batch_size=batch_size) print('Test loss : {:.4f}'.format(score[0]))

print('Test accuracy : {:.4f}'.format(score[1]))

print("Size of weight matrix in the embedding layer : ", \

model2.layers[0].get_weights()[0].shape)

print("Size of weight matrix in the hidden layer : ", \

model2.layers[1].get_weights()[0].shape)

print("Size of weight matrix in the output layer : ", \

model2.layers[2].get_weights()[0].shape)
```

Output:

```
4500 parsed sentence in the training set

Show a parsed sentence in the training set :
 ['the', 'crimson', 'rivers', 'is', 'one', 'of', 'the', 'most', 'over', 'directed', 'over', 'the', 'top', 'over', 'e
verything', 'mess', 'i', 've', 'ever', 'seen', 'come', 'out', 'of', 'france', 'there', 's', 'nothing', 'worse', 'tha
n', 'a', 'french', 'production', 'trying', 'to', 'out', 'do', 'films', 'made', 'in', 'hollywood', 'and', 'cr', 'is',
'a', 'perfect', 'example', 'of', 'such', 'a', 'wannabe', 'horror', 'action', 'buddy', 'flick', 'i', 'almost', 'stopp
ed', 'it', 'halfway', 'through', 'because', 'i', 'knew', 'it', 'wouldn', 't', 'amount', 'to', 'anything', 'but', 'fr
ench', 'guys', 'trying', 'to', 'show', 'off', 'the', 'film', 'starts', 'off', 'promisingly', 'like', 'some', 'sort',
'of', 'expansive', 'horror', 'film', 'but', 'it', 'quickly', 'shifts', 'genres', 'from', 'horror', 'to', 'action', '
to', 'x', 'files', 'type', 'to', 'buddy', 'flick', 'that', 'in', 'the', 'end', 'cr', 'is', 'all', 'of', 'it', 'and',
'also', 'none', 'of', 'it', 'it', 's', 'so', 'full', 'of', 'clich', 's', 'that', 'at', 'one', 'point', 'i', 'thought
', 'the', 'whole', 'thing', 'was', 'a', 'comedy', 'the', 'painful', 'dialogue', 'and', 'those', 'silent', 'pauses',
'with', 'fades', 'outs', 'and', 'fades', 'ins', 'just', 'at', 'the', 'right', 'expositionary', 'moments', 'made', 'm
e', 'groan', 'i', 'thought', 'only', 'films', 'made', 'in', 'hollywood', 'used', 'this', 'hackneyed', 'technique', '
the', 'chase', 'scene', 'with', 'vincent', 'cassel', 'running', 'after', 'the', 'killer', 'is', 'so', 'over', 'direc
```

```
Classification report :
              precision    recall  f1-score   support

           0       0.87      0.87      0.87       249
           1       0.87      0.88      0.87       251

    accuracy                           0.87       500
   macro avg       0.87      0.87      0.87       500
weighted avg       0.87      0.87      0.87       500


Confusion Matrix :
 [[216  33]
 [ 31 220]]
```

Conclusion/Outcome: LSTM implemented for Sentiment Analysis

Marks & Signature:

R1 (4 Marks)	R2 (4 Marks)	R3 (4 Marks)	R4 (3 Marks)	Total (15 Marks)	Signature