# Assignment 5

**Student Name: Krishna Vamsi Koppula**
**Student ID: 700745021**

**GitHub Link:**
**https://github.com/Krishnavamsikoppula/Assignment_5_NNDL_Summer**

**Video Link: https://drive.google.com/file/d/1fqzUcDYr-ZBZj8u58L92sU82E1DITz3J/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")

```
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
```

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

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

# Preprocess the text data
data['text'] = data['text'].apply(lambda x: x.lower())
data['text'] = data['text'].apply(lambda x: re.sub('[^a-zA-Z0-9\s]', '', x))
data['text'] = data['text'].apply(lambda x: x.replace('rt', ' '))  # Remove 'rt' (Retweets)
```

```
<ipython-input-5-3b7761336ca1>:9: 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-5-3b7761336ca1>:10: 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))
<ipython-input-5-3b7761336ca1>:11: 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.replace('rt', ' '))  # Remove 'rt' (Retweets)
```

```
# Define the function to create the LSTM model
def createmodel():
    model = Sequential()
    model.add(Embedding(max_features, 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

# Tokenization
max_features = 2000
tokenizer = Tokenizer(num_words=max_features, split=' ')
tokenizer.fit_on_texts(data['text'].values)
X = tokenizer.texts_to_sequences(data['text'].values)
X = pad_sequences(X)

# Label Encoding
label_encoder = LabelEncoder()
integer_encoded = label_encoder.fit_transform(data['sentiment'])
y = to_categorical(integer_encoded)
```

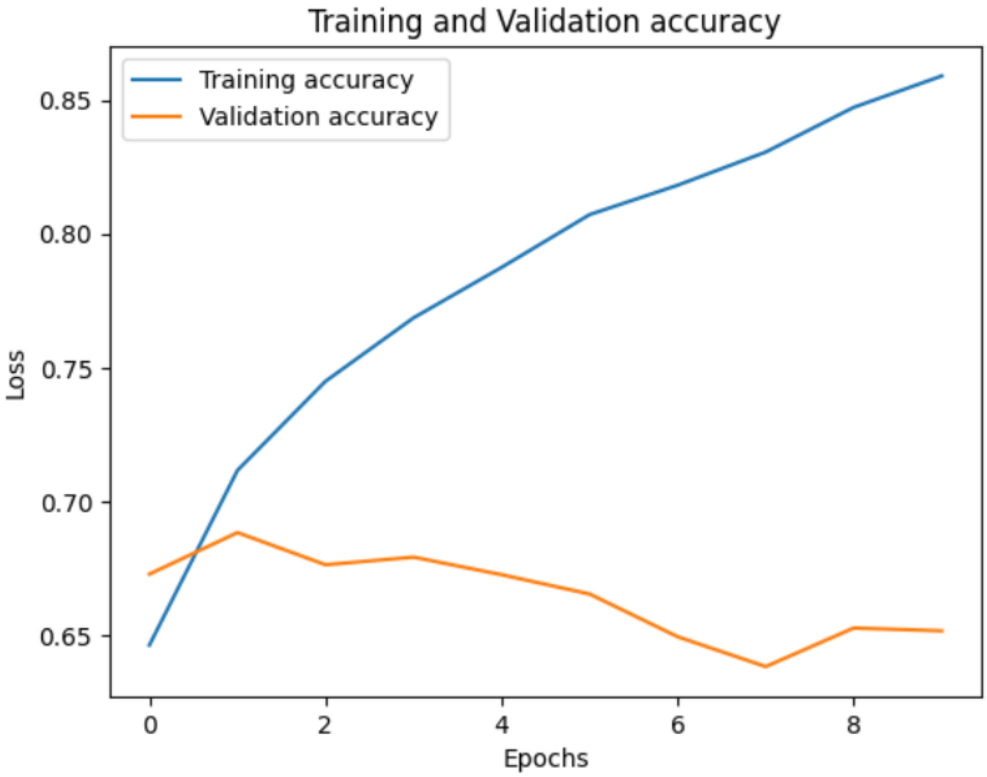
[7]
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

```python
# LSTM Model Architecture
embed_dim = 128
lstm_out = 196

model = Sequential()
model.add(Embedding(max_features, 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'])

