# NEURAL NETWORK DEEP LEARNING
# ASSIGNMENT 9
# 700748022
# YAMINI SARASWATHI BORRA

**GitHub:**
Repository URL for the source code:
https://github.com/YaminiSai786/CS5720-Neural-Networks-Deep-Learning---ICP

**Video Link:**
https://github.com/YaminiSai786/CS5720-Neural-Networks-Deep-Learning---ICP

**1. Save the model and use the saved model to predict on new text data (ex, “A lot of good things are happening. We are respected again throughout the world, and that's a great thing.@realDonaldTrump”)**

```
#ICP 9 - Neural Networks & Deep Learning
#Student Name : Yamini Saraswathi Borra
#Student ID   : 700748022
#Submission Date : March 28th 2024
import pandas as pd #Basic packages for creating dataframes and loading dataset
import numpy as np

import matplotlib.pyplot as plt #Package for visualization

import re #importing package for Regular expression operations

from sklearn.model_selection import train_test_split #Package for splitting the data

from sklearn.preprocessing import LabelEncoder #Package for conversion of categorical to Numerical

from keras.preprocessing.text import Tokenizer #Tokenization
from tensorflow.keras.preprocessing.sequence import pad_sequences #Add zeros or crop based on the length
from keras.models import Sequential #Sequential Neural Network
from keras.layers import Dense, Embedding, LSTM, SpatialDropout1D #For layers in Neural Network
from keras.utils.np_utils import to_categorical
```
[3]

```
df = pd.read_csv("C:\\Users\\M1097753\\Documents\\GITHUB\\Sentiment (3).csv")
```
[4]

```
import pandas as pd

# Load the dataset as a Pandas DataFrame
dataset = pd.read_csv(path_to_csv, header=0)

# Select only the necessary columns 'text' and 'sentiment'
mask = dataset.columns.isin(['text', 'sentiment'])
data = dataset.loc[:, mask]

# Keeping only the necessary columns
```
[27]

Importing the packages, reading the .csv file, loading the dataset as a Panda DataFrame, selecting only the necessary columns that are ‘text’ and ‘sentiment’.

```
data['text'] = data['text'].apply(lambda x: x.lower())
data['text'] = data['text'].apply((lambda x: re.sub('[^a-zA-z0-9\s]', '', x)))
```
[29]

```
<ipython-input-29-cee1da567eb8>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data['text'] = data['text'].apply(lambda x: x.lower())
<ipython-input-29-cee1da567eb8>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data['text'] = data['text'].apply((lambda x: re.sub('[^a-zA-z0-9\s]', '', x)))
```

```
for idx, row in data.iterrows():
    row[0] = row[0].replace('rt', ' ') #Removing Retweets
```
[30]

Using lambda function to convert to lowercase.

Using re.sub() regular expression to replace any non-alphanumeric characters and non-whitespace characters with an empty string.

Then, Removing the retweets.

```
max_fatures = 2000
tokenizer = Tokenizer(num_words=max_fatures, split=' ') #Maximum words is 2000 to tokenize sentence
tokenizer.fit_on_texts(data['text'].values)
X = tokenizer.texts_to_sequences(data['text'].values) #taking values to feature matrix
```
[31]

```
X = pad_sequences(X) #Padding the feature matrix

embed_dim = 128 #Dimension of the Embedded layer
lstm_out = 196 #Long short-term memory (LSTM) layer neurons
```
[32]

Total number of words to tokenize a sentence is limited to 2000 and takes values to feature matrices. Now padding the feature matrix with Dimension 128 and long short-term memory to 196.

```python
def createmodel():
    model = Sequential() #Sequential Neural Network
    model.add(Embedding(max_fatures, embed_dim,input_length = X.shape[1])) #input dimension 2000 Neurons, output dimension 128 Neurons
    model.add(LSTM(lstm_out, dropout=0.2, recurrent_dropout=0.2)) #Drop out 20%, 196 output Neurons, recurrent dropout 20%
    model.add(Dense(3,activation='softmax')) #3 output neurons[positive, Neutral, Negative], softmax as activation
    model.compile(loss = 'categorical_crossentropy', optimizer='adam',metrics = ['accuracy']) #Compiling the model
    return model
# print(model.summary())
```
[33]

