Saemi Ramirez

011926418

D213 PA 2

Sentiment Analysis Using Neural Networks

10/25/2024

WGU

A. Purpose of this data analysis

1. Research Question

   Can we predict the customer's sentiment based on their comments using the neural network model and NLP?

2. Define objectives or goals

   The goal is to build the neural network model and NLP and to analyze and predict if the comment was either positive or negative.

3. Identify a type of neural network capable of performing a text classification task that can be trained to produce useful predictions on text sequences on the selected data set

   The LSTM (Long Short-Term Memory network) which is a type of a recurrent neural network is used for this analysis (Brownlee)

B. Summarize assumptions of a time series modeling including stationarity and autocorrelated data

1. Perform exploratory data analysis on the chosen data set

   i. Presence of unusual characters – regex was used to identify any non-english characters in the review column

```
for description in df['Reviews']:
    #Remove special characters from string using regex
    description = re.sub('[^a-zA-Z0-9]', ' ', description)
```

   ii. Vocabulary size

   The vocabulary size is 4284. First, the 'Frequent' column was created after cleaning up and filtering the 'Review' column. Tokenizer() and fit_on_texts functions were used then counted the length of the word with word_index + 1

```
#Identify vocabulary size
tokenizer = Tokenizer()
tokenizer.fit_on_texts(frequent_list)
vocab_size = len(tokenizer.word_index)+1
print("vocabulary size: ", vocab_size)
```

```
vocabulary size:  4284
```

   iii. Proposed word embedding length

   Embedding length was calculated as 8 using the np.sqrt function with already calculated vocab_size (4284)

```
#Identify the embedding size
max_sequence_embedding = int(round(np.sqrt(np.sqrt(vocab_size)), 0))
print('Max sequence embedding: ', max_sequence_embedding)
```

```
Max sequence embedding:  8
```

iv. Statistical justification for the chosen max sequence length
The max sequence length is 3971

```
#Determine min, med, and max legnths of reviews
review_length = []
for index, row in df.iterrows():
    review_length.append(len(entire_string(row, 'Frequent')))

review_max = int(np.max(review_length))
review_min = int(np.min(review_length))
review_median = int(np.median(review_length))
print('Max length of the sequence is: ', review_max)
print('Min length of the sequence is: ', review_min)
print('Median length of the sequence is: ', review_median)
```

```
Max length of the sequence is:  3971
Min length of the sequence is:  0
Median length of the sequence is:  27
```

2. Describe the goals of the tokenization process, including any code generated and packages that are used to normalize text during the tokenization process
    i. Nltk.word_tokenizer() was used to break down customer's reviews into single words

```
#Download packages for stopwords, punkt, and wordnet
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')

tokenized_list=[]
cleaned_list=[]
frequent_list=[]
frequent_list_tokenized=[]
stop_words = stopwords.words('english')

for description in df['Reviews']:
    #Remove special characters from string using regex
    description = re.sub('[^a-zA-Z0-9]', ' ', description)
    #Convert to lower case
    description_lower = description.lower()
    #Perform tokenization
    description_tokenized = nltk.word_tokenize(description_lower)
    #Perform lemmatization
    lemma = nltk.WordNetLemmatizer()
    description_lemma = [lemma.lemmatize(word) for word in description_tokenized]
    tokenized_list.append(description_lemma)
    #Removing stopwords
    description_no_stopwords = [word for word in description_lemma if not word in stop_words]
    description_no_stopwords = " ".join(description_no_stopwords)
    description_tokenized2 = nltk.word_tokenize(description_no_stopwords)
    cleaned_list.append(description_tokenized2)
    #Removing infrequent words
    description_freq = [word for word in description_tokenized2 if len(word)>3]
```

```
    description_tokenized3 = " ".join(description_freq)
    frequent_list.append(description_tokenized3)
    description_tokenized3 = nltk.word_tokenize(description_tokenized3)
    frequent_list_tokenized.append(description_tokenized3)


#Add the tokenized_list and cleaned_list to the dataframe
df['Tokenized'] = pd.DataFrame(zip(tokenized_list))
df['No_Stopwords'] = pd.DataFrame(zip(cleaned_list))
df['Frequent'] = pd.DataFrame(frequent_list)
df.head(10)
```

