Neural Networks and Deep Learning – ICP 9

GitHub Link: https://github.com/srija1609/NNDL_ICP-9
Name: Baby Srija Bitra
Student ID: 700755908

LSTM:

Use Case Description: 1. Sentiment Analysis on the Twitter dataset.

Recurrent Neural Networks (RNN): These are designed to process sequential data, where the output of a neuron can be fed back as input to the same neuron or to other neurons in the network. RNNs are well-suited for tasks such as time series prediction, speech recognition, and natural language processing.

Long Short-Term Memory (LSTM) Networks: These are a type of RNN that address the issue of vanishing gradients and can capture long-term dependencies in sequential data. LSTMs are widely used for tasks that require modeling of sequences with long-term dependencies.

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")**

To perform this task I have first imported the required packages.
keras.models.Sequential: This is a class from the Keras library used for creating a sequential neural network, where layers are added one by one in a sequential manner.
keras.utils.np_utils.to_categorical: This is a function from the Keras library used for converting numerical labels into one-hot encoded vectors, typically used for multi-class classification tasks.
Then I have written the code which reads a CSV file using pandas and loads it into a DataFrame named 'dataset'. The 'path_to_csv' variable should be replaced with the actual file path to the CSV file.

Then, it creates a boolean mask 'mask' to filter the columns in the DataFrame. It uses the 'isin' method to check if the column names 'text' and 'sentiment' are present in the columns of the 'dataset' DataFrame.

Next, it selects only the columns that are True in the 'mask' using the 'loc' method, and assigns it to a new DataFrame named 'data' as shown below:

```
+ Code  + Text

[ ]  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.np_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(path_to_csv, header=0)

     # 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()): This line of code uses the 'apply' method to apply a lambda function to each element in the 'text' column of the 'data' DataFrame. The lambda function converts each element to lowercase using the 'lower()' method.

data['text'] = data['text'].apply((lambda x: re.sub('[^a-zA-z0-9\s]', '', x))): This line of code also uses the 'apply' method to apply a lambda function to each element in the 'text' column of the 'data' DataFrame.

Then the following code is used to remove retweets from the 'text' column in the 'data' DataFrame.

The code iterates through each row in the 'data' DataFrame using the 'iterrows()' method. For each row, it replaces all occurrences of the string 'rt' with a space character using the 'replace()' method. This is done to remove retweets, as 'rt' is often used as a prefix in tweets to indicate a retweet.

```
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-29-ce1da567eb8>: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-29-ce1da567eb8>: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)))
```

+ Code + Text

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

The following sets the maximum number of features to 2000 using the 'max_fatures' variable. It then initializes a tokenizer object from the Keras 'Tokenizer' class. 'texts_to_sequences()' method is used to convert the text values in the 'text' column of the 'data' DataFrame into sequences of integers, with each integer representing the index of a word in the tokenizer's vocabulary.

The model architecture consists of the following layers:

Embedding layer: This layer creates word embeddings for the input text. It takes the maximum number of features (max_fatures) as the input dimension, the embedding dimension (embed_dim) as the output dimension.

LSTM (Long Short-Term Memory) layer: This layer is a type of recurrent layer that can capture long-range dependencies in sequential data. It has 196 LSTM cells (neurons) and a dropout of 0.2.

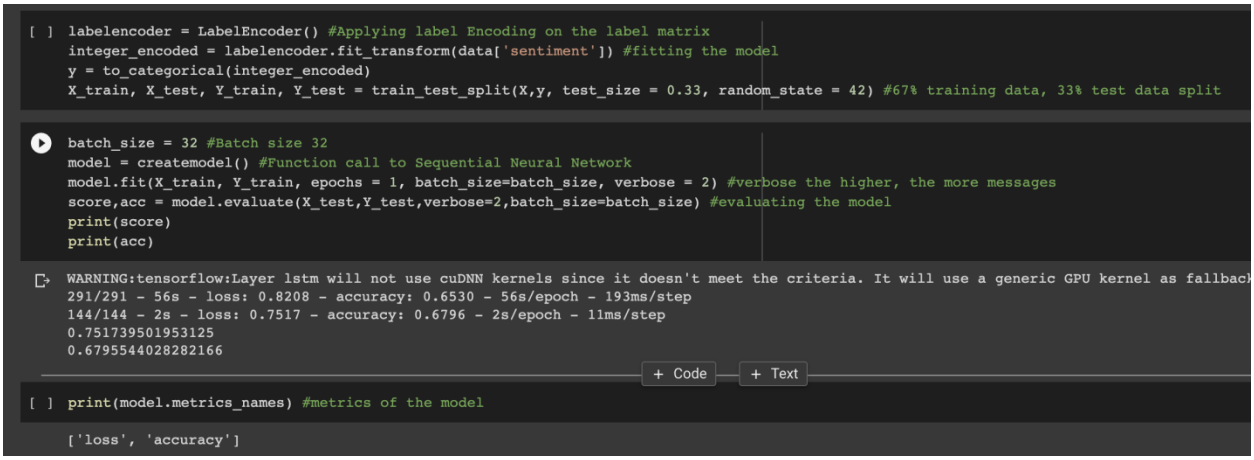
Dense output layer: This layer has 3 output neurons, representing the three sentiment classes (positive, neutral, negative). Model compilation: The model is compiled using the categorical_crossentropy loss function, which is commonly used for multi-class classification problems.

