Assignment-9 DEEKSHITH ATHMAKUR 700743388

Neural Networks and Deep learning

GitHub link: (https://github.com/DEEKSHITH-ATHMAKUR/ICP9)

```
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 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
```

```
[ ] import pandas as pd

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

    # Select only the necessary columns 'text' and 'sentiment'
    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)))
```

```
<ipython-input-17-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-17-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
```

+ Code + Text

```
[ ] 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():
        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 data, 33% test data split
```

```
[ ] 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 - 51s - loss: 0.8223 - accuracy: 0.6493 - 51s/epoch - 175ms/step
144/144 - 2s - loss: 0.7555 - accuracy: 0.6796 - 2s/epoch - 11ms/step
0.75551837682724
0.6795544028282166
```

```
[ ] 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:3079: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This fi
  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 fallback when running on GPU
```

```
[ ] 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
```

```
# 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 - 250ms/epoch - 250ms/step
[0.6810064  0.11271847 0.20627514]
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
```

```
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
```

```
93/93 - 1s - 920ms/epoch - 10ms/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 - 55s - loss: 0.8161 - accuracy: 0.6475 - 55s/epoch - 147ms/step
Epoch 2/2
372/372 - 46s - loss: 0.6656 - accuracy: 0.7174 - 46s/epoch - 125ms/step
93/93 - 1s - 974ms/epoch - 10ms/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 - 33s - loss: 0.8516 - accuracy: 0.6330 - 33s/epoch - 179ms/step
47/47 - 1s - 629ms/epoch - 13ms/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 - 32s - loss: 0.8379 - accuracy: 0.6356 - 32s/epoch - 172ms/step
47/47 - 1s - 693ms/epoch - 15ms/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 - 32s - loss: 0.8476 - accuracy: 0.6334 - 32s/epoch - 173ms/step
47/47 - 1s - 591ms/epoch - 13ms/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 - 32s - loss: 0.8433 - accuracy: 0.6356 - 32s/epoch - 175ms/step
47/47 - 1s - 596ms/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 - 32s - loss: 0.8455 - accuracy: 0.6381 - 32s/epoch - 172ms/step
47/47 - 1s - 603ms/epoch - 13ms/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 - 31s - loss: 0.8499 - accuracy: 0.6316 - 31s/epoch - 166ms/step
Epoch 2/2
186/186 - 24s - loss: 0.6903 - accuracy: 0.7038 - 24s/epoch - 128ms/step
47/47 - 1s - 630ms/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 - 34s - loss: 0.8451 - accuracy: 0.6388 - 34s/epoch - 183ms/step
```

```
Epoch 2/2
186/186 - 24s - loss: 0.6903 - accuracy: 0.7038 - 24s/epoch - 128ms/step
47/47 - 1s - 630ms/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 - 34s - loss: 0.8451 - accuracy: 0.6388 - 34s/epoch - 183ms/step
Epoch 2/2
186/186 - 24s - loss: 0.6893 - accuracy: 0.7061 - 24s/epoch - 131ms/step
47/47 - 1s - 621ms/epoch - 13ms/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: 0.8523 - accuracy: 0.6312 - 32s/epoch - 173ms/step
Epoch 2/2
186/186 - 24s - loss: 0.6835 - accuracy: 0.7086 - 24s/epoch - 128ms/step
47/47 - 1s - 634ms/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: 0.8492 - accuracy: 0.6348 - 33s/epoch - 177ms/step
Epoch 2/2
186/186 - 24s - loss: 0.6868 - accuracy: 0.6994 - 24s/epoch - 130ms/step
47/47 - 1s - 603ms/epoch - 13ms/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 - 33s - loss: 0.8461 - accuracy: 0.6371 - 33s/epoch - 176ms/step
Epoch 2/2
186/186 - 24s - loss: 0.6786 - accuracy: 0.7102 - 24s/epoch - 127ms/step
47/47 - 1s - 588ms/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 - 37s - loss: 0.8307 - accuracy: 0.6451 - 37s/epoch - 158ms/step
Epoch 2/2
233/233 - 30s - loss: 0.6809 - accuracy: 0.7091 - 30s/epoch - 129ms/step
Best: 0.680404 using {'batch_size': 40, 'epochs': 2}
```