	Reviews	Label	Source	Tokenized	No_Stopwords	Frequent
0	So there is no way for me to plug it in here i...	0	Amazon	[so, there, is, no, way, for, me, to, plug, it...	[way, plug, u, unless, go, converter]	plug unless converter
1	Good case, Excellent value.	1	Amazon	[good, case, excellent, value]	[good, case, excellent, value]	good case excellent value
2	Great for the jawbone.	1	Amazon	[great, for, the, jawbone]	[great, jawbone]	great jawbone
3	Tied to charger for conversations lasting more...	0	Amazon	[tied, to, charger, for, conversation, lasting...	[tied, charger, conversation, lasting, 45, min...	tied charger conversation lasting minute major...
4	The mic is great.	1	Amazon	[the, mic, is, great]	[mic, great]	great
5	I have to jiggle the plug to get it to line up...	0	Amazon	[i, have, to, jiggle, the, plug, to, get, it, ...	[jiggle, plug, get, line, right, get, decent, ...	jiggle plug line right decent volume
6	If you have several dozen or several hundred c...	0	Amazon	[if, you, have, several, dozen, or, several, h...	[several, dozen, several, hundred, contact, im...	several dozen several hundred contact imagine ...
7	If you are Razr owner...you must have this!	1	Amazon	[if, you, are, razr, owner, you, must, have, t...	[razr, owner, must]	razr owner must
8	Needless to say, I wasted my money.	0	Amazon	[needle, to, say, i, wasted, my, money]	[needle, say, wasted, money]	needle wasted money
9	What a waste of money and time!.	0	Amazon	[what, a, waste, of, money, and, time]	[waste, money, time]	waste money time

3. Explain the padding process
    i. If the padding occurs before or after the text sequence

    The pad_sequences was applied to the both X_train and X_test sets after the texts_to_sequences

    ii. Screenshot of a single padded sequence

```
#Apply padding to training data
review_train = tokenizer.texts_to_sequences(X_train)
padded_train = pad_sequences(review_train, maxlen=review_median, padding='post', truncating='post')

#Apply padding to test data
review_test = tokenizer.texts_to_sequences(X_test)
padded_test = pad_sequences(review_test, maxlen=review_median, padding='post', truncating='post')

#Display added sequences
np.set_printoptions(threshold=sys.maxsize)
padded_train[1]
```

```
array([  1,  46,  27, 267,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0])
```

4. Identify how many categories of sentiment will be used and an activation function for the final dense layer of the network

    Two categories of sentiment which are positive or negative (0 or 1) will be used with ‘softmax’ as an activation function for the final dense layer of the network

5. Explain the steps used to prepare the data for analysis including the size of the training, validation, and test set split

i. Imported the data (Amazon, IMDB, and Yelp), added a third column (Source) and concatenated 3 data sources

```
#Import the data from Amazon, IMDB, and Yelp
amazon = pd.read_csv(r'C:\Users\Saemi\OneDrive\Educatior
amazon['Source'] = 'Amazon'
imdb = pd.read_csv(r'C:\Users\Saemi\OneDrive\Education\w
imdb['Source'] = 'IMDB'
yelp = pd.read_csv(r'C:\Users\Saemi\OneDrive\Education\w
yelp['Source'] = 'Yelp'
df = pd.concat([amazon, imdb, yelp], ignore_index=True)
df.columns = ['Reviews', 'Sentiment', 'Source']
df.Sentiment = df.Sentiment.astype(int)
```