the 'createmodel()' function creates a Sequential model with an Embedding layer to handle the input data, an LSTM layer to extract features from the input sequences, and a Dense layer with softmax activation to predict the sentiment class. The model is then compiled and returned.

```python
labelencoder = LabelEncoder() #Applying label Encoding on the label matrix
integer_encoded = labelencoder.fit_transform(data['sentiment']) #fitting the model
y = to_categorical(integer_encoded)
X_train, X_test, Y_train, Y_test = train_test_split(X,y, test_size = 0.33, random_state = 42) #67% training data, 33% test data split
```
[34]

the 'fit_transform()' method of the 'LabelEncoder' instance is to convert the categorical sentiment labels in the data DataFrame to numerical values. The resulting numerical values are stored in the integer_encoded variable. Then this splits the input data (X) and target data (y) into training and testing sets using the train_test_split function from the Scikit-learn model selection module. The testing set is set to be 33% of the total data (test_size=0.33) and the random seed is set to 42 (random_state=42). The resulting variables are X_train (training input data), X_test (testing input data), Y_train (training target data), and Y_test (testing target data).

```python
print(model.metrics_names) #metrics of the model
```
[36] Python

```
['loss', 'accuracy']
```

#1. Save the model and use the saved model to predict on new text data (ex, "A lot of good things are happening. We are respected again throughout the world, and that's a great thing.@realDonaldTrump")

```python
model.save('sentimentAnalysis.h5') #Saving the model
```
[37] Python

```python
from keras.models import load_model #Importing the package for importing the saved model
model= load_model('sentimentAnalysis.h5') #loading the saved model
```
[38] Python

```
WARNING:tensorflow:Layer lstm will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
```

```python
print(integer_encoded)
print(data['sentiment'])
```
[39] Python

```
[1 2 1 ... 2 0 2]
0         Neutral
1        Positive
2         Neutral
3        Positive
4        Positive
           ...
13866    Negative
13867    Positive
13868    Positive
13869    Negative
13870    Positive
Name: sentiment, Length: 13871, dtype: object
```

Now printing the metrics, saving the model, then printing the accuracy of each data.

```python
# Predicting on the text data
sentence = ['A lot of good things are happening. We are respected again throughout the world, and that is a great thing.@realDonaldTrump']
sentence = tokenizer.texts_to_sequences(sentence) # Tokenizing the sentence
sentence = pad_sequences(sentence, maxlen=28, dtype='int32', value=0) # Padding the sentence
sentiment_probs = model.predict(sentence, batch_size=1, verbose=2)[0] # Predicting the sentence text
sentiment = np.argmax(sentiment_probs)

print(sentiment_probs)
if sentiment == 0:
    print("Neutral")
elif sentiment < 0:
    print("Negative")
elif sentiment > 0:
    print("Positive")
else:
    print("Cannot be determined")
```
[44]

```
1/1 - 0s - 22ms/epoch - 22ms/step
[0.3347626  0.16386913 0.5013683 ]
Positive
```

Now predicting the model by testing on a sentence. And the result is positive.

