ICP9

Vamsi Krishna Remala
Student id: 700744730

GitHub Link: https://github.com/vamsikrishnaremala/700744730_NNDL_ICP9

Video Link: https://drive.google.com/file/d/19jVzn3kR5rMvwEMXUsms-nsMSB5CyTv1/view?usp=sharing

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") Importing the necessary libraries and Sentiment.csv dataset

Iterating through the rows and removing retweets and tokenizing and padding the feature matrix.

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

```python
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
```

```python
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
```

```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())
```

```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
```

```python
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)
```

```
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.
291/291 - 53s - loss: 0.8235 - accuracy: 0.6435 - 53s/epoch - 183ms/step
144/144 - 2s - loss: 0.7565 - accuracy: 0.6693 - 2s/epoch - 13ms/step
0.7565157413482666
0.669287919998169
```

```python
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

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

     /usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3079: UserWarning: You are saving your model as an HDF5 file via
       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

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

```
 ▶  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
```

OUTPUT

Predicting on the text data and got neutral as the output

```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")
```

```
1/1 - 0s - 274ms/epoch - 274ms/step
[0.51147914 0.12045491 0.36806592]
Neutral
```

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

```
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) (23.2)
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) (1.23.5)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.0->scikeras) (1.11.3)
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) (3.2.0)
Installing collected packages: scikeras
Successfully installed scikeras-0.12.0
```

```python
from scikeras.wrappers import KerasClassifier #importing Keras classifier

from sklearn.model_selection import GridSearchCV #importing Grid search CV

model = KerasClassifier(model=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
```

Output:

```
47/47 - 1s - 646ms/epoch - 14ms/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 - 34s - loss: 0.8486 - accuracy: 0.6321 - 34s/epoch - 181ms/step
47/47 - 1s - 591ms/epoch - 13ms/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 - 33s - loss: 0.8401 - accuracy: 0.6348 - 33s/epoch - 176ms/step
47/47 - 1s - 867ms/epoch - 18ms/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 - 34s - loss: 0.8423 - accuracy: 0.6375 - 34s/epoch - 184ms/step
Epoch 2/2
186/186 - 25s - loss: 0.6871 - accuracy: 0.7049 - 25s/epoch - 136ms/step
47/47 - 1s - 590ms/epoch - 13ms/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 - 32s - loss: 0.8348 - accuracy: 0.6373 - 32s/epoch - 171ms/step
Epoch 2/2
186/186 - 24s - loss: 0.6907 - accuracy: 0.7059 - 24s/epoch - 131ms/step
47/47 - 1s - 672ms/epoch - 14ms/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 - 32s - loss: 0.8397 - accuracy: 0.6326 - 32s/epoch - 173ms/step
Epoch 2/2
186/186 - 24s - loss: 0.6822 - accuracy: 0.7097 - 24s/epoch - 127ms/step
47/47 - 1s - 592ms/epoch - 13ms/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 - 33s - loss: 0.8452 - accuracy: 0.6344 - 33s/epoch - 176ms/step
Epoch 2/2
186/186 - 27s - loss: 0.6867 - accuracy: 0.7081 - 27s/epoch - 144ms/step
47/47 - 1s - 678ms/epoch - 14ms/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 - 32s - loss: 0.8412 - accuracy: 0.6351 - 32s/epoch - 172ms/step
Epoch 2/2
186/186 - 24s - loss: 0.6751 - accuracy: 0.7111 - 24s/epoch - 131ms/step
47/47 - 1s - 599ms/epoch - 13ms/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
233/233 - 40s - loss: 0.8319 - accuracy: 0.6393 - 40s/epoch - 173ms/step
Epoch 2/2
233/233 - 30s - loss: 0.6757 - accuracy: 0.7124 - 30s/epoch - 129ms/step
Best: 0.683741 using {'batch_size': 40, 'epochs': 2}
```