Fall 2023: CS5720

Neural Networks & Deep Learning - ICP-9

Name: Manisha Lakkarsu

Student Id: 700746573

Git link: https://github.com/Mani543/Manisha_NNDL_ICP9

Video link:

https://drive.google.com/file/d/1I8K0D810aKh95KiwYWdDPfPqV5VjR2i6/view?usp=sharing

In class programming:

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”)

In [2]:
```
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 tensorflow.keras.utils import to_categorical
```

In [3]:
```
# 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]

# 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)
```

In [4]:
```
# 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)
```

In [5]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

In [6]:
```
# 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)
```

Output:

```
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: 511391 (1.95 MB)
Trainable params: 511391 (1.95 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
Epoch 1/10
291/291 - 18s - loss: 0.8252 - accuracy: 0.6436 - val_loss: 0.7496 - val_accuracy: 0.6747 - 18s/epoch - 62ms/step
Epoch 2/10
```

```
291/291 - 18s - loss: 0.8252 - accuracy: 0.6436 - val_loss: 0.7496 - val_accuracy: 0.6747 - 18s/epoch - 62ms/step
Epoch 2/10
291/291 - 17s - loss: 0.6791 - accuracy: 0.7114 - val_loss: 0.7438 - val_accuracy: 0.6863 - 17s/epoch - 59ms/step
Epoch 3/10
291/291 - 17s - loss: 0.6141 - accuracy: 0.7417 - val_loss: 0.7571 - val_accuracy: 0.6824 - 17s/epoch - 59ms/step
Epoch 4/10
291/291 - 17s - loss: 0.5738 - accuracy: 0.7610 - val_loss: 0.7851 - val_accuracy: 0.6693 - 17s/epoch - 58ms/step
Epoch 5/10
291/291 - 17s - loss: 0.5290 - accuracy: 0.7779 - val_loss: 0.8194 - val_accuracy: 0.6732 - 17s/epoch - 58ms/step
Epoch 6/10
291/291 - 17s - loss: 0.4864 - accuracy: 0.8005 - val_loss: 0.8653 - val_accuracy: 0.6684 - 17s/epoch - 59ms/step
Epoch 7/10
291/291 - 17s - loss: 0.4477 - accuracy: 0.8153 - val_loss: 0.8908 - val_accuracy: 0.6586 - 17s/epoch - 59ms/step
Epoch 8/10
291/291 - 17s - loss: 0.4130 - accuracy: 0.8361 - val_loss: 1.0199 - val_accuracy: 0.6584 - 17s/epoch - 58ms/step
Epoch 9/10
291/291 - 17s - loss: 0.3818 - accuracy: 0.8401 - val_loss: 1.0483 - val_accuracy: 0.6560 - 17s/epoch - 59ms/step
Epoch 10/10
291/291 - 17s - loss: 0.3535 - accuracy: 0.8544 - val_loss: 1.2282 - val_accuracy: 0.6566 - 17s/epoch - 58ms/step
```

- Evaluating the model on test data and plotting accuracy and loss graphs

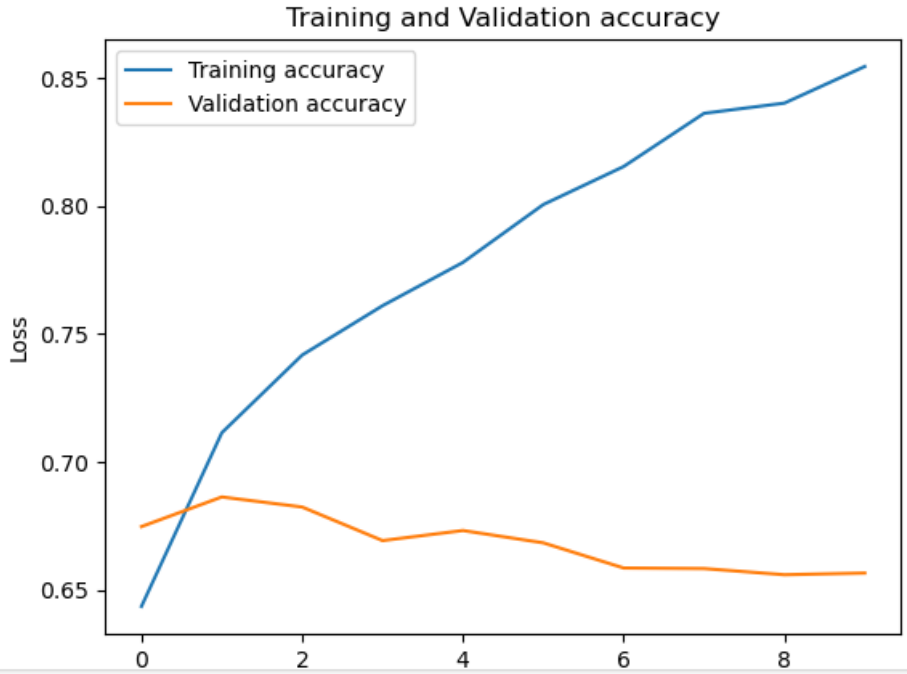
In [7]:

```
# 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 - 1s - loss: 1.2282 - accuracy: 0.6566 - 1s/epoch - 8ms/step
Test Loss: 1.2282441854476929
Test Accuracy: 0.656618595123291
```



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

In [8]:
```
# Save the trained model
model.save('sentimentAnalysis.keras')
```

In [9]:
```
from keras.models import load_model
model = load_model('sentimentAnalysis.keras')
```

