Neural Networks and Deep Learning ICP-5:

Student Details:

Student Id No: 700745501

Name: Pavan Naga Sai Gullapalli

CRN: 31176

Video link: https://drive.google.com/file/d/1IviND5x25YxatD61-pbxT5_R4yjvr61p/view?usp=drive_link

Github link:
https://github.com/PavanNagaSaiG/Neural_Neetwork_DeepLearning

Question-1:

```
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
```

- Import pandas as pd: Python library for data manipulation and analysis.
- Import numpy as np: Python library for working with arrays.
- Import matplotlib.pyplot as plt: Module from matplotlib library for easy-to-use plotting.
- Import re: Module for working with Regular Expressions in text operations.
- Import train_test_split from sklearn.model_selection: Function for splitting data into training and testing subsets.
- Import LabelEncoder from sklearn.preprocessing: Class for converting categorical data into numeric form.

- Import Tokenizer from keras.preprocessing.text: Class for text tokenization.
- Import pad_sequences from tensorflow.keras.preprocessing.sequence: Function for ensuring sequences have the same length.
- Import Sequential from keras.models: Model type for building plain stack of layers.
- Import Dense, Embedding, LSTM, SpatialDropout1D from keras.layers: Types of layers that can be added to Sequential models.
- Import to_categorical from keras.utils.np_utils: Function for converting class vector to binary class matrix.

```
import pandas as pd

path_to_csv = '/content/Sentiment.csv'
data = pd.read_csv(path_to_csv)

# Select only the necessary columns 'text' and 'sentiment'
mask = data.columns.isin(['text', 'sentiment'])
data = data.loc[:, mask]
```

```
[4]  # 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)))
```

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

```
[6]  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
```

```
[7]  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
```

- **for idx, row in data.iterrows():** - This line starts a loop that goes through each row in the DataFrame 'data'.
- **row[0] = row[0].replace('rt', ' ')** - This line replaces any instances of 'rt' in the first column of the current row with a space, effectively removing Twitter retweet indicators.

- **max_fatures = 2000** - This line sets a variable named 'max_fatures' to 2000, which will be used as the maximum number of words to consider when tokenizing the text.
- **tokenizer = Tokenizer(num_words=max_fatures, split=' ')** - This line creates a Tokenizer object, which is used to convert text into sequences of integers, based on word frequency. The number of words to consider is defined by 'max_fatures' and words are split by spaces.
- **tokenizer.fit_on_texts(data['text'].values)** - This line fits the tokenizer on the texts in the 'text' column of the 'data' DataFrame, effectively learning the vocabulary of the text.
- **X = tokenizer.texts_to_sequences(data['text'].values)** - This line uses the fitted tokenizer to convert the texts in the 'text' column of the 'data' DataFrame into sequences of integers, which can then be used for further analysis or model training.

```
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
```

```
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)
```

```
291/291 - 56s - loss: 0.8191 - accuracy: 0.6444 - 56s/epoch - 194ms/step
144/144 - 3s - loss: 0.7384 - accuracy: 0.6820 - 3s/epoch - 24ms/step
0.7383694052696228
0.6819571852684021
```

```
print(model.metrics_names) #metrics of the model
```

- Apply padding to the feature matrix X using the `pad_sequences` function, ensuring all sequences are of equal length. Define the dimensions of the embedded layer as 128 and the number of LSTM layer neurons as 196. Set up a function to create a sequential neural network model.
- Build the sequential neural network model by adding an embedding layer, LSTM layer, and a dense layer with the softmax activation function. Then compile the model with a loss function, optimizer, and performance metrics.
- Use a LabelEncoder to transform the 'sentiment' column of the data into integer-encoded labels. Convert these integer-encoded labels to one-hot encoded categorical labels for model training.

- Split the data into training and testing datasets, with the test size set at 33% and a random seed of 42 for reproducibility. Set the batch size to 32 for model training.
- Create the model using the defined function and train it for 1 epoch using the training data. The verbosity is set to 2 to show the intermediate progress messages during the training process.
- Evaluate the model's performance on the test data, storing the loss (score) and accuracy. Print these values along with the names of the metrics used to evaluate the 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) # 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 - 316ms/epoch - 316ms/step
[0.5062279  0.14353454 0.3502375 ]
Neutral
```

- Save the trained machine learning model in a file named 'sentimentAnalysis.h5'.
- Import the `load_model` function from the keras.models module, which enables loading previously saved models.
- Load the previously saved model 'sentimentAnalysis.h5' using the `load_model` function.
- Display the contents of the 'integer_encoded' variable, which is expected to contain the integer-encoded representation of the data.
- Print the 'sentiment' column from the DataFrame, which is expected to contain the sentiment labels for the dataset.

```
Epoch 2/2
186/186 - 34s - loss: 0.6887 - accuracy: 0.7105 - 34s/epoch - 180ms/step
47/47 - 1s - loss: 0.7411 - accuracy: 0.6880 - 1s/epoch - 29ms/step
Epoch 1/2
186/186 - 36s - loss: 0.8498 - accuracy: 0.6296 - 36s/epoch - 193ms/step
Epoch 2/2
186/186 - 33s - loss: 0.6981 - accuracy: 0.7030 - 33s/epoch - 180ms/step
47/47 - 1s - loss: 0.7437 - accuracy: 0.6798 - 1s/epoch - 29ms/step
Epoch 1/2
186/186 - 38s - loss: 0.8444 - accuracy: 0.6331 - 38s/epoch - 203ms/step
Epoch 2/2
186/186 - 34s - loss: 0.6917 - accuracy: 0.7064 - 34s/epoch - 181ms/step
47/47 - 1s - loss: 0.7937 - accuracy: 0.6755 - 1s/epoch - 29ms/step
Epoch 1/2
233/233 - 46s - loss: 0.8291 - accuracy: 0.6423 - 46s/epoch - 196ms/step
Epoch 2/2
233/233 - 42s - loss: 0.6853 - accuracy: 0.7084 - 42s/epoch - 180ms/step
Best: 0.680512 using {'batch_size': 40, 'epochs': 2}
```

- Import the `KerasClassifier` wrapper from Keras, allowing Keras models to utilize functionalities from the scikit-learn library.
- Import `GridSearchCV` from scikit-learn, a tool enabling exhaustive parameter tuning by searching over specified parameter values for an estimator.
- Initialize a `KerasClassifier` object with the `createmodel` function as the model architecture to be trained, setting the verbosity level to 2.
- Define a list of batch sizes and epoch values to be tested in the grid search.
- Create a dictionary `param_grid` where the keys are the parameter names to optimize and the values are the corresponding lists of values for those parameters.
- Initialize a `GridSearchCV` object with the defined model and parameter grid to conduct the grid search.
- Fit the model with the training data, initiating the grid search over the parameter grid, testing all possible combinations of batch sizes and epochs.
- Print the highest validation score achieved during the grid search and the parameters that resulted in that score.