ii. Used the regex, lower(), WordNetLemmatizer(), stopwords, remove infrequent words in one for loop

```
#Download packages for stopwords, punkt, and wordnet
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')

tokenized_list=[]
cleaned_list=[]
frequent_list=[]
frequent_list_tokenized=[]
stop_words = stopwords.words('english')

for description in df['Reviews']:
    #Remove special characters from string using regex
    description = re.sub('[^a-zA-Z0-9]', ' ', description)
    #Convert to lower case
    description_lower = description.lower()
    #Perform tokenization
    description_tokenized = nltk.word_tokenize(description_lower)
    #Perform lemmatization
    lemma = nltk.WordNetLemmatizer()
    description_lemma = [lemma.lemmatize(word) for word in description_tokenized]
    tokenized_list.append(description_lemma)
    #Removing stopwords
    description_no_stopwords = [word for word in description_lemma if not word in stop_words]
    description_no_stopwords = " ".join(description_no_stopwords)
    description_tokenized2 = nltk.word_tokenize(description_no_stopwords)
    cleaned_list.append(description_tokenized2)
```

```
    #Removing infrequent words
    description_freq = [word for word in description_tokenized2 if len(word)>3]
    description_tokenized3 = " ".join(description_freq)
    frequent_list.append(description_tokenized3)
    description_tokenized3 = nltk.word_tokenize(description_tokenized3)
    frequent_list_tokenized.append(description_tokenized3)


#Add the tokenized_list and cleaned_list to the dataframe
df['Tokenized'] = pd.DataFrame(zip(tokenized_list))
df['No_Stopwords'] = pd.DataFrame(zip(cleaned_list))
df['Frequent'] = pd.DataFrame(frequent_list)
df.head(10)
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Saemi\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Saemi\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\Saemi\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

	Reviews	Sentiment	Source	Tokenized	No_Stopwords	Frequent
0	So there is no way for me to plug it in here i...	0	Amazon	[so, there, is, no, way, for, me, to, plug, it...	[way, plug, u, unless, go, converter]	plug unless converter
1	Good case, Excellent value.	1	Amazon	[good, case, excellent, value]	[good, case, excellent, value]	good case excellent value
2	Great for the jawbone.	1	Amazon	[great, for, the, jawbone]	[great, jawbone]	great jawbone
3	Tied to charger for conversations lasting more...	0	Amazon	[tied, to, charger, for, conversation, lasting...	[tied, charger, conversation, lasting, 45, min...	tied charger conversation lasting minute major...
4	The mic is great.	1	Amazon	[the, mic, is, great]	[mic, great]	great
5	I have to jiggle the plug to get it to line up...	0	Amazon	[i, have, to, jiggle, the, plug, to, get, it, ...	[jiggle, plug, get, line, right, get, decent, ...	jiggle plug line right decent volume
6	If you have several dozen or several hundred c...	0	Amazon	[if, you, have, several, dozen, or, several, h...	[several, dozen, several, hundred, contact, im...	several dozen several hundred contact imagine ...
7	If you are Razr owner...you must have this!	1	Amazon	[if, you, are, razr, owner, you, must, have, t...	[razr, owner, must]	razr owner must
8	Needless to say, I wasted my money.	0	Amazon	[needle, to, say, i, wasted, my, money]	[needle, say, wasted, money]	needle wasted money
9	What a waste of money and time!.	0	Amazon	[what, a, waste, of, money, and, time]	[waste, money, time]	waste money time

iii. Saved the DataFrame to csv file

```
df.to_csv('cleaned_data.csv')
```

iv. Created a function to make an one long string out of the ‘Frequent’ column

```
#Make long string out of entire Frequent column
def entire_string (df, column):
    entire = (' '.join(word for word in df[column]))
    entire_tokenized = nltk.word_tokenize(entire)
    return entire_tokenized
```

v. Calculated max, min, and median lengths of the result of entire_string function with ‘Frequent’ column