# Model Summary
print(model.summary())

# Train the model
history = model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test), verbose=2)
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 28, 128)           256000

 lstm (LSTM)                 (None, 196)               254800

 dense (Dense)               (None, 3)                 591

=================================================================
Total params: 511,391
Trainable params: 511,391
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/10
291/291 - 33s - loss: 0.8253 - accuracy: 0.6465 - val_loss: 0.7528 - val_accuracy: 0.6730 - 33s/epoch - 115ms/step
Epoch 2/10
291/291 - 30s - loss: 0.6818 - accuracy: 0.7117 - val_loss: 0.7303 - val_accuracy: 0.6885 - 30s/epoch - 103ms/step
Epoch 3/10
291/291 - 30s - loss: 0.6166 - accuracy: 0.7450 - val_loss: 0.7560 - val_accuracy: 0.6765 - 30s/epoch - 103ms/step
Epoch 4/10
291/291 - 30s - loss: 0.5661 - accuracy: 0.7686 - val_loss: 0.7918 - val_accuracy: 0.6793 - 30s/epoch - 104ms/step
Epoch 5/10
291/291 - 33s - loss: 0.5184 - accuracy: 0.7875 - val_loss: 0.8440 - val_accuracy: 0.6728 - 33s/epoch - 113ms/step
Epoch 6/10
291/291 - 30s - loss: 0.4824 - accuracy: 0.8072 - val_loss: 0.8733 - val_accuracy: 0.6656 - 30s/epoch - 102ms/step
Epoch 7/10
291/291 - 31s - loss: 0.4429 - accuracy: 0.8181 - val_loss: 0.9958 - val_accuracy: 0.6496 - 31s/epoch - 107ms/step
Epoch 8/10
291/291 - 31s - loss: 0.4122 - accuracy: 0.8305 - val_loss: 1.0711 - val_accuracy: 0.6385 - 31s/epoch - 105ms/step
Epoch 9/10
291/291 - 30s - loss: 0.3754 - accuracy: 0.8472 - val_loss: 1.1761 - val_accuracy: 0.6529 - 30s/epoch - 105ms/step
Epoch 10/10
291/291 - 32s - loss: 0.3510 - accuracy: 0.8589 - val_loss: 1.1748 - val_accuracy: 0.6518 - 32s/epoch - 109ms/step
```

```python
# Evaluate the model on test data
score, accuracy = model.evaluate(X_test, y_test, verbose=2, batch_size=32)
print("Test Loss:", score)
print("Test Accuracy:", accuracy)


# Plot training and validation accuracy over epochs
plt.plot(history.history['accuracy'], label='Training accuracy')
plt.plot(history.history['val_accuracy'], label='Validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title('Training and Validation accuracy')
plt.show()

# Plot training and validation loss over epochs
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title('Training and Validation Loss')
plt.show()
```

```
144/144 - 2s - loss: 1.1748 - accuracy: 0.6518 - 2s/epoch - 17ms/step
Test Loss: 1.1747829914093018
Test Accuracy: 0.6518130302429199
```




```
# Save the trained model
model.save('sentimentAnalysis.h5')
```

```
[11] from keras.models import load_model

model = load_model('sentimentAnalysis.h5')
```

```
[12] # Define the text data to predict sentiment
sentence = ['A lot of good things are happening. We are respected again throughout the world, and that is a great thing. @realDonaldTrump']

# Tokenize and pad the sentence
sentence = tokenizer.texts_to_sequences(sentence)
sentence = pad_sequences(sentence, maxlen=28, dtype='int32', value=0)
```

```python
# Make predictions using the loaded model
sentiment_probs = model.predict(sentence, batch_size=1, verbose=2)[0]

# Convert sentiment probabilities to sentiment label
sentiment = np.argmax(sentiment_probs)

# Print the sentiment label
if sentiment == 0:
    print("Neutral")
elif sentiment < 0:
    print("Negative")
elif sentiment > 0:
    print("Positive")
else:
    print("Cannot be determined")
```

