WGU D213 Task2

August 21, 2024

[2958]:
```
%%html
<style>
.toc-item > li {
    list-style-type: upper-alpha;
}
</style>
```

```
<IPython.core.display.HTML object>
```

0.1 Kamal Shaham

0.2 D213 Task 2: Sentiment Analysis Using Neural Networks

```
<h2>Table of Contents</h2>
<ul class="toc-item">
    <li><a href="#research-question">Research Question</a>
        <ul>
            <li><a href="#goals">Analysis Goals</a></li>
            <li><a href="#nn">Neural Network</a></li>
        </ul>
    </li>
     <li><a href="#exploration">Data Exploration</a>
        <ul>
            <li><a href="#tokenization">Tokenization</a></li>
            <li><a href="#padding">Padding</a></li>
            <li><a href="#sentiment">Sentiment Categories</a></li>
            <li><a href="#steps">Data Preparation Steps</a></li>
            <li><a href="#dataset">Prepared Dataset</a></li>
        </ul>
    </li>
     <li><a href="#arch">Network Architecture</a>
    <ul>
            <li><a href="#layers">Layers</a></li>
            <li><a href="#hyperparameters">Hyperparameters</a></li>
        </ul>
    </li>
    <li><a href="#evaluation">Model Evaluation</a>
    <ul>
            <li><a href="#fitness">Model Fitness</a></li>
```

```
            <li><a href="#visualizations">Training Visualizations</a></li>
            <li><a href="#predictive">Predictive Accuracy</a></li>
        </ul>
    </li>
    <li><a href="#model">Model Code</a>
    <li><a href="#functionality">Model Functionality</a>
    <li><a href="#action">Course of Action</a>
    <li><a href="#thirdparty">Third-party code references</a></li>
    <li><a href="#references">References</a></li>
</ul>
```

0.3 A. Research Question

The research question for this analysis is: Can a neural network model effectively predict the sentiment of reviews in the IMDB subset from the UCI datasets?

0.3.1 A2. Analysis Goals

The main goals of this analysis are outlined below:

- Develop a predictive model that analyzes IMDB movie reviews to determine whether the model can accurately classify reviewer opinions as either positive or negative based on historical review data. This involves training the model on a labeled dataset of past reviews to learn the patterns associated with sentiment.
- Utilize model evaluation metrics to understand how well the model is performing. This evaluation will also help identify whether further adjustments or refinements to the model are necessary.
- Recommend a course of action for model improvements or for use in a production environment.

0.3.2 A3. Neural Network

The type of neural network that will be used in this analysis is a Recurrent Neural Network (RNN). It is a deep neural network that can be trained on sequential or time series data to then create a machine learning model. RNNs are capable of solving ordinal or temporal problems such as language translation or natural language processing (What Is a Recurrent Neural Network (RNN)? | IBM, n.d.). Given natural language processing is one of RNNs capabilities, it will be used to answer the research question mentioned in section one.

0.4 B. Data Exploration

The dataset will be imported and the column names will be set, it will be checked for duplicate and null values. The dataset description, information, and visualizations will be displayed below. The following steps will also be preformed:

- Unusual Characters: The dataset will be checked for unusual characters using a function to encode characters to utf-8 and then decode with ascii. If the characters cannot be encoded and decoded, it will add the character to a set of non_english_chars. This will handle checking for any emoji in the dataset as they cannot be encoded in ascii.
- Vocabulary Size: After removal of stopwords the vocabulary size will be checked. The code below displays a vocabulary size of 2862.

- Proposed Word Embedding Length: The code below displays a proposed word embedding length of 7. This is because the embedding vector dimension should be the 4th root of the vocabulary size (Introducing TensorFlow Feature Columns, 2017). The vocabulary size is 2862 and the fourth root rounded to the nearest whole number is 7.
- Statistical justification for maximum sequence length: The maximum sequence length was calculated by taking the row with the longest length of words from the reviews column (Medium, 2020). The maximum sequence length for this dataset is 41 as displayed below.