```
#Determine min, med, and max legnths of reviews
review_length = []
for index, row in df.iterrows():
    review_length.append(len(entire_string(row, 'Frequent')))

review_max = int(np.max(review_length))
review_min = int(np.min(review_length))
review_median = int(np.median(review_length))
print('Max length of the sequence is: ', review_max)
print('Min length of the sequence is: ', review_min)
print('Median length of the sequence is: ', review_median)
```

```
Max length of the sequence is:  3971
Min length of the sequence is:  0
Median length of the sequence is:  27
```

vi. Split the data to train and test sets

```
#Split the data to train and test sets
split = round(len(df)*0.8)

X_train = df['Frequent'][:split]
X_test = df['Frequent'][split:]
y_train = df['Sentiment'][:split]
y_test = df['Sentiment'][split:]
```

vii. Padded on both of the train and the test sets using pad_sequences after applying texts_to_sequences() function

```
paddnig='post'
truncating='post'

#Apply padding to training data
review_train = tokenizer.texts_to_sequences(X_train)
padded_train = pad_sequences(review_train, maxlen=review_median, padding=paddnig, truncating=truncating)

#Apply padding to test data
review_test = tokenizer.texts_to_sequences(X_test)
padded_test = pad_sequences(review_test, maxlen=review_median, padding=paddnig, truncating=truncating)

#Display added sequences
np.set_printoptions(threshold=sys.maxsize)
padded_train[1]
```

```
array([  1,  46,  27, 267,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0])
```

viii. Converted the padded data to NumPy array to be used in the model

```
#Convert padded data to numpy array to be used in the model
training_padded = np.array(padded_train)
training_label = np.array(y_train)
test_padded = np.array(padded_test)
test_label = np.array(y_test)
```

ix. Exported the NumPy arrayed sets to csv files

```
#Export the data to csv file
pd.DataFrame(training_padded).to_csv('training_padded.csv')
pd.DataFrame(training_label).to_csv('training_label.csv')
pd.DataFrame(test_padded).to_csv('test_padded.csv')
pd.DataFrame(test_label).to_csv('test_label.csv')
```

6. Provide a copy of the prepared dataset ‘cleaned_data.csv’, ‘test_label.csv’, ‘test_padded.csv’, ‘training_label.csv’, and ‘training_padded.csv’ are submitted

C. Describe the type of network used

1. Provide the output of the model summary of the function from TensorFlow

```
#Create EarlyStopping
early_stopping_monitor = tf.keras.callbacks.EarlyStopping(patience=2)
num_epochs=20

#Build a neural network model
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, max_sequence_embedding, input_length=review_max),
    tf.keras.layers.GlobalAveragePooling1D(),
    tf.keras.layers.Dense(100, activation='relu'),
    tf.keras.layers.Dense(50, activation='relu'),
    tf.keras.layers.Dense(2, activation='softmax')
])

model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
#Print model summary
#model.summary()

history = model.fit(training_padded, training_label, batch_size=32, epochs=num_epochs,
                    validation_split=0.3, callbacks=[early_stopping_monitor], verbose=True)
model.summary()
```