```
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
```

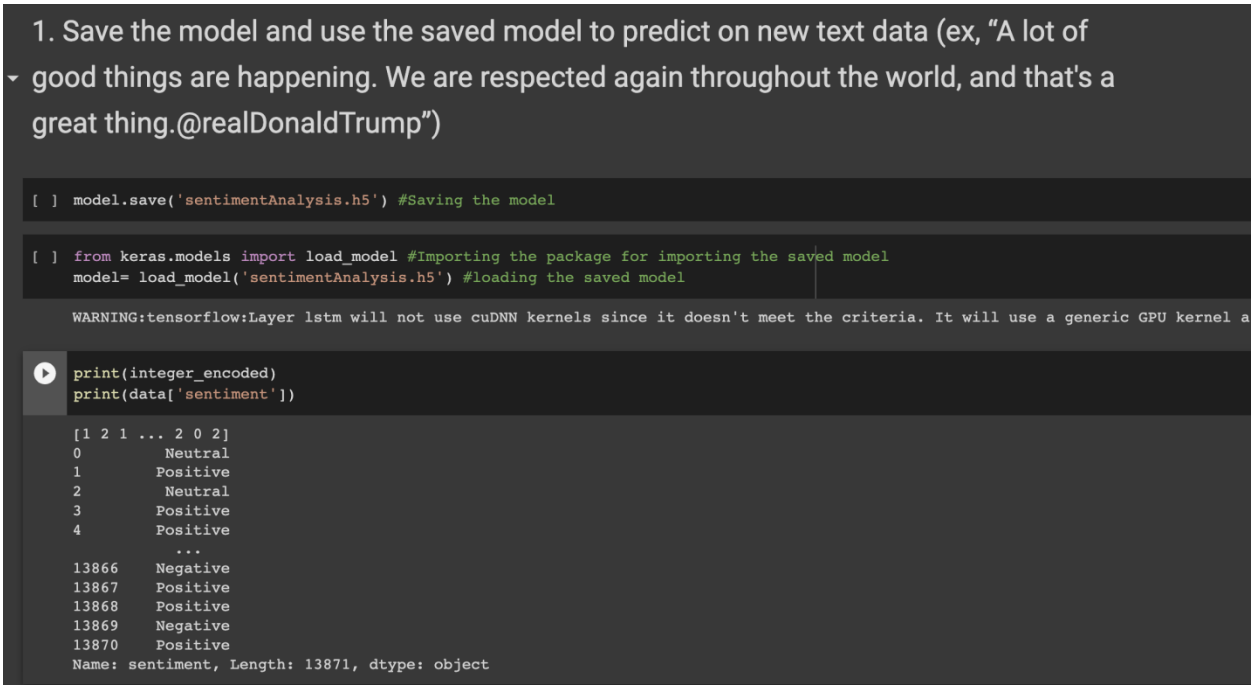
```
def createmodel():
    model = Sequential() #Sequential Neural Network
    model.add(Embedding(max_fatures, embed_dim,input_length = X.shape[1])) #input dimension 2000 Neurons, output dimension 128 Neurons
    model.add(LSTM(lstm_out, dropout=0.2, recurrent_dropout=0.2)) #Drop out 20%, 196 output 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())
```

I have now given the code that trains the created model using the fit() function. Finally, it prints the score (loss) and accuracy of the model on the test data using the print() function.



```
[ ] 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 fallbac
    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

                                                    + Code   + Text

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

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

The code is loading a saved model using the load model() function from the keras.models module. The saved model is named 'sentimentAnalysis.h5'.