[2965]:
```
%matplotlib inline

# importing our statistical libraries
import pandas as pd
import matplotlib.pyplot as plt

# importing the imdb dataset
imdb = pd.read_csv('imdb_labelled.txt', sep='          ', engine='python',␣
 ↪header=None)
imdb.columns = ['review', 'sentiment']
imdb
```

[2965]:

	review	sentiment
0	A very, very, very slow-moving, aimless movie …	0
1	Not sure who was more lost - the flat characte…	0
2	Attempting artiness with black & white and cle…	0
3	Very little music or anything to speak of.	0
4	The best scene in the movie was when Gerardo i…	1
..	…	…
995	I just got bored watching Jessice Lange take h…	0
996	Unfortunately, any virtue in this film's produ…	0
997	In a word, it is embarrassing.	0
998	Exceptionally bad!	0
999	All in all its an insult to one's intelligence…	0

[1000 rows x 2 columns]

[2966]:
```
# additional dataset information

print("Dataset Shape: ",imdb.shape,"\n")
print("Dataset Info: ", imdb.info(),"\n")
print("Dataset Description: \n")
print(imdb.describe().transpose())
```

```
Dataset Shape:  (1000, 2)

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype
```

```
---  ------     --------------  -----
 0   review     1000 non-null   object
 1   sentiment  1000 non-null   int64
dtypes: int64(1), object(1)
memory usage: 15.8+ KB
Dataset Info:  None

Dataset Description:

            count  mean      std  min  25%  50%  75%  max
sentiment  1000.0   0.5  0.50025  0.0  0.0  0.5  1.0  1.0
```

[2967]:
```
#checking for missing/null values
print(imdb.isnull().sum())

#checking for duplicate values of any rows
print("\nDuplicates: ",imdb.duplicated().sum())
```

```
review       0
sentiment    0
dtype: int64

Duplicates:  3
```

[2968]:
```
# dropping duplicates
imdb = imdb.drop_duplicates()

#checking for duplicate after dropping
print("\nDuplicates: ",imdb.duplicated().sum(),"\n")

# checking shape after duplicate removals
print("Dataset Shape: ", imdb.shape)
print("\nDataset Description: \n")
print(imdb.describe().transpose())
```

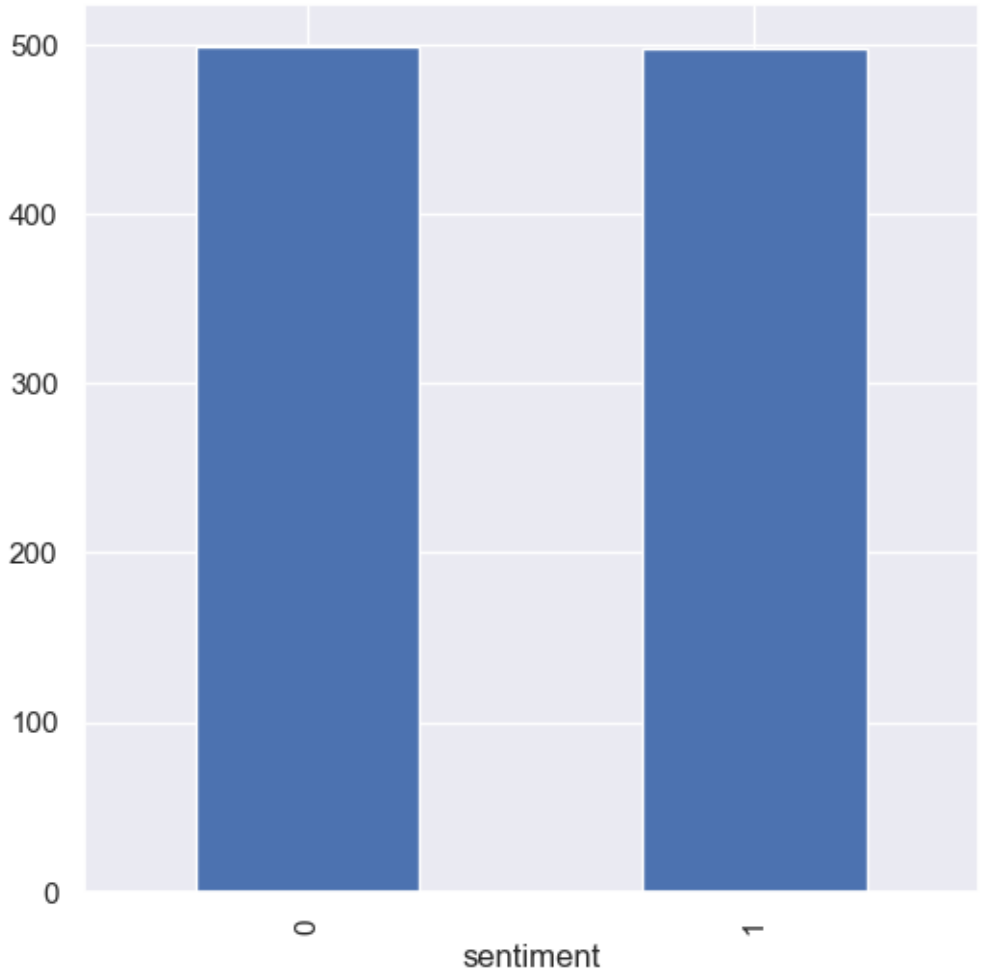
```
Duplicates:  0

Dataset Shape:  (997, 2)

Dataset Description:

            count      mean       std  min  25%  50%  75%  max
sentiment  997.0  0.499498  0.500251  0.0  0.0  0.0  1.0  1.0
```