## 2. Apply GridSearchCV on the source code provided in the class

#2. Apply GridSearchCV on the source code provided in the class

```python
from keras.wrappers.scikit_learn import KerasClassifier #importing Keras classifier
from sklearn.model_selection import GridSearchCV #importing Grid search CV

model = KerasClassifier(build_fn=createmodel,verbose=2) #initiating model to test performance by applying multiple hyper parameters
batch_size= [10, 20, 40] #hyper parameter batch_size
epochs = [1, 2] #hyper parameter no. of epochs
param_grid= {'batch_size':batch_size, 'epochs':epochs} #creating dictionary for batch size, no. of epochs
grid  = GridSearchCV(estimator=model, param_grid=param_grid) #Applying dictionary with hyper parameters
grid_result= grid.fit(X_train,Y_train) #Fitting the model
# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_)) #best score, best hyper parameters
```
[45] Python

```
<ipython-input-45-6c99b49150f4>:4: DeprecationWarning: KerasClassifier is deprecated, use Sci-Keras (https://github.com/adriangb/scikeras) instead. See https://www.adriangb.com/scikeras/stable/migration
  model = KerasClassifier(build_fn=createmodel,verbose=2) #initiating model to test performance by applying multiple hyper parameters
WARNING:tensorflow:Layer lstm_1 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
744/744 - 108s - loss: 0.8243 - accuracy: 0.6433 - 108s/epoch - 145ms/step
186/186 - 2s - loss: 0.7794 - accuracy: 0.6681 - 2s/epoch - 12ms/step
WARNING:tensorflow:Layer lstm_2 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
744/744 - 106s - loss: 0.8200 - accuracy: 0.6476 - 106s/epoch - 143ms/step
186/186 - 2s - loss: 0.7681 - accuracy: 0.6719 - 2s/epoch - 11ms/step
WARNING:tensorflow:Layer lstm_3 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
744/744 - 107s - loss: 0.8218 - accuracy: 0.6480 - 107s/epoch - 143ms/step
186/186 - 2s - loss: 0.7843 - accuracy: 0.6869 - 2s/epoch - 12ms/step
WARNING:tensorflow:Layer lstm_4 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
744/744 - 106s - loss: 0.8325 - accuracy: 0.6387 - 106s/epoch - 143ms/step
186/186 - 2s - loss: 0.7679 - accuracy: 0.6615 - 2s/epoch - 12ms/step
WARNING:tensorflow:Layer lstm_5 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
744/744 - 107s - loss: 0.8203 - accuracy: 0.6440 - 107s/epoch - 143ms/step
186/186 - 2s - loss: 0.7734 - accuracy: 0.6679 - 2s/epoch - 11ms/step
WARNING:tensorflow:Layer lstm_6 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
Epoch 1/2
744/744 - 108s - loss: 0.8251 - accuracy: 0.6481 - 108s/epoch - 145ms/step
Epoch 2/2
744/744 - 96s - loss: 0.6777 - accuracy: 0.7098 - 96s/epoch - 129ms/step
186/186 - 2s - loss: 0.7344 - accuracy: 0.6902 - 2s/epoch - 12ms/step
WARNING:tensorflow:Layer lstm_7 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
Epoch 1/2
744/744 - 105s - loss: 0.8208 - accuracy: 0.6488 - 105s/epoch - 141ms/step
```

```
Epoch 1/2
744/744 - 105s - loss: 0.8208 - accuracy: 0.6488 - 105s/epoch - 141ms/step
Epoch 2/2
744/744 - 95s - loss: 0.6808 - accuracy: 0.7127 - 95s/epoch - 127ms/step
186/186 - 3s - loss: 0.7464 - accuracy: 0.6778 - 3s/epoch - 16ms/step
WARNING:tensorflow:Layer lstm_8 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
Epoch 1/2
744/744 - 108s - loss: 0.8200 - accuracy: 0.6455 - 108s/epoch - 145ms/step
Epoch 2/2
744/744 - 96s - loss: 0.6682 - accuracy: 0.7186 - 96s/epoch - 130ms/step
186/186 - 2s - loss: 0.7458 - accuracy: 0.6864 - 2s/epoch - 11ms/step