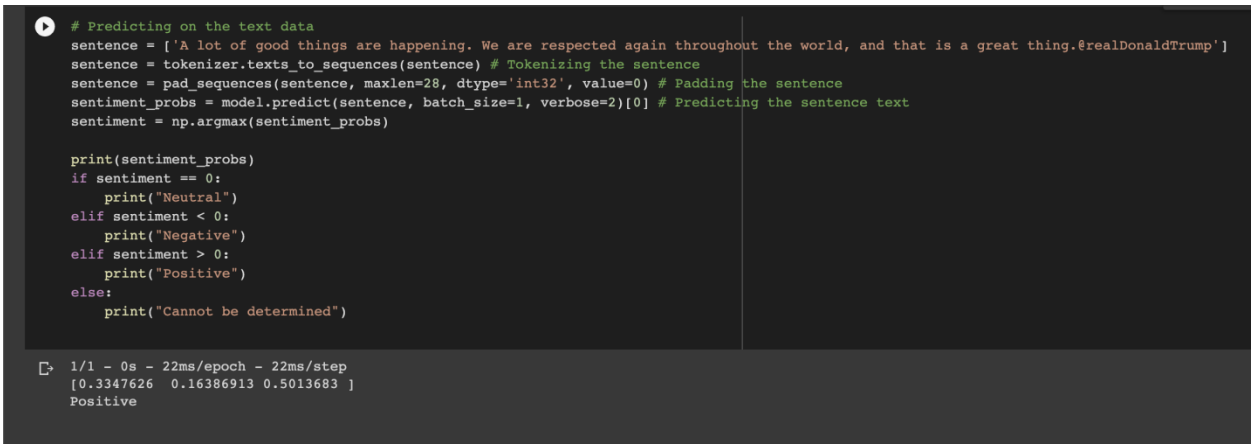
```
1. Save the model and use the saved model to predict on new text data (ex, “A lot of
good things are happening. We are respected again throughout the world, and that's a
great thing.@realDonaldTrump”)

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

[ ] 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 a

    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
```

Now I have written the code which is predicting the sentiment label for a given text sentence using the trained model.

sentence: The text sentence for which sentiment prediction is to be made.
tokenizer.texts_to_sequences(sentence): Tokenizes the text sentence using the same tokenizer that was used during training.
pad_sequences(sentence, maxlen=28, dtype='int32', value=0): Pads the tokenized sentence to have a fixed length of 28, which should match the input length expected by the model. Padding is done with zeros (0) to make all sentences of the same length.

model.predict(sentence, batch_size=1, verbose=2)[0]: Predicts the sentiment probabilities for the given sentence using the loaded model. batch_size is set to 1, as we are making predictions for a single sentence. verbose is set to 2 to display progress during prediction. The predicted probabilities are stored in sentiment_probs, which is a numpy array.

np.argmax(sentiment_probs): Retrieves the index of the highest predicted probability from sentiment_probs, which corresponds to the predicted sentiment label.



```
# 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")
```

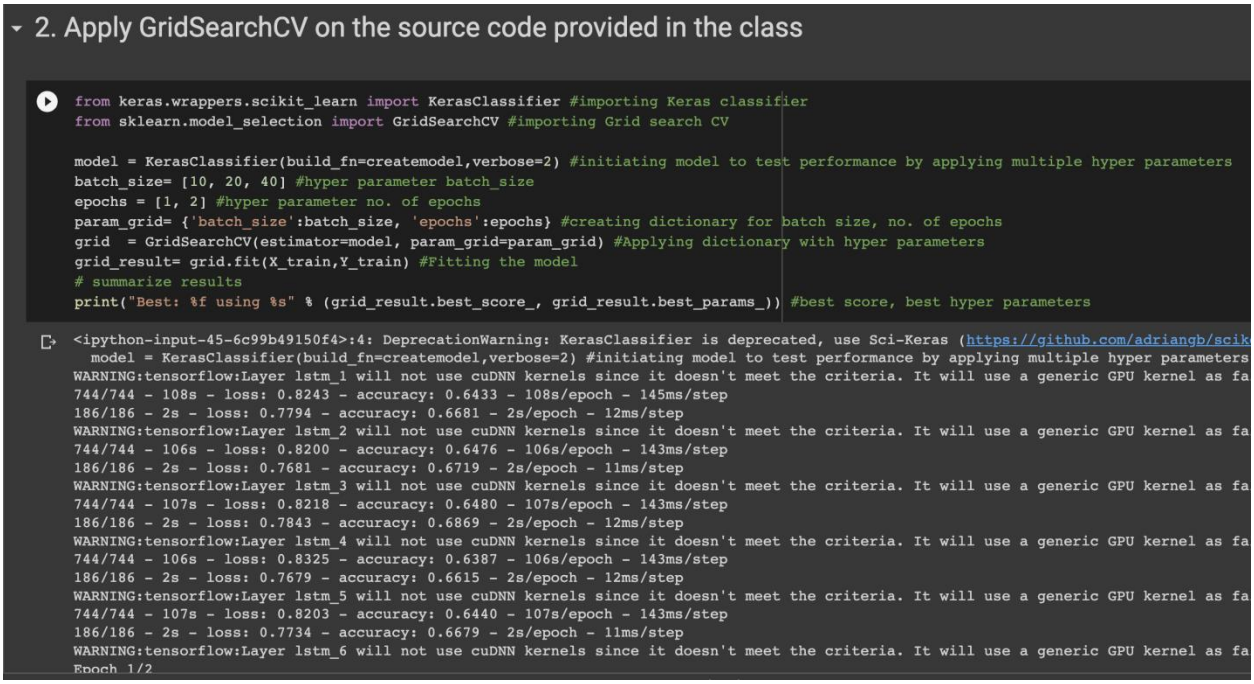
```
1/1 - 0s - 22ms/epoch - 22ms/step
[0.3347626  0.16386913 0.5013683 ]
Positive
```