In [10]:
```
# 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. @rea

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

In [11]:
```
# 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 - 222ms/epoch - 222ms/step
Neutral
```

In [12]:
```
# Apply GridSearchCV on the source code provided in the class
from scikeras.wrappers import KerasClassifier #importing Keras classifier
from sklearn.model_selection import GridSearchCV #importing Grid search CV
```

In [*]:
```
# 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_))
```

Output:

```
print("Best Score: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
```

```
C:\Users\manis\anaconda3\lib\site-packages\scikeras\wrappers.py:915: UserWarning: ``build_fn`` will be renamed to ``model`
` in a future release, at which point use of ``build_fn`` will raise an Error instead.
  X, y = self._initialize(X, y)
744/744 - 30s - loss: 0.8219 - accuracy: 0.6538 - 30s/epoch - 41ms/step
186/186 - 1s - 1s/epoch - 7ms/step
C:\Users\manis\anaconda3\lib\site-packages\scikeras\wrappers.py:915: UserWarning: ``build_fn`` will be renamed to ``model`
` in a future release, at which point use of ``build_fn`` will raise an Error instead.
  X, y = self._initialize(X, y)
744/744 - 35s - loss: 0.8195 - accuracy: 0.6462 - 35s/epoch - 47ms/step
186/186 - 2s - 2s/epoch - 9ms/step
C:\Users\manis\anaconda3\lib\site-packages\scikeras\wrappers.py:915: UserWarning: ``build_fn`` will be renamed to ``model`
` in a future release, at which point use of ``build_fn`` will raise an Error instead.
  X, y = self._initialize(X, y)
744/744 - 32s - loss: 0.8239 - accuracy: 0.6451 - 32s/epoch - 44ms/step
186/186 - 1s - 1s/epoch - 8ms/step
```

```
C:\Users\manis\anaconda3\lib\site-packages\scikeras\wrappers.py:915: UserWarning: ``build_fn`` will be renamed to ``model`
` in a future release, at which point use of ``build_fn`` will raise an Error instead.
  X, y = self._initialize(X, y)
Epoch 1/2
186/186 - 45s - loss: 0.8373 - accuracy: 0.6399 - 45s/epoch - 244ms/step
Epoch 2/2
186/186 - 37s - loss: 0.6831 - accuracy: 0.7080 - 37s/epoch - 201ms/step
47/47 - 5s - 5s/epoch - 111ms/step
C:\Users\manis\anaconda3\lib\site-packages\scikeras\wrappers.py:915: UserWarning: ``build_fn`` will be renamed to ``model`
` in a future release, at which point use of ``build_fn`` will raise an Error instead.
  X, y = self._initialize(X, y)
Epoch 1/2
233/233 - 42s - loss: 0.8345 - accuracy: 0.6422 - 42s/epoch - 182ms/step
Epoch 2/2
233/233 - 37s - loss: 0.6830 - accuracy: 0.7086 - 37s/epoch - 157ms/step
Best Score: 0.683848 using {'batch_size': 40, 'epochs': 2}
```

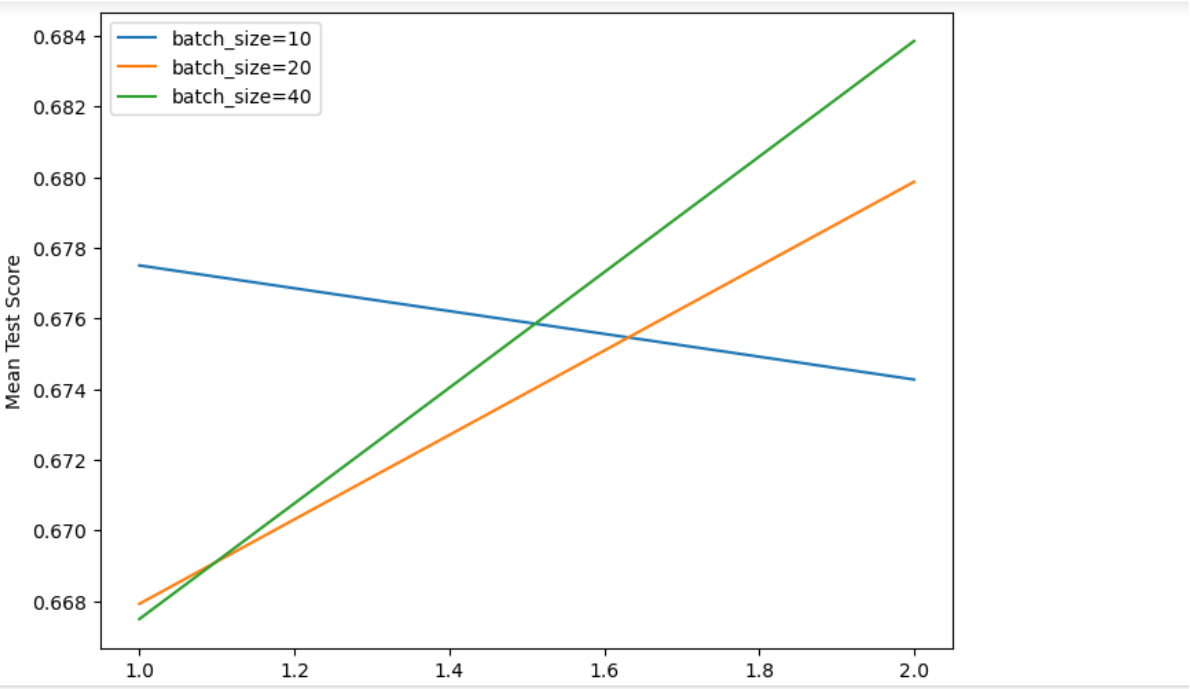
- Evaluating GridSearchCV

In [ ]:

```
# 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()
```


batch_size=10
batch_size=20
batch_size=40
Mean Test Score
0.684
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