[2969]:
```
imdb.groupby(['sentiment'])[['sentiment']].count()
```

[2969]:
```
           sentiment
sentiment
0                499
1                498
```

[2970]:
```
# sentiment bar plot
imdb.sentiment.value_counts().plot(kind='bar')
```

[2970]:
```
<Axes: xlabel='sentiment'>
```



Unusual Characters

[2972]:
```
# created a function to check if chars are english
def is_english(char):
    try:
        char.encode(encoding='utf-8').decode('ascii')
        return True
```

```
    except UnicodeDecodeError:
        return False

# collecting all non-english characters in the dataset
non_english_chars = set()

# looping through each review and collecting the non-english characters
for review in imdb['review']:
    non_english_chars.update({char for char in review if not is_english(char)})

# outputting the non-english characters
print(list(non_english_chars))
```

```
['\x97', 'é', '\x85', 'å', '\x96']
```

[2973]:
```
import nltk
from nltk.corpus import stopwords

# making sure stopwords are downloads and defined
nltk.download('stopwords')
stop_words = (stopwords.words('english'))

# printing stopwords
print(stop_words)
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're",
"you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he',
'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's",
'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what',
'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', 'am', 'is',
'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having',
'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or',
'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about',
'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above',
'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under',
'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why',
'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some',
'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very',
's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now',
'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'couldn',
"couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn',
"hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't",
'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn',
"shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn',
"wouldn't"]
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/kshaham/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

Vocabulary Size

[2975]:
```
import re
import unicodedata

# creating a function to clean and lemmatize the reviews
def clean_reviews(sentence):
    wnl = nltk.stem.WordNetLemmatizer()
    sentence = (unicodedata.normalize('NFKD', sentence)
                .encode('ascii', 'ignore')
                .decode('utf-8', 'ignore')
                .lower())
    words = re.sub(r'[^\w\s]', '', sentence).split()
    word_list = [wnl.lemmatize(word) for word in words if word not in␣
 ↪stop_words]
    return word_list

# making a deep copy of the dataframe
imdb_copy = imdb.copy()

# applying the clean_reviews function to create a tokenized column
imdb_copy['tokenized'] = imdb_copy['review'].apply(clean_reviews)

# creating a word_count column by counting the words in the tokenized column
imdb_copy['word_count'] = imdb_copy['tokenized'].apply(len)

# creating a cleaned_text column by joining the tokenized words back into a␣
 ↪single string
imdb_copy['cleaned_text'] = imdb_copy['tokenized'].apply(lambda x: ' '.join(x))

# defining positive and negative reviews
positive_reviews = imdb_copy.loc[imdb_copy['sentiment'] == 1]
negative_reviews = imdb_copy.loc[imdb_copy['sentiment'] == 0]

# created a function to get words from the dataset
def get_words(df, column):
    combined_text = ' '.join(df[column].astype(str).tolist())
    return clean_reviews(combined_text)

# getting the words from positive/negative and the entire dataset
all_words = get_words(imdb_copy, 'review')
positive_words = get_words(positive_reviews, 'review')
negative_words = get_words(negative_reviews, 'review')

# calculating the word frequencies
positive_word_freq = pd.Series(positive_words).value_counts()
negative_word_freq = pd.Series(negative_words).value_counts()
all_word_freq = pd.Series(all_words).value_counts()
```