WARNING:tensorflow:Layer lstm_9 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
Epoch 1/2
744/744 - 107s - loss: 0.8252 - accuracy: 0.6452 - 107s/epoch - 144ms/step
Epoch 2/2
744/744 - 95s - loss: 0.6764 - accuracy: 0.7123 - 95s/epoch - 128ms/step
186/186 - 2s - loss: 0.7443 - accuracy: 0.6712 - 2s/epoch - 11ms/step
WARNING:tensorflow:Layer lstm_10 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
Epoch 1/2
744/744 - 105s - loss: 0.8182 - accuracy: 0.6490 - 105s/epoch - 141ms/step
Epoch 2/2
744/744 - 94s - loss: 0.6692 - accuracy: 0.7143 - 94s/epoch - 127ms/step
186/186 - 2s - loss: 0.7689 - accuracy: 0.6749 - 2s/epoch - 11ms/step
WARNING:tensorflow:Layer lstm_11 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
372/372 - 61s - loss: 0.8300 - accuracy: 0.6429 - 61s/epoch - 165ms/step
93/93 - 1s - loss: 0.7640 - accuracy: 0.6606 - 1s/epoch - 12ms/step
WARNING:tensorflow:Layer lstm_12 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
372/372 - 59s - loss: 0.8303 - accuracy: 0.6438 - 59s/epoch - 160ms/step
93/93 - 1s - loss: 0.7571 - accuracy: 0.6794 - 1s/epoch - 14ms/step
WARNING:tensorflow:Layer lstm_13 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
372/372 - 59s - loss: 0.8337 - accuracy: 0.6450 - 59s/epoch - 158ms/step
93/93 - 1s - loss: 0.7684 - accuracy: 0.6735 - 1s/epoch - 12ms/step
WARNING:tensorflow:Layer lstm_14 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
372/372 - 58s - loss: 0.8267 - accuracy: 0.6398 - 58s/epoch - 157ms/step
93/93 - 2s - loss: 0.7480 - accuracy: 0.6787 - 2s/epoch - 18ms/step
WARNING:tensorflow:Layer lstm_15 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
372/372 - 58s - loss: 0.8273 - accuracy: 0.6482 - 58s/epoch - 155ms/step
93/93 - 2s - loss: 0.7958 - accuracy: 0.6642 - 2s/epoch - 18ms/step
WARNING:tensorflow:Layer lstm_16 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
Epoch 1/2
372/372 - 59s - loss: 0.8283 - accuracy: 0.6447 - 59s/epoch - 159ms/step
Epoch 2/2
```

```
Epoch 1/2
372/372 - 59s - loss: 0.8283 - accuracy: 0.6447 - 59s/epoch - 159ms/step
Epoch 2/2
372/372 - 48s - loss: 0.6820 - accuracy: 0.7147 - 48s/epoch - 129ms/step
93/93 - 1s - loss: 0.7243 - accuracy: 0.6907 - 1s/epoch - 12ms/step
WARNING:tensorflow:Layer lstm_17 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
Epoch 1/2
372/372 - 59s - loss: 0.8281 - accuracy: 0.6407 - 59s/epoch - 158ms/step
Epoch 2/2
372/372 - 48s - loss: 0.6886 - accuracy: 0.7097 - 48s/epoch - 129ms/step
93/93 - 1s - loss: 0.7455 - accuracy: 0.6859 - 1s/epoch - 12ms/step
WARNING:tensorflow:Layer lstm_18 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
Epoch 1/2
372/372 - 58s - loss: 0.8366 - accuracy: 0.6407 - 58s/epoch - 155ms/step
Epoch 2/2
372/372 - 48s - loss: 0.6866 - accuracy: 0.7123 - 48s/epoch - 130ms/step
93/93 - 1s - loss: 0.7401 - accuracy: 0.6826 - 1s/epoch - 12ms/step
WARNING:tensorflow:Layer lstm_19 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
Epoch 1/2
372/372 - 61s - loss: 0.8347 - accuracy: 0.6399 - 61s/epoch - 164ms/step
Epoch 2/2
372/372 - 47s - loss: 0.6746 - accuracy: 0.7119 - 47s/epoch - 126ms/step
93/93 - 1s - loss: 0.7483 - accuracy: 0.6636 - 1s/epoch - 15ms/step
WARNING:tensorflow:Layer lstm_20 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
Epoch 1/2
372/372 - 59s - loss: 0.8256 - accuracy: 0.6414 - 59s/epoch - 159ms/step
Epoch 2/2
372/372 - 46s - loss: 0.6711 - accuracy: 0.7114 - 46s/epoch - 125ms/step
93/93 - 1s - loss: 0.7793 - accuracy: 0.6841 - 1s/epoch - 14ms/step