```
1/1 - 0s - 255ms/epoch - 255ms/step
Positive
```

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

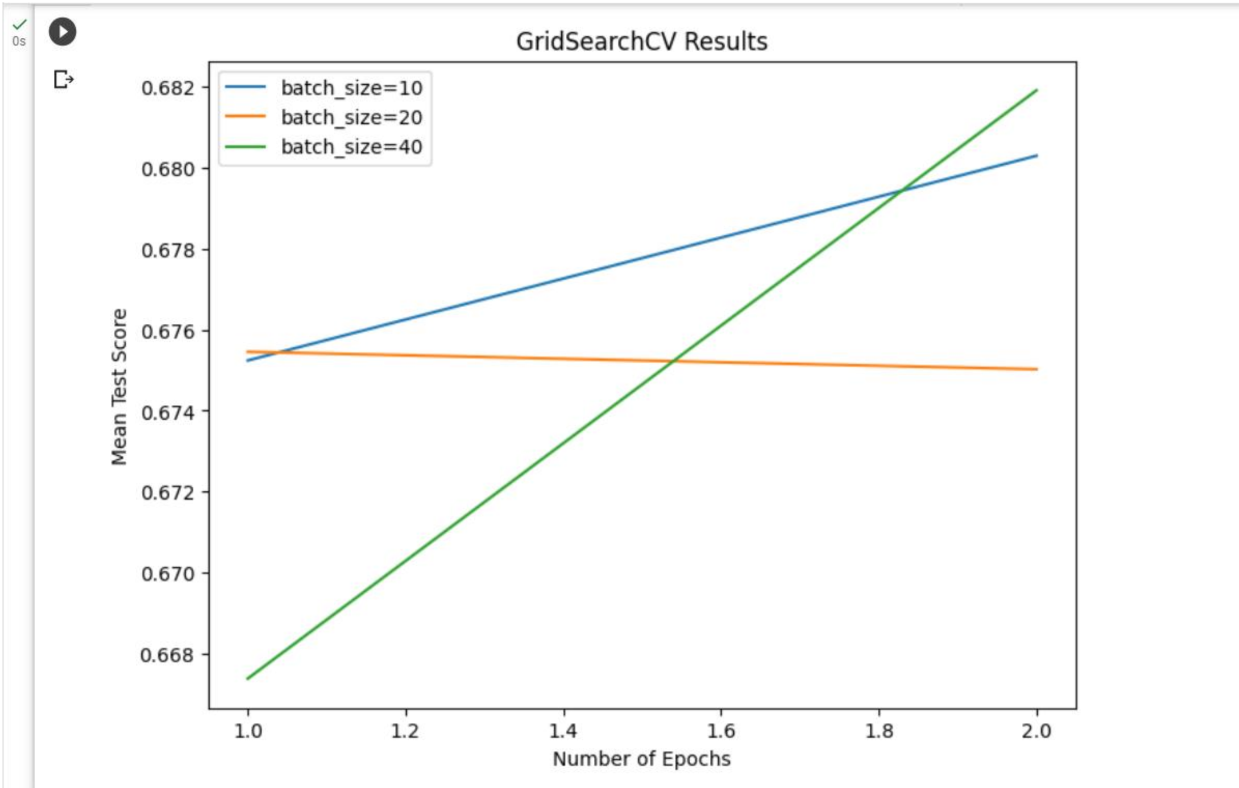
```
from keras.wrappers.scikit_learn import KerasClassifier #importing Keras classifier
from sklearn.model_selection import GridSearchCV #importing Grid search CV
```

```
[15] # Now you can proceed with the GridSearchCV
model = KerasClassifier(build_fn=createmodel, verbose=2)
batch_size = [10, 20, 40]
epochs = [1, 2]
param_grid = {'batch_size': batch_size, 'epochs': epochs}
grid = GridSearchCV(estimator=model, param_grid=param_grid)
grid_result = grid.fit(X_train, y_train)


# Print the best score and best hyperparameters found by GridSearchCV
print("Best Score: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
```

```
372/372 - 35s - loss: 0.6810 - accuracy: 0.7084 - 35s/epoch - 95ms/step
93/93 - 1s - loss: 0.7344 - accuracy: 0.6724 - 1s/epoch - 15ms/step
Epoch 1/2
372/372 - 37s - loss: 0.8332 - accuracy: 0.6379 - 37s/epoch - 100ms/step
Epoch 2/2
372/372 - 34s - loss: 0.6788 - accuracy: 0.7121 - 34s/epoch - 91ms/step
93/93 - 2s - loss: 0.7464 - accuracy: 0.6864 - 2s/epoch - 21ms/step
Epoch 1/2
372/372 - 39s - loss: 0.8298 - accuracy: 0.6377 - 39s/epoch - 106ms/step
Epoch 2/2
372/372 - 37s - loss: 0.6805 - accuracy: 0.7083 - 37s/epoch - 99ms/step
93/93 - 2s - loss: 0.7453 - accuracy: 0.6868 - 2s/epoch - 17ms/step
Epoch 1/2
372/372 - 38s - loss: 0.8261 - accuracy: 0.6461 - 38s/epoch - 101ms/step
Epoch 2/2
372/372 - 35s - loss: 0.6710 - accuracy: 0.7173 - 35s/epoch - 95ms/step
93/93 - 2s - loss: 0.7940 - accuracy: 0.6555 - 2s/epoch - 19ms/step
186/186 - 23s - loss: 0.8496 - accuracy: 0.6333 - 23s/epoch - 124ms/step
47/47 - 2s - loss: 0.7789 - accuracy: 0.6547 - 2s/epoch - 33ms/step
186/186 - 25s - loss: 0.8466 - accuracy: 0.6410 - 25s/epoch - 133ms/step
```