```
Epoch 1/20
C:\Users\Saemi\anaconda3\Lib\site-packages\keras\src\layers\core\embedding.py:90: UserWarning: Argument `input_length` is deprecated. Just remove it.
  warnings.warn(
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step - accuracy: 0.5119 - loss: 0.6933 - val_accuracy: 0.3970 - val_loss: 0.7026
Epoch 2/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.5268 - loss: 0.6913 - val_accuracy: 0.3970 - val_loss: 0.6996
Epoch 3/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.5127 - loss: 0.6872 - val_accuracy: 0.4030 - val_loss: 0.7218
Epoch 4/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6608 - loss: 0.6445 - val_accuracy: 0.7030 - val_loss: 0.6133
Epoch 5/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6845 - loss: 0.5847 - val_accuracy: 0.7348 - val_loss: 0.5452
Epoch 6/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8515 - loss: 0.3753 - val_accuracy: 0.7091 - val_loss: 0.5776
Epoch 7/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8640 - loss: 0.2972 - val_accuracy: 0.7409 - val_loss: 0.5276
Epoch 8/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9320 - loss: 0.1940 - val_accuracy: 0.6939 - val_loss: 0.6261
Epoch 9/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9266 - loss: 0.1675 - val_accuracy: 0.7015 - val_loss: 0.6051
```

Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 27, 8)	34,272
global_average_pooling1d (GlobalAveragePooling1D)	(None, 8)	0
dense (Dense)	(None, 100)	900
dense_1 (Dense)	(None, 50)	5,050
dense_2 (Dense)	(None, 2)	102

Total params: 120,974 (472.56 KB)

Trainable params: 40,324 (157.52 KB)

Non-trainable params: 0 (0.00 B)

Optimizer params: 80,650 (315.04 KB)

2. Discuss the number of layers, the type of layers, and the total number of parameters
   Total 3 different types of layers were used which are Embedding, GlobalAveragePooling1D, and the Dense.
   Embedding layer is a method used to represent words as numerical vectors which means it transforms text into numbers. (Medewar)
   GlobalAveragePooling1D outputs a matrix of batch x embedding_size by averaging across the sequence dimension. “1D” refers to the fact that the averaging occurs over a single dimension. This process can handle varying sequence lengths. (Lathan)
   Dense layers are essential components of neural networks, composed of neurons that are each connected to every neuron in the preceding layer. The term “dense” indicates that each neuron has a connection to all neurons in the previous layer. (Keras)
   Out of numbers of parameters that the Embedding function has, 3 parameters, the input_dim for the vocabulary size, the output_dim for the embedding size, and the input_length for the review_max were used for this analysis.

```
#Create EarlyStopping
early_stopping_monitor = tf.keras.callbacks.EarlyStopping(patience=2)
num_epochs=20

#Build a neural network model
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, max_sequence_embedding, input_length=review_max),
    tf.keras.layers.GlobalAveragePooling1D(),
    tf.keras.layers.Dense(100, activation='relu'),
    tf.keras.layers.Dense(50, activation='relu'),
    tf.keras.layers.Dense(2, activation='softmax')
])

model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
#Print model summary
#model.summary()

history = model.fit(training_padded, training_label, batch_size=32, epochs=num_epochs,
                    validation_split=0.3, callbacks=[early_stopping_monitor], verbose=True)
model.summary()
```

3. Justify the choice of hyperparameters
    i. Activation functions – 'softmax' was used for the activation parameter for the dense function to "convert(s) a vector of values to a probability distribution" (Layer)
    ii. Number of nodes per layer

Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 27, 8)	34,272
global_average_pooling1d (GlobalAveragePooling1D)	(None, 8)	0
dense (Dense)	(None, 100)	900
dense_1 (Dense)	(None, 50)	5,050
dense_2 (Dense)	(None, 2)	102

Total params: 120,974 (472.56 KB)

Trainable params: 40,324 (157.52 KB)

Non-trainable params: 0 (0.00 B)

Optimizer params: 80,650 (315.04 KB)

    iii. Loss function – 'sparse_categorical_crossentropy', which "computes the crossentropy loss between the labels and predictions" (tf.keras.losses.SparseCategorical.Crossentropy), was used for the loss in the compile function for the model
    iv. Optimizer – 'adam' was used for the optimizer in the compile function for the model. Optimizer 'adam' is a stochastic gradient descent technique that relies on adaptive estimation of both first-order and second-order moments (Adam)
    v. Stopping criteria – EarlyStopping function with patience set to 2 was utilized for the callbacks parameter in the model.fit function. EarlyStopping callback allows

monitoring a metric and halts training if no improvement is detected. (Early) Epochs was set to 20 as a starting point

vi. Evaluation metric – 'Accuracy' was used for the metric in the compile function for the model. Accuracy "calculates how often predictions equal labels" (Accuracy).

D. Evaluate the model training process and its relevant outcomes

1. Discuss the impact of using stopping criteria to include defining the number of epochs including a screenshot showing the final training epoch

   Stopping criteria stops the model training when there is no more to improve on the model. This helps preventing overfitting by halting the training process early, before the model becomes overly complex and starts memorizing the training data (Miseta) As the screenshot shown below, the model stopped training after the 9th iteration with the accuracy of 92.66%