```
# calculating the vocabulary size
vocab_size = len(all_word_freq)
print('Vocabulary size: ' + str(vocab_size))

# replacing the original imdb dataset with the modified one
imdb = imdb_copy

# displaying the first few rows of the updated dataframe
imdb.head()
```

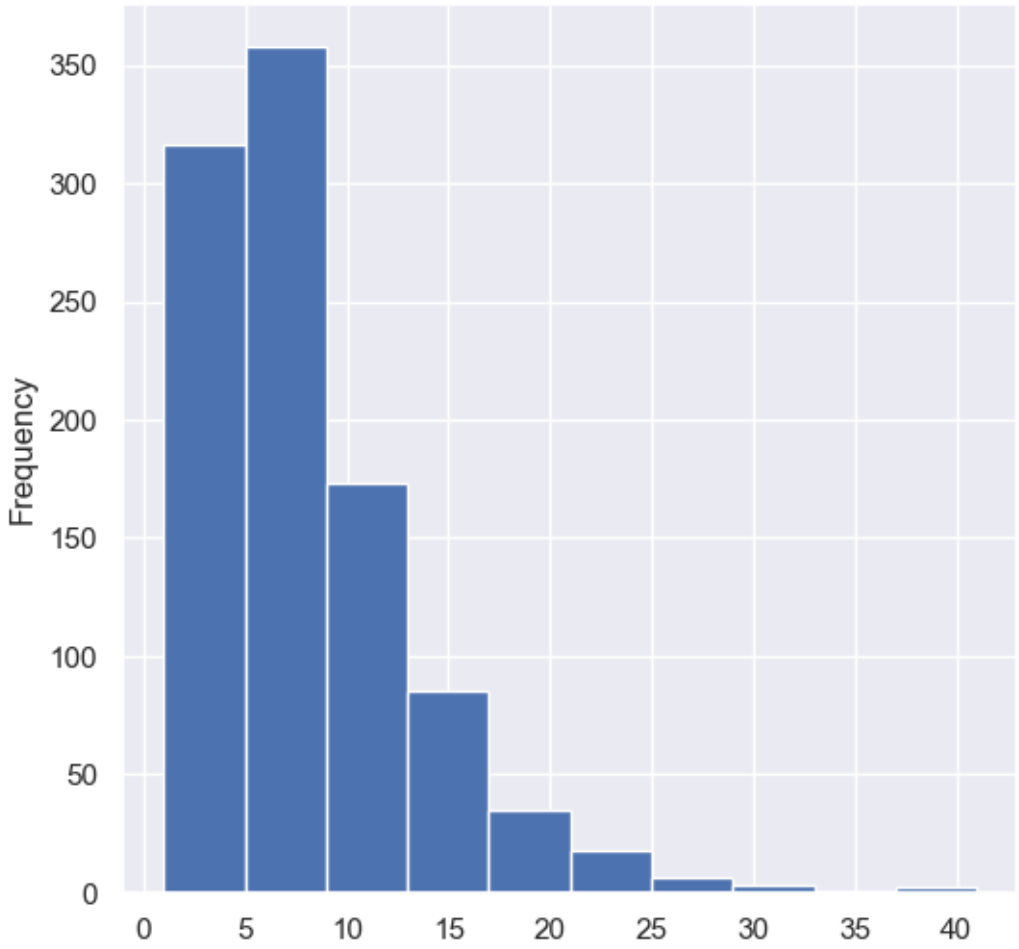
```
Vocabulary size: 2862
```

[2975]:

	review	sentiment	tokenized	word_count	cleaned_text
0	A very, very, very slow-moving, aimless movie …	0	[slowmoving, aimless, movie, distressed, drift…	7	slowmoving aimless movie distressed drifting y…
1	Not sure who was more lost - the flat characte…	0	[sure, lost, flat, character, audience, nearly…	8	sure lost flat character audience nearly half …
2	Attempting artiness with black & white and cle…	0	[attempting, artiness, black, white, clever, c…	18	attempting artiness black white clever camera …
3	Very little music or anything to speak of.	0	[little, music, anything, speak]	4	little music anything speak
4	The best scene in the movie was when Gerardo i…	1	[best, scene, movie, gerardo, trying, find, so…	10	best scene movie gerardo trying find song keep…

[2976]:
```
imdb['word_count'].plot(kind='hist')
```

[2976]:
```
<Axes: ylabel='Frequency'>
```



[2977]:
```
# checking word frequency in reviews
word_counts = (pd.concat([all_word_freq, positive_word_freq,␣
 ↪negative_word_freq], axis=1, sort=True)
                .set_axis(['all_reviews', 'positive_reviews',␣
 ↪'negative_reviews'], axis=1)
                .fillna(0)
                .apply(lambda s: s.astype(int)))