WARNING:tensorflow:Layer lstm_21 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
186/186 - 36s - loss: 0.8497 - accuracy: 0.6390 - 36s/epoch - 196ms/step
47/47 - 1s - loss: 0.7564 - accuracy: 0.6633 - 747ms/epoch - 16ms/step
WARNING:tensorflow:Layer lstm_22 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
186/186 - 37s - loss: 0.8519 - accuracy: 0.6326 - 37s/epoch - 198ms/step
47/47 - 1s - loss: 0.7828 - accuracy: 0.6482 - 766ms/epoch - 16ms/step
WARNING:tensorflow:Layer lstm_23 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
186/186 - 34s - loss: 0.8474 - accuracy: 0.6333 - 34s/epoch - 185ms/step
47/47 - 1s - loss: 0.7797 - accuracy: 0.6595 - 719ms/epoch - 15ms/step
WARNING:tensorflow:Layer lstm_24 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
186/186 - 36s - loss: 0.8389 - accuracy: 0.6409 - 36s/epoch - 192ms/step
47/47 - 1s - loss: 0.7430 - accuracy: 0.6830 - 700ms/epoch - 15ms/step
WARNING:tensorflow:Layer lstm_25 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
186/186 - 37s - loss: 0.8363 - accuracy: 0.6356 - 37s/epoch - 200ms/step
```

```
186/186 - 37s - loss: 0.8363 - accuracy: 0.6356 - 37s/epoch - 200ms/step
47/47 - 1s - loss: 0.7755 - accuracy: 0.6668 - 730ms/epoch - 16ms/step
WARNING:tensorflow:Layer lstm_26 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
Epoch 1/2
186/186 - 35s - loss: 0.8437 - accuracy: 0.6391 - 35s/epoch - 188ms/step
Epoch 2/2
186/186 - 24s - loss: 0.6866 - accuracy: 0.7086 - 24s/epoch - 131ms/step
47/47 - 1s - loss: 0.7250 - accuracy: 0.6859 - 705ms/epoch - 15ms/step
WARNING:tensorflow:Layer lstm_27 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
Epoch 1/2
186/186 - 36s - loss: 0.8450 - accuracy: 0.6347 - 36s/epoch - 193ms/step
Epoch 2/2
186/186 - 25s - loss: 0.6936 - accuracy: 0.7010 - 25s/epoch - 136ms/step
47/47 - 1s - loss: 0.7462 - accuracy: 0.6837 - 730ms/epoch - 16ms/step
WARNING:tensorflow:Layer lstm_28 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
Epoch 1/2
186/186 - 38s - loss: 0.8465 - accuracy: 0.6363 - 38s/epoch - 202ms/step
Epoch 2/2
186/186 - 24s - loss: 0.6809 - accuracy: 0.7076 - 24s/epoch - 129ms/step
47/47 - 1s - loss: 0.7555 - accuracy: 0.6799 - 737ms/epoch - 16ms/step
WARNING:tensorflow:Layer lstm_29 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
Epoch 1/2
186/186 - 36s - loss: 0.8497 - accuracy: 0.6370 - 36s/epoch - 192ms/step
Epoch 2/2
186/186 - 26s - loss: 0.6874 - accuracy: 0.7052 - 26s/epoch - 139ms/step
47/47 - 1s - loss: 0.7363 - accuracy: 0.6889 - 748ms/epoch - 16ms/step
WARNING:tensorflow:Layer lstm_30 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
Epoch 1/2
186/186 - 37s - loss: 0.8370 - accuracy: 0.6371 - 37s/epoch - 198ms/step
Epoch 2/2
186/186 - 26s - loss: 0.6795 - accuracy: 0.7098 - 26s/epoch - 140ms/step
47/47 - 1s - loss: 0.7777 - accuracy: 0.6652 - 730ms/epoch - 16ms/step
WARNING:tensorflow:Layer lstm_31 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
Epoch 1/2
465/465 - 74s - loss: 0.8138 - accuracy: 0.6524 - 74s/epoch - 159ms/step
Epoch 2/2
465/465 - 62s - loss: 0.6739 - accuracy: 0.7108 - 62s/epoch - 134ms/step
Best: 0.681371 using {'batch_size': 20, 'epochs': 2}
```

Initiating model to test performance by applying multiple hyper parameters. Defining the batch_size, no. of epochs and then creating a dictionary for batch size & no, of epochs. Then fitting the model and summarizing the results. The accuracy is 0.7108 with batch size 20 and epochs as 2.