Neural Networks and Deep Learning-Assignment 5

Name: Sai Swetha Nambari ID:700742846

Github link: https://github.com/SwethaNam/nnassign5.git

Video link:
https://drive.google.com/file/d/1QJ90Ei2c7kPZXKv5peBz008asLsbEP54/view?usp=sharing

Question 1:

First we run the code given in the class as shown below before question 1:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import re
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense, Embedding, LSTM, SpatialDropout1D
from keras.utils.np_utils import to_categorical
```

```
from google.colab import drive
drive.mount('/content/gdrive')
path_to_csv = '/content/gdrive/My Drive/Sentiment.csv'
```

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

```
import pandas as pd
dset = pd.read_csv(path_to_csv, header=0)
mask = dset.columns.isin(['text', 'sentiment'])
data = dset.loc[:, mask]                                          # Keeping only the necessary columns
data['text'] = data['text'].apply(lambda x: x.lower())
data['text'] = data['text'].apply((lambda x: re.sub('[^a-zA-z0-9\s]', '', x)))
```

In this step we import the libraries and then import the the sentiment.csv from the drive. Later we read the csv file and select only the necessary columns text and sentient as shown above.

Next we use maximum words on 200o to tokenize the sentence and thencompile the model as shown bellow:

```python
for idx, row in data.iterrows():
    row[0] = row[0].replace('rt', ' ')
max_fatures = 2000
tokenizer = Tokenizer(num_words=max_fatures, split=' ')
tokenizer.fit_on_texts(data['text'].values)
X = tokenizer.texts_to_sequences(data['text'].values)

X = pad_sequences(X) #Padding the feature matrix

embed_dim = 128
lstm_out = 196
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))
    model.add(Dense(3,activation='softmax'))
    model.compile(loss = 'categorical_crossentropy', optimizer='adam',metrics = ['accuracy']) #Compiling the model
    return model
# print(model.summary())
```

```python
label_encoder = LabelEncoder() #Applying label Encoding on the label matrix
integer_encoded = label_encoder.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)

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)
score,acc = model.evaluate(X_test,Y_test,verbose=2,batch_size=batch_size)         #evaluating the model
print(score)
print(acc)
```

```
291/291 - 61s - loss: 0.8245 - accuracy: 0.6424 - 61s/epoch - 210ms/step
144/144 - 3s - loss: 0.7612 - accuracy: 0.6769 - 3s/epoch - 23ms/step
0.7611876726150513
0.6769331693649292
```

```python
print(model.metrics_names)
```

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

In the above screenshot, we apply label encoding on the label matrix and then fit the model and lastly evaluate the model.

Then we print the model metrics.

Question 1: Save the model and use the saved model to predict 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”)

```
[ ] #question 1:
    model.save('sentimentAnalysis.h5') #Saving the model
```

```
[ ] from keras.models import load_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
```

Here we have saved the model,and loaded ad printed the saved model.

```
# 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)
sentence = pad_sequences(sentence, maxlen=28, dtype='int32', value=0)
sentiment_probs = model.predict(sentence, batch_size=1, verbose=2)[0]
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 - 324ms/epoch - 324ms/step
[0.40014437 0.10924453 0.4906111 ]
Positive
```

Predicting on the text data.

Question 2:Apply GridSearchCV on the source code provided in the class

New Tab | Untitled1.ipynb - Colaboratory

colab.research.google.com/drive/1bPMHTQ2UR8bdoifRjirlVL7HEihsvzLz#scrollTo=PJY6ZfuJM2o_

Untitled1.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```python
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import GridSearchCV
model = KerasClassifier(build_fn=createmodel,verbose=2)
batch_size= [10, 20, 40]
epochs = [1, 2]
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
grid_result= grid.fit(X_train,Y_train) #Fitting the model

print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_)) #best score, best hyper parameters
```

```
372/372 - 62s - loss: 0.6827 - accuracy: 0.7097 - 62s/epoch - 168ms/step
93/93 - 3s - loss: 0.7318 - accuracy: 0.6783 - 3s/epoch - 35ms/step
Epoch 1/2
372/372 - 65s - loss: 0.8365 - accuracy: 0.6403 - 65s/epoch - 175ms/step
Epoch 2/2
372/372 - 62s - loss: 0.6765 - accuracy: 0.7147 - 62s/epoch - 168ms/step
93/93 - 3s - loss: 0.7480 - accuracy: 0.6918 - 3s/epoch - 35ms/step
Epoch 1/2
372/372 - 70s - loss: 0.8342 - accuracy: 0.6437 - 70s/epoch - 187ms/step
Epoch 2/2
372/372 - 68s - loss: 0.6697 - accuracy: 0.7166 - 68s/epoch - 182ms/step
93/93 - 2s - loss: 0.7546 - accuracy: 0.6620 - 2s/epoch - 25ms/step
Epoch 1/2
372/372 - 68s - loss: 0.8312 - accuracy: 0.6443 - 68s/epoch - 184ms/step
Epoch 2/2
372/372 - 64s - loss: 0.6711 - accuracy: 0.7099 - 64s/epoch - 172ms/step
93/93 - 2s - loss: 0.8009 - accuracy: 0.6362 - 2s/epoch - 22ms/step
186/186 - 43s - loss: 0.8446 - accuracy: 0.6381 - 43s/epoch - 233ms/step
47/47 - 1s - loss: 0.7671 - accuracy: 0.6482 - 1s/epoch - 30ms/step
```

```
186/186 - 44s - loss: 0.8503 - accuracy: 0.6340 - 44s/epoch - 236ms/step
Epoch 2/2
186/186 - 42s - loss: 0.6878 - accuracy: 0.7057 - 42s/epoch - 224ms/step
47/47 - 2s - loss: 0.7525 - accuracy: 0.6789 - 2s/epoch - 33ms/step
Epoch 1/2
186/186 - 45s - loss: 0.8436 - accuracy: 0.6373 - 45s/epoch - 242ms/step
Epoch 2/2
186/186 - 42s - loss: 0.6903 - accuracy: 0.7044 - 42s/epoch - 226ms/step
47/47 - 3s - loss: 0.7339 - accuracy: 0.6808 - 3s/epoch - 53ms/step
Epoch 1/2
186/186 - 41s - loss: 0.8470 - accuracy: 0.6374 - 41s/epoch - 222ms/step
Epoch 2/2
186/186 - 41s - loss: 0.6861 - accuracy: 0.7053 - 41s/epoch - 220ms/step
47/47 - 1s - loss: 0.7870 - accuracy: 0.6749 - 1s/epoch - 31ms/step
Epoch 1/2
930/930 - 154s - loss: 0.8103 - accuracy: 0.6552 - 154s/epoch - 165ms/step
Epoch 2/2
930/930 - 150s - loss: 0.6683 - accuracy: 0.7156 - 150s/epoch - 162ms/step
Best: 0.679758 using {'batch_size': 10, 'epochs': 2}
```