# checking most frequently occurring words
most_freq_words = word_counts.sort_values(by='all_reviews', ascending=False).
 ↪head(10)
print(most_freq_words)
```

	all_reviews	positive_reviews	negative_reviews
movie	207	101	106
film	182	114	68
one	79	43	36

bad	67	3	64
character	58	36	22
good	56	39	17
like	49	23	26
time	47	19	28
really	41	19	22
acting	41	15	26

[2978]:
```
# checking for unique words in both positive and negative reviews
unique_word_counts = pd.concat([
    word_counts[word_counts.negative_reviews == 0].
  ↪sort_values(by='positive_reviews').tail(10),
    word_counts[word_counts.positive_reviews == 0].
  ↪sort_values(by='negative_reviews').tail(10)
])

print(unique_word_counts)
```

	all_reviews	positive_reviews	negative_reviews
role	8	8	0
child	8	8	0
cinema	8	8	0
enjoyed	9	9	0
liked	10	10	0
loved	10	10	0
played	10	10	0
beautiful	11	11	0
excellent	15	15	0
wonderful	18	18	0
crap	6	0	6
horrible	6	0	6
poor	6	0	6
mess	8	0	8
avoid	8	0	8
suck	9	0	9
hour	9	0	9
worst	11	0	11
awful	14	0	14
stupid	14	0	14

Word Embedding Length

[2980]:
```
list_of_lens = []
for index, row in imdb.iterrows():
    list_of_lens.append(len(clean_reviews(row['review'])))
max_sequence_length = max(list_of_lens)
min_sequence_length = min(list_of_lens)
median_sequence_length = np.median(list_of_lens)
```

```
print('Max Sequence length: ' + str(max_sequence_length))
print('Min Sequence length: ' + str(min_sequence_length))
print('Median Sequence length: ' + str(median_sequence_length))
```

```
Max Sequence length: 41
Min Sequence length: 1
Median Sequence length: 6.0
```

[2981]:
```
proposed_embedding_length = int(round(np.sqrt(np.sqrt(vocab_size)), 0))
print('Proposed sequence embedding length: ' + str(proposed_embedding_length))
```

```
Proposed sequence embedding length: 7
```

[2982]:
```
imdb
```

[2982]:

	review	sentiment	tokenized	word_count	cleaned_text
0	A very, very, very slow-moving, aimless movie …	0	[slowmoving, aimless, movie, distressed, drift…	7	slowmoving aimless movie distressed drifting y…
1	Not sure who was more lost - the flat characte…	0	[sure, lost, flat, character, audience, nearly…	8	sure lost flat character audience nearly half …
2	Attempting artiness with black & white and cle…	0	[attempting, artiness, black, white, clever, c…	18	attempting artiness black white clever camera …
3	Very little music or anything to speak of.	0	[little, music, anything, speak]	4	little music anything speak
4	The best scene in the movie was when Gerardo i…	1	[best, scene, movie, gerardo, trying, find, so…	10	best scene movie gerardo trying find song keep…
..	…	…	…	…	…
995	I just got bored watching Jessice Lange take h…	0	[got, bored, watching, jessice, lange, take, c…	7	
996	Unfortunately, any virtue in this film's produ…	0	[unfortunately, virtue, film, production, work…	8	
997	In a word, it is embarrassing.	0	[word, embarrassing]	2	
998	Exceptionally bad!	0	[exceptionally, bad]	2	
999	All in all its an insult to one's intelligence…	0	[insult, one, intelligence, huge, waste, money]	6	

```
995      got bored watching jessice lange take clothes
996  unfortunately virtue film production work lost…
997                                    word embarrassing
998                                    exceptionally bad
999           insult one intelligence huge waste money