```
47/47 - 1s - loss: 0.7817 - accuracy: 0.6719 - 1s/epoch - 22ms/step
186/186 - 25s - loss: 0.8453 - accuracy: 0.6294 - 25s/epoch - 134ms/step
47/47 - 1s - loss: 0.8109 - accuracy: 0.6681 - 1s/epoch - 23ms/step
186/186 - 24s - loss: 0.8447 - accuracy: 0.6334 - 24s/epoch - 128ms/step
47/47 - 1s - loss: 0.7523 - accuracy: 0.6728 - 921ms/epoch - 20ms/step
186/186 - 23s - loss: 0.8409 - accuracy: 0.6416 - 23s/epoch - 121ms/step
47/47 - 1s - loss: 0.7801 - accuracy: 0.6695 - 1s/epoch - 27ms/step
Epoch 1/2
186/186 - 25s - loss: 0.8416 - accuracy: 0.6349 - 25s/epoch - 135ms/step
Epoch 2/2
186/186 - 22s - loss: 0.6997 - accuracy: 0.7039 - 22s/epoch - 116ms/step
47/47 - 1s - loss: 0.7354 - accuracy: 0.6762 - 1s/epoch - 24ms/step
Epoch 1/2
186/186 - 26s - loss: 0.8391 - accuracy: 0.6408 - 26s/epoch - 138ms/step
Epoch 2/2
186/186 - 22s - loss: 0.6920 - accuracy: 0.7035 - 22s/epoch - 118ms/step
47/47 - 1s - loss: 0.7383 - accuracy: 0.6891 - 1s/epoch - 22ms/step
Epoch 1/2
186/186 - 24s - loss: 0.8465 - accuracy: 0.6310 - 24s/epoch - 127ms/step
Epoch 2/2
186/186 - 21s - loss: 0.6895 - accuracy: 0.7037 - 21s/epoch - 114ms/step
47/47 - 1s - loss: 0.7488 - accuracy: 0.6815 - 896ms/epoch - 19ms/step
Epoch 1/2
186/186 - 23s - loss: 0.8477 - accuracy: 0.6293 - 23s/epoch - 125ms/step
Epoch 2/2
186/186 - 20s - loss: 0.6971 - accuracy: 0.6951 - 20s/epoch - 108ms/step
47/47 - 1s - loss: 0.7409 - accuracy: 0.6900 - 890ms/epoch - 19ms/step
Epoch 1/2
186/186 - 23s - loss: 0.8356 - accuracy: 0.6381 - 23s/epoch - 126ms/step
Epoch 2/2
186/186 - 22s - loss: 0.6834 - accuracy: 0.7095 - 22s/epoch - 118ms/step
47/47 - 1s - loss: 0.7752 - accuracy: 0.6728 - 964ms/epoch - 21ms/step
Epoch 1/2
233/233 - 29s - loss: 0.8382 - accuracy: 0.6411 - 29s/epoch - 124ms/step
Epoch 2/2
233/233 - 26s - loss: 0.6885 - accuracy: 0.7064 - 26s/epoch - 113ms/step
Best Score: 0.681911 using {'batch_size': 40, 'epochs': 2}
```

```python
# Plot the results of GridSearchCV
mean_scores = grid_result.cv_results_['mean_test_score']
param_batch_size = grid_result.cv_results_['param_batch_size']
param_epochs = grid_result.cv_results_['param_epochs']

plt.figure(figsize=(8, 6))
for i, batch_size in enumerate(batch_size):
    plt.plot(epochs, mean_scores[i * len(epochs): (i + 1) * len(epochs)], label=f'batch_size={batch_size}')

plt.xlabel('Number of Epochs')
plt.ylabel('Mean Test Score')
plt.title('GridSearchCV Results')
plt.legend()
plt.show()
```


GridSearchCV Results
batch_size=10
batch_size=20
batch_size=40
Mean Test Score
0.682
0.680
0.678
0.676
0.674
0.672
0.670
0.668
1.0
1.2
1.4
1.6
1.8
2.0
Number of Epochs