```
early_stopping_monitor = tf.keras.callbacks.EarlyStopping(patience=2)
num epochs=20
history = model.fit(training_padded, training_label, batch_size=32, epochs=num_epochs,
                    validation_split=0.3, callbacks=[early_stopping_monitor], verbose=True)
model.summary()
```

```
Epoch 1/20
C:\Users\Saemi\anaconda3\Lib\site-packages\keras\src\layers\core\embedding.py:90: UserWarning: Argument `input_length` is depre
cated. Just remove it.
  warnings.warn(
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step - accuracy: 0.5119 - loss: 0.6933 - val_accuracy: 0.3970 - val_loss: 0.7026
Epoch 2/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.5268 - loss: 0.6913 - val_accuracy: 0.3970 - val_loss: 0.6996
Epoch 3/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.5127 - loss: 0.6872 - val_accuracy: 0.4030 - val_loss: 0.7218
Epoch 4/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6608 - loss: 0.6445 - val_accuracy: 0.7030 - val_loss: 0.6133
Epoch 5/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6845 - loss: 0.5847 - val_accuracy: 0.7348 - val_loss: 0.5452
Epoch 6/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8515 - loss: 0.3753 - val_accuracy: 0.7091 - val_loss: 0.5776
Epoch 7/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8640 - loss: 0.2972 - val_accuracy: 0.7409 - val_loss: 0.5276
Epoch 8/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9320 - loss: 0.1940 - val_accuracy: 0.6939 - val_loss: 0.6261
Epoch 9/20
49/49 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9266 - loss: 0.1675 - val_accuracy: 0.7015 - val_loss: 0.6051
```

2. Assess the fitness of the model and any actions taken to address overfitting

   As the screenshot above, the model stopped training further after 9th iteration to prevent overfitting. Also EarlyStopping function was added on the callbacks parameter to avoid overfitting on the model.

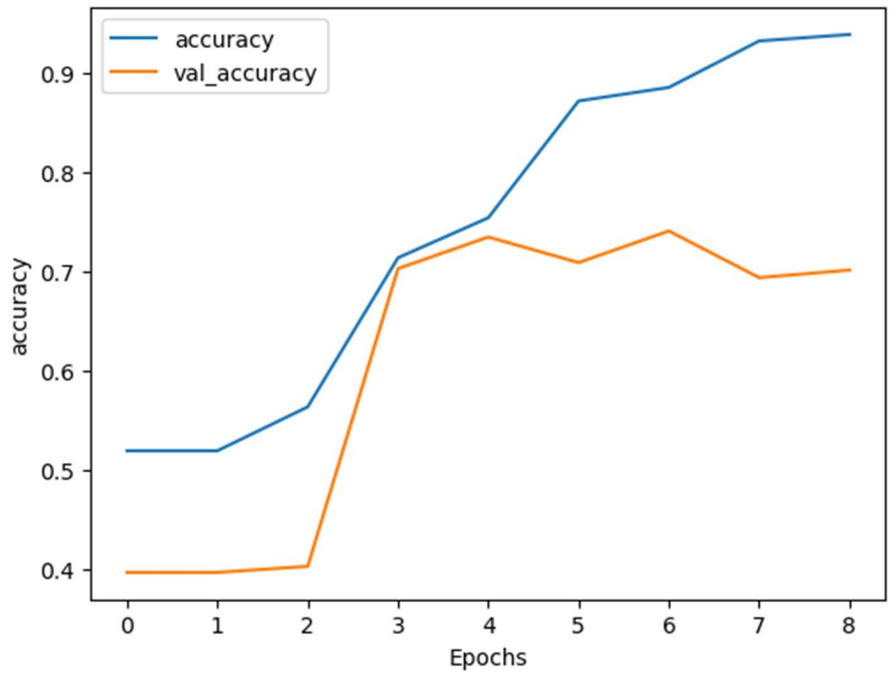
   The training model has the accuracy of approximately 88.54% on this analysis.

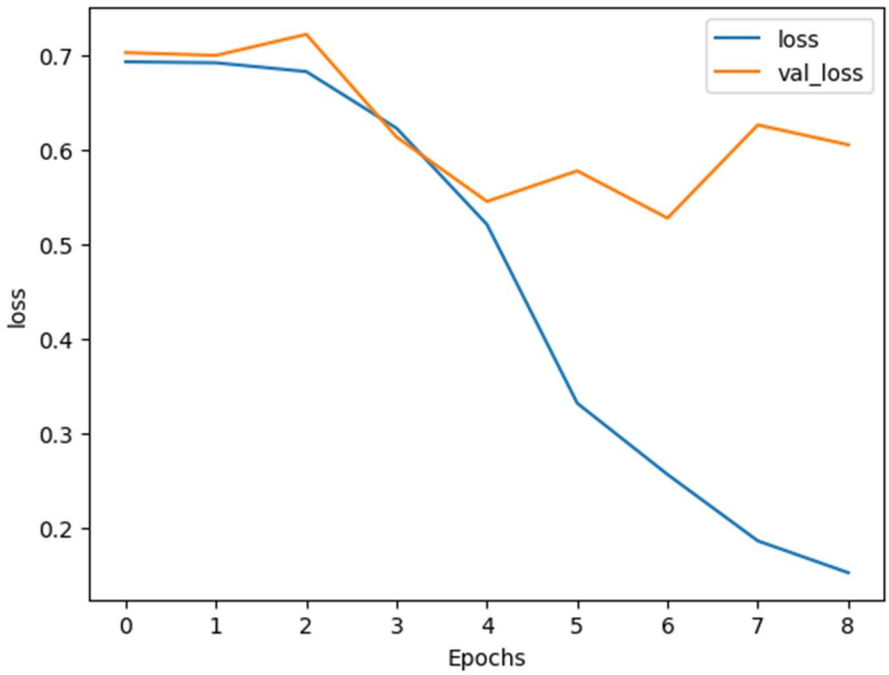
```
#Verify model accuracy on training data
score = model.evaluate(training_padded, training_label, verbose=0)
print(f'Training loss: {score[0]} / Training accuracy: {score[1]}')
```