[997 rows x 5 columns]
```

0.4.1 B2. Tokenization

Tokenization aims to transform raw text into a format that is more useful for analysis, particularly by converting sentences into smaller, meaningful units or tokens (Burchfiel, 2023). In machine learning, tokenization is especially valuable as it converts words into sequences of integers, making them easier to process.

To avoid any potential bias, the data will be split into training and testing/validation sets before tokenization is performed. This ensures that no information from the test set influences the training process. The data will be split using a 70/30 ratio, with 70% allocated to training and 30% to testing/validation. The tokenizer is then fitted only on the training dataset before the datasets are converted to sequences. The `oov_token` parameter in the Tokenizer is used to manage out-of-vocabulary words, ensuring that words in the test set that were not encountered during training are accounted for.

[2984]:
```
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

# splitting the data into training (70%) and test/validation sets (30%)
x_train, x_temp, y_train, y_temp = train_test_split(imdb.cleaned_text, imdb.
 ↪sentiment,
                                                    train_size=0.7,␣
 ↪stratify=imdb.sentiment, random_state=17)

# second splitting the x_temp and y_temp into 15% for test and 15% for␣
 ↪validation
x_test, x_val, y_test, y_val = train_test_split(x_temp, y_temp,
                                                test_size=0.5, stratify=y_temp,␣
 ↪random_state=17)

# initializing and fitting the Tokenizer on the training data
tokenizer = Tokenizer(oov_token='OOV')
tokenizer.fit_on_texts(x_train)

# converting training, test, and validation splits to sequences
train_seq = tokenizer.texts_to_sequences(x_train)
test_seq = tokenizer.texts_to_sequences(x_test)
val_seq = tokenizer.texts_to_sequences(x_val)
```

```
# displaying the first padded sequence in the training set
print(train_seq[0])
```

```
[156, 435, 2, 35, 436, 288]
```

0.4.2 B3. Padding

Padding is a critical technique for use in neural networks, the process involves adding extra values, usually zeros to the sequences. This is because neural networks need to have inputs that are of similar shape and size. When padding sequences, the original lengths are adjusted by adding zeros to the end (or beginning) of sequences until they all match the maximum sequence length in the dataset. This is so the RNN model can distinguish between actual data and padding by allowing it to focus only on the non-padded portions of each sequence (GeeksforGeeks, 2024).

In the code below, padding is applied using the `pad_sequences` function to the end of the sequences using the `post` option as the padding parameter. The `maxlen` parameter was set to the maximum sequence length identified in the earlier section. This is to ensure the start of each sequence aligns with the model's processing and preserves the order of words in natural language.

[2987]:
```
from tensorflow.keras.utils import pad_sequences

# adding padding to reviews
padded_train = pad_sequences(train_seq, maxlen=max_sequence_length,␣
 ↪padding='post')
padded_test = pad_sequences(test_seq, maxlen=max_sequence_length,␣
 ↪padding='post')
padded_val = pad_sequences(val_seq, maxlen=max_sequence_length, padding='post')
padded_train[0]
```

[2987]:
```
array([156, 435,   2,  35, 436, 288,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0], dtype=int32)
```

0.4.3 B4. Sentiment Categories

There will be two categories of sentiment, with zero representing negative reviews and one representing positive reviews. The model will use a sigmoid activation function in the output layer, as it produces a probability between 0 and 1, aligning with the binary nature of the sentiment classification task (Sharma, 2022). In the model architecture section, a dense layer with 1 unit and a sigmoid activation function will be added, which makes the model suitable for binary classification.

0.4.4 B5. Data Preparation Steps

The steps used to prepare the data are outlined below:

- Importing the dataset: The dataset was inspected for characteristics, and duplicate values were removed if they existed.
- Cleaning the data: Special characters and symbols were removed, the text was converted to lowercase, and lemmatization was performed.

- Text Tokenization: Two columns were added: `tokenized`, which contains the reviews in tokenized form, and `cleaned_text`, which joins the tokens back into a sequence.
- Splitting Data: The data was split into training, test, and validation datasets with a 70/30 ratio. The split was performed before tokenizing to prevent the test data from influencing the training process.
- Padding: Padding was added to the sequences to ensure they are the same length before being used in the model.

0.4.5 B6. Prepared Dataset

[3017]:
```
# converting to dataframe to export
padded_train_df = pd.DataFrame(padded_train)
padded_test_df = pd.DataFrame(padded_test)
padded_val_df = pd.DataFrame(padded_val)

