NN&DeepLearning_ICP10: LSTM

Bhavana Billa
700756590

GitHub link: https://github.com/BillaBhavana7/neuralN

```
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 Network
from keras.utils.np_utils import to_categorical
```

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

```
import pandas as pd
dataset = pd.read_csv(path_to_csv, header=0)
mask = dataset.columns.isin(['text', 'sentiment'])
data = dataset.loc[:, mask]
data['text'] = data['text'].apply(lambda x: x.lower())
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', ' ')

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) #taking values to feature matrix

X = pad_sequences(X)

embed_dim = 128
lstm_out = 196
```

```
def createmodel():
    model = Sequential()
    model.add(Embedding(max_fatures, embed_dim,input_length = X.shape[1]))
    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'])
    return model
```

```
labelencoder = LabelEncoder()
integer_encoded = labelencoder.fit_transform(data['sentiment'])
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()
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)
print(score)
print(acc)
```

```
291/291 - 56s - loss: 0.8208 - accuracy: 0.6530 - 56s/epoch - 193ms/step
144/144 - 2s - loss: 0.7517 - accuracy: 0.6796 - 2s/epoch - 11ms/step
0.751739501953125
0.6795544028282166
```

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

```
['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')
```

```
from keras.models import load_model
model= load_model('sentimentAnalysis.h5')
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
```

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

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

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

In [45]:

```
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
```

```
<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.html for help migrating.
  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
```

```
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}
```