```
Training loss: 0.2636036276817322 / Training accuracy: 0.8853503465652466
```

3. Provide visualizations of the model's training process including a line graph of the loss and chosen evaluation metric

```
#Graph training and validation accuracy & loss
def plot_graphs(history, string):
    plt.plot(history.history[string])
    plt.plot(history.history['val_'+string])
    plt.xlabel("Epochs")
    plt.ylabel(string)
    plt.legend([string, 'val_'+string])
    plt.show()
plot_graphs(history, "accuracy")
plot_graphs(history, "loss")
```





4. Discuss the predictive accuracy of the trained network using the chosen evaluation metric
   The accuracy on the test model is approximately 71.64%

```
#Verify model accuracy on test data
score = model.evaluate(test_padded, test_label, verbose=0)
print(f'Test loss: {score[0]} / Test accuracy: {score[1]}')
```

```
Test loss: 0.6452460289001465 / Test accuracy: 0.7163636088371277
```

E. Provide the code
   'Saemi Ramirez D213 PA2 – Sentiment Analysis Using Neural Networks.ipynb' and 'my_model.keras' are submitted

```
#Save and reloading the model to perform predictions
#Save the model
model.save('my_model.keras')

#Load the model for performing predictions
my_model = load_model('my_model.keras')
```

F. Discuss the functionality of your neural network including the impact of the network architecture
   The neural network model was developed to classify sentiments either positive or negative. Model evaluation showed an accuracy of 88.5% on the training set and 71.6% on the test set. Overfitting was effectively mitigated by implementing an EarlyStopping function and limited training to the designated number of epochs, thus reducing the risk of memorization of the training data.
   The model architecture comprises five layers, including one Embedding layer, one GlobalAveragePooling1D, and three Dense layers. The first Dense layer contains 100 neurons with 900 parameters using relu activation. The second Dense layer consists of 50 neurons with 5050 parameters and employes relu activation. The final Dense layer has 2 neurons with 102 parameters using softmax activation.

G. Recommend a course of action
   Bigger size of samples is recommended for the better prediction rate for the test set.
   I also recommend this model can be used for the gaming reviews such as Steam by analyzing the user's reviews. It can be helpful for the players who play the game, but it will be more beneficial for the developers to analyze their weaknesses and apply it on their next game.

H. Show your neural network in an industry-relevant interactive development environment
   'Saemi Ramirez (011926418) D213 PA 2 Sentiment Analysis Using Neural Networks.pdf' is submitted

I. List 3rd party code
   1. DS Archives. *Part – 10 How to use FreqDist in NLTK with python (Natural Language Toolkit Tutorial)*. YouTube. (September 26, 2020). https://www.youtube.com/watch?v=1K2ZQ5MYdnM.
   2. *Implementing and Analyzing N-Grams in Python*. Open Library. https://ecampusontario.pressbooks.pub/nudh3/chapter/implementing-and-analyzing-n-grams-in-python.

3. Mark. *How to convert list of lists into list – python.* Stack overflow. (December 1, 2019). https://stackoverflow.com/questions/59130959/how-to-convert-list-of-lists-into-list-python.
4. Nithyashree. *What Are N-Grams and How to Implement Them in Python?* Analytics Vidhya. (September 17, 2024). https://www.analyticsvidhya.com/blog/2021/09/what-are-n-grams-and-how-to-implement-them-in-python.
5. *Seaborn.*countplot. Seaborn. https://seaborn.pydata.org/generated/seaborn.countplot.html.
6. Tim_xyz. *How to install stop-words package for Anaconda*. StackOverflow. (January 22, 2018). https://stackoverflow.com/questions/48385829/how-to-install-stop-words-package-for-anaconda.

J. List 3rd party text citations or references

1. *Accuracy metrics*. Keras. https://keras.io/api/metrics/accuracy_metrics.
2. *Adam*. Keras. https://keras.io/api/optimizers/adam.
3. Brownlee, Jason. *Time Series Prediction with LSTM Recurrent Neural Networks in Python with Keras*. Machine Learning Mastery. (August 7, 2022). https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/.
4. *Early Stopping*. Lightning AI. https://lightning.ai/docs/pytorch/stable/common/early_stopping.html.
5. *Keras dense layer*. Educative. (July 24, 2023). https://www.educative.io/answers/keras-dense-layer.
6. Lathan. *What does GlobalAveragePooling1D do in keras?* Stack Overflow. (July 29, 2023). https://stackoverflow.com/questions/75067335/what-does-globalaveragepooling1d-do-in-keras.
7. *Layer activation functions*. Keras. https://keras.io/api/layers/activations.
8. Medewar, Soham. *Embedding Layers in Keras*. Code360. (March 27, 2024). https://www.naukri.com/code360/library/embedding-layers-in-keras.
9. Miseta, Tamas. *Surpassing early stopping: A novel correlation-based stopping criterion for neural networks*. ScienceDirect. (January 28, 2024). https://www.sciencedirect.com/science/article/pii/S0925231223011517.
10. *Tf.keras.losses.SparseCategoricalCrossentropy*. TensorFlow. https://www.tensorflow.org/api_docs/python/tf/keras/losses/SparseCategoricalCrossentropy.

K. Demonstrate professional communication in the content and presentation of your submission Panopto Link: https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=d7438a04-61c3-4125-b2fe-b214013ba01b