# exporting as csvs
padded_train_df.to_csv('padded_train.csv', index=False)
padded_test_df.to_csv('padded_test.csv', index=False)
padded_val_df.to_csv('padded_val.csv', index=False)
imdb.to_csv('imdb_cleaned.csv', index=False)
```

0.5 C. Network Architecture

[2993]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, GRU, Dense, Dropout,␣
 ↪BatchNormalization
from tensorflow.keras.optimizers import Nadam
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.regularizers import l2
import keras_tuner as kt

def create_model(hps):
    model = Sequential(name="GRU-Model")

    # embedding layer: converting sequences to vectors
    model.add(Embedding(input_dim=vocab_size,
                        output_dim=proposed_embedding_length,
                        input_length=max_sequence_length,
                        mask_zero=True))

    # gru layer with dropout, batch normalization, and l2 regularization
    model.add(GRU(units=hps.Int("neurons_n", min_value=16, max_value=128),␣
 ↪activation='tanh',
                  return_sequences=False, kernel_regularizer=l2(0.02)))
    model.add(BatchNormalization())
    model.add(Dropout(hps.Float("dropout_rate", min_value=0.25, max_value=0.6,␣
 ↪step=0.05)))
```

```
    # output layer for binary classification with l2 regularization
    model.add(Dense(1, activation='sigmoid', kernel_regularizer=l2(0.02)))

    # compiling the model with nadam optimizer and further reduced learning rate
    model.compile(loss="binary_crossentropy",
                  optimizer=Nadam(learning_rate=hps.Float("learning_rate",
                                                          min_value=1e-5,
                                                          max_value=3e-4,
                                                          sampling="log")),
                  metrics=['accuracy'])

    return model


# setting up callbacks for early stopping and saving the top model
early_stop_cb = EarlyStopping(patience=3, restore_best_weights=True)
model_check_cb = ModelCheckpoint("top_model.keras", save_best_only=True)

# initiating hyperparameter search using Keras Tuner's Hyperband
tuner = kt.Hyperband(create_model,
                     objective="val_accuracy",
                     overwrite=False,
                     max_epochs=10,
                     factor=3,
                     directory="hp_results",
                     project_name="sentiment_analysis",
                     seed=47)

tuner.search(padded_train, y_train,
             epochs=10,
             validation_data=(padded_val, y_val),
             batch_size=32,
             callbacks=[early_stop_cb, model_check_cb])

# retrieving and summarize the top model
top_model = tuner.get_best_models(num_models=1)[0]
top_model.summary()

# evaluating the model on the training, validation, and test sets
train_loss, train_accuracy = top_model.evaluate(padded_train, y_train)
val_loss, val_accuracy = top_model.evaluate(padded_val, y_val)
test_loss, test_accuracy = top_model.evaluate(padded_test, y_test)

print(f"\nTraining Accuracy: {train_accuracy:.4f} | Training Loss: {train_loss:.
 ↪4f}")
```

```
print(f"Validation Accuracy: {val_accuracy:.4f} | Validation Loss: {val_loss:.
  ↪4f}")
print(f"Test Accuracy: {test_accuracy:.4f} | Test Loss: {test_loss:.4f}")
```

```
Trial 30 Complete [00h 00m 06s]
val_accuracy: 0.5400000214576721

Best val_accuracy So Far: 0.8133333325386047
Total elapsed time: 00h 01m 36s
```

Model: "GRU-Model"

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 41, 7)	20,034
gru (GRU)	(None, 81)	21,870
batch_normalization (BatchNormalization)	(None, 81)	324
dropout (Dropout)	(None, 81)	0
dense (Dense)	(None, 1)	82

Total params: 42,310 (165.27 KB)

Trainable params: 42,148 (164.64 KB)

Non-trainable params: 162 (648.00 B)

```
22/22             0s 3ms/step -
accuracy: 0.9667 - loss: 0.6888
5/5               0s 3ms/step -
accuracy: 0.8063 - loss: 0.7789
5/5               0s 2ms/step -
accuracy: 0.6405 - loss: 0.8029

Training Accuracy: 0.9713 | Training Loss: 0.6901
Validation Accuracy: 0.8133 | Validation Loss: 0.7748
Test Accuracy: 0.6533 | Test Loss: 0.7997
```

