Student ID: 700752115

Student Name: Rumana Shaik

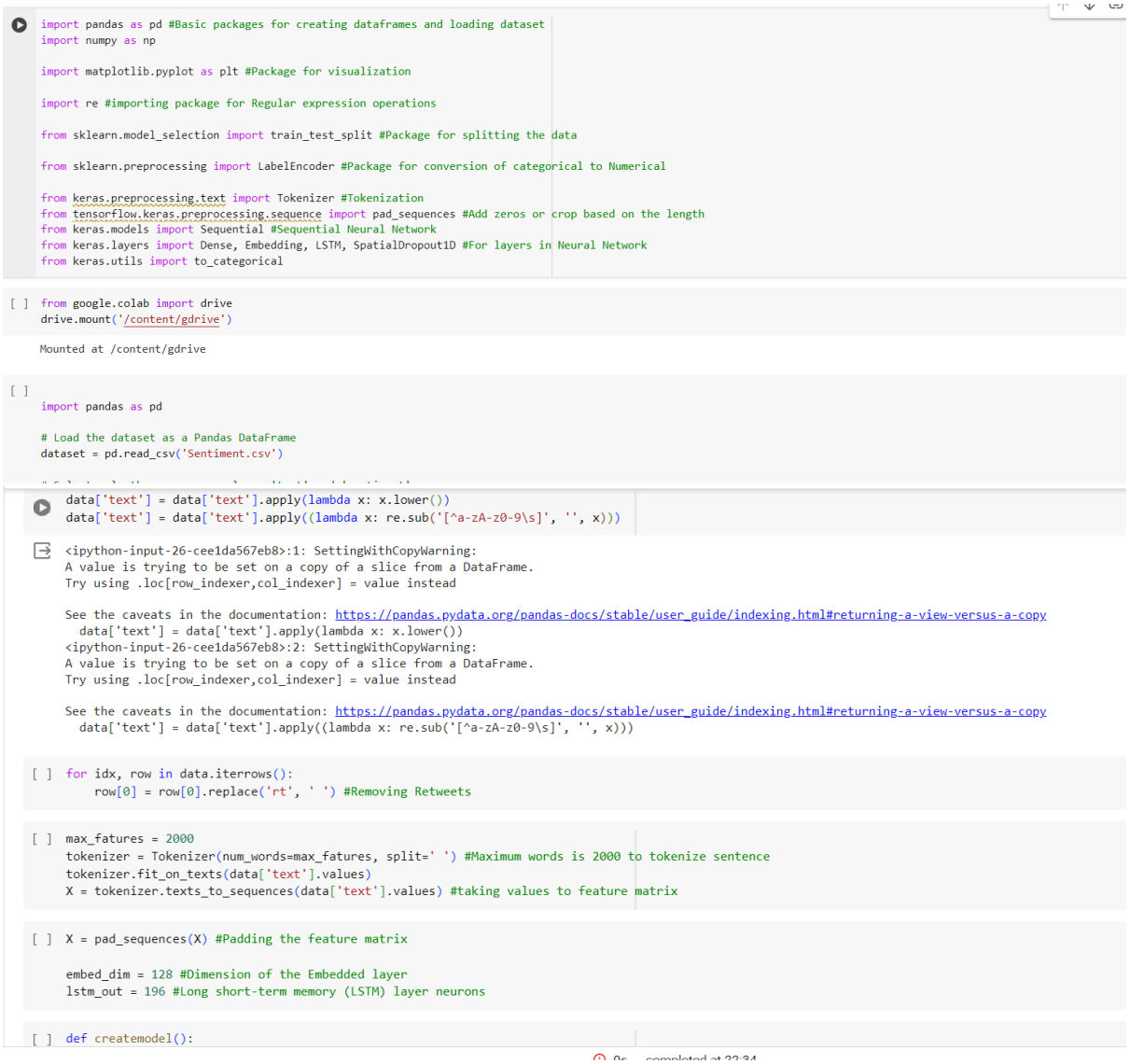
Neural Networks & Deep Learning – Assignment - 9

Github link: https://github.com/ShaikRumana301/Neural-Network-DL-ICP-9.git

Video Link: https://drive.google.com/file/d/1-d_RUX0323BMBBMtZ0TOewoc8l5f8F--/view?usp=sharing

In class programming:

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”)



```
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 import to_categorical
```

```
[ ] from google.colab import drive
    drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

```
[ ]
    import pandas as pd

    # Load the dataset as a Pandas DataFrame
    dataset = pd.read_csv('Sentiment.csv')
```

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

```
<ipython-input-26-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-26-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
```

```
[ ] 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
```

```
[ ] 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
```

```
[ ] def createmodel():
```

```
[ ] 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())
```

```
[ ] 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
```

```
[ ] batch_size = 32 #Batch size 32
    model = createmodel() #Function call to Sequential Neural Network
    model.fit(X_train, Y_train, epochs = 1, batch_size=batch_size, verbose = 2) #verbose the higher, the more messages
    score,acc = model.evaluate(X_test,Y_test,verbose=2,batch_size=batch_size) #evaluating the model
    print(score)
    print(acc)
```

```
291/291 - 51s - loss: 0.8261 - accuracy: 0.6442 - 51s/epoch - 175ms/step
144/144 - 3s - loss: 0.7469 - accuracy: 0.6689 - 3s/epoch - 22ms/step
0.7469469904899597
0.6688510179519653
```

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

```
['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")

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

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. W
  saving_api.save_model(
```

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

```
[ ] print(integer_encoded)
    print(data['sentiment'])
```

```
[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
```

0s completed at 22:34

```
[ ] # 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")
```

```
1/1 - 0s - 259ms/epoch - 259ms/step
[0.5638268  0.13124636 0.30492687]
Neutral
```

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

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

```
[2] pip install scikeras
```

```
Collecting scikeras
  Downloading scikeras-0.12.0-py3-none-any.whl (27 kB)
Requirement already satisfied: packaging>=0.21 in /usr/local/lib/python3.10/dist-packages (from scikeras) (24.0)
Requirement already satisfied: scikit-learn>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from scikeras) (1.2.2)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.0->scikeras) (1.25.2)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.0->scikeras) (1.11.4)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.0->scikeras) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.0->scikeras) (3.4.0)
Installing collected packages: scikeras
Successfully installed scikeras-0.12.0
```

```
[ ] from sklearn.model_selection import train_test_split
    from scikeras.wrappers import KerasClassifier
    from sklearn.model_selection import GridSearchCV

    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
```

```
186/186 - 24s - loss: 0.6869 - 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
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
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
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}
```