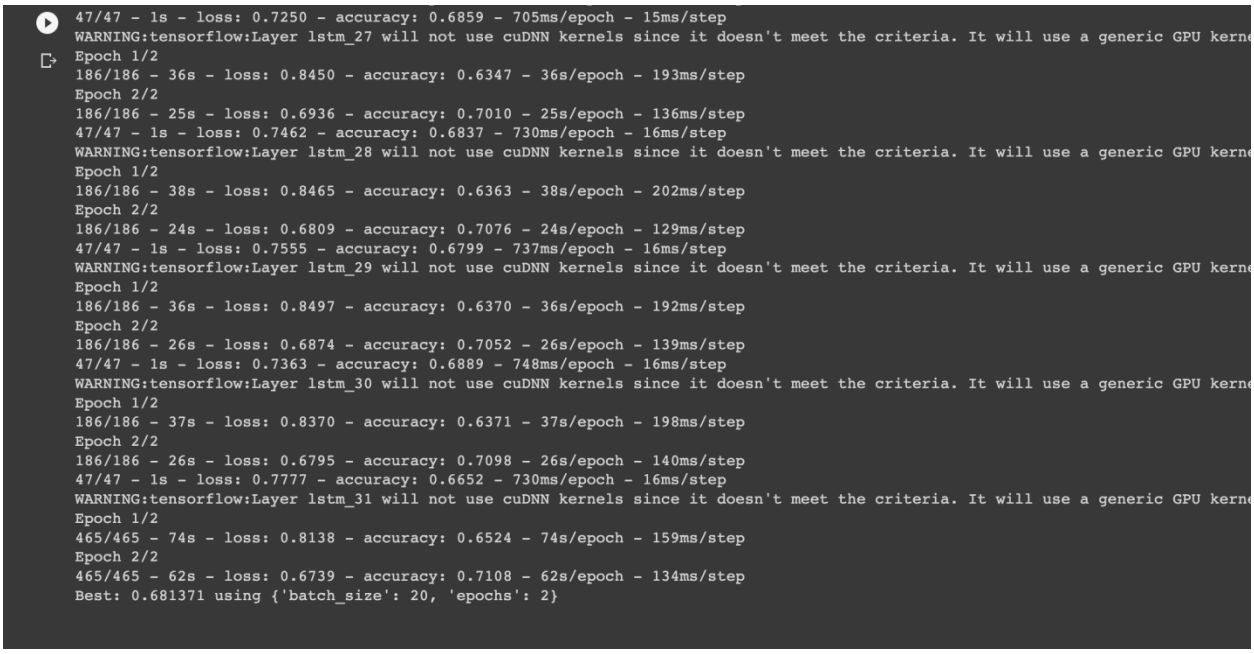
2. Apply GridSearchCV on the source code provided in the class

To perform this task I have written the code snippet which is using Grid Search Cross-Validation (GridSearchCV) from scikit-learn to search for the best hyperparameters for the KerasClassifier model.

KerasClassifier(build_fn=createmodel, verbose=2): The KerasClassifier is used as an estimator in GridSearchCV. It takes the createmodel() function as an argument, which returns the compiled Keras model. verbose=2 specifies the verbosity level during training.
batch_size: A hyperparameter for the batch size used during training.
GridSearchCV(estimator=model, param_grid=param_grid): GridSearchCV is initialized with the KerasClassifier model and the hyperparameter grid to search.

grid_result.best_score_: After fitting the model, the best_score_ attribute of the grid_result object provides the best mean cross-validated score across all folds for the best hyperparameter combination.
grid_result.best_params_: The best_params_ attribute of the grid_result object provides the best hyperparameter combination that resulted in the best score.





The output shows the progress of the model training using GridSearchCV for hyperparameter tuning.

Epoch 1/2: The model is trained for the first epoch with a batch size of 20. It took 74 seconds to complete the epoch, and the loss is 0.8138 with an accuracy of 0.6524.
Epoch 2/2: The model is trained for the second epoch with a batch size of 20. It took 62 seconds to complete the epoch, and the loss is 0.6739 with an accuracy of 0.7108.
After training, the best mean cross-validated score across all folds is found to be 0.681371, and the best hyperparameter combination is {'batch_size': 20, 'epochs': 2}, which resulted in this best score.