[2994]:
```
top_trial = tuner.oracle.get_best_trials(num_trials=1)[0]
top_trial.summary()
```

```
Trial 0026 summary
Hyperparameters:
neurons_n: 81
dropout_rate: 0.3
learning_rate: 0.0002793674101922152
tuner/epochs: 10
tuner/initial_epoch: 0
tuner/bracket: 0
tuner/round: 0
Score: 0.8133333325386047
```

0.5.1 C2. Layers and Parameters

Layers The model consists of fives layers and are described below:

- Embedding Layer:
  - Converts sequences into vectors.
  - Enables the model to learn the semantic meaning of words.
- GRU Layer:
  - Gated Recurrent Unit (GRU) designed to maintain memory over sequences.
  - Selectively updates or forgets parts of the hidden state.
  - Addresses the vanishing gradient problem present in standard RNNs.
  - Includes L2 regularization to penalize large weights and prevent overfitting.
- Batch Normalization Layer:
  - Stabilizes the learning process.
  - Improves model performance by normalizing the output from the GRU layer.
- Dropout Layer:
  - Randomly deactivates a fraction of neurons during training.
  - Reduces the risk of overfitting by preventing the model from becoming too reliant on specific neurons.
- Dense Layer:
  - Receives processed information from the GRU layer and subsequent layers.
  - Determines the sentiment as either positive or negative (0 or 1).

Parameters

- Embedding layer: As this layer is converting the sequences into vectors, the number of parameters is determined by the size of the vocabulary, in this case the vocabulary size is 2862 multiplied by the `proposed_embedding_length` which was set to 7. The total number of parameters are: `2862 * 7 = 20034`
- GRU Layer: The number of parameters in the GRU layer depends on the number of neurons (`neurons_n`) and the input dimension (`proposed_embedding_length`). The total number of parameters is calculated as: 3×(neurons_n×neurons_n+neurons_n×input_dim+neurons_n) or simplified down to: `3(n^2+nm+2n)` (How Many Parameters Are in a Gated Recurrent Unit (GRU) Recurrent Neural Network (RNN) Layer?, n.d.).

- Batch Normalization Layer: This layer adds additional parameters equal to 2 times the number of neurons in the GRU layer. 1 parameter for scaling (gamma) and 1 for shifting (beta).
- Dropout Layer: The Dropout layer does not introduce any additional trainable parameters as it actually deactivates a portion of neurons during training to prevent overfitting.
- Dense Layer: The number of parameters in the Dense layer is equal to the `neurons_n` from the GRU layer plus one additional parameter for the bias term.

0.5.2 C3. Hyperparameters

Activation Functions The `tanh` activation was used in the GRU layer to assist the model with regulating the values flowing through the network (Phi, 2020). This allows the model to capture a broader range of information, particularly when deciding which information to keep or discard as it processes sequences. In the dense layer a `sigmoid` activation function was added, which makes the model suitable for binary classification.

Nodes Per Layer The range for the number of nodes, `neurons_n`, was selected between 16 and 128 to allow the model to explore varying levels of complexity. This range was chosen based on recommendations from the Keras documentation, and aimed to strike a balance between avoiding underfitting and overfitting. The Hyperband tuner identified 81 nodes as having the highest accuracy score, indicating that this level of complexity was most effective for the model.

Loss Function The Binary Crossentropy loss function was chosen because it is used in classification tasks where the goal is to categorize an instance into one of two distinct classes. According to Leikin (2024), "Binary Cross-Entropy measures the dissimilarity between the actual labels and the predicted probabilities of the data points being in the positive class. It penalizes the predictions that are confident but wrong.". This loss function is particularly effective in binary classification problems, as it ensures the model is penalized more heavily for incorrect confident predictions, encouraging the model to be more cautious in its predictions.

Optimizer The Nadam optimizer was used as it is an extension of the Adam algorithm that incorporates Nesterov momentum and can result in better performance of the optimization algorithm (Brownlee, 2021). This can be particularly beneficial in training deep neural networks, as it may lead to faster convergence and potentially improved model accuracy compared to standard Adam.

Stopping Criteria The analysis used early stopping as the stopping criterion, with a patience value of three, meaning the model would halt training if the validation loss did not improve over three consecutive epochs. This technique is intended to prevent overfitting. The final validation accuracy of 81.33% suggests a drop in performance on unseen data, highlighting potential generalization issues. The training accuracy of 97.13% raises concerns about overfitting, as the model performed exceptionally well on the training data, yet struggled to generalize to new data. This indicates that further model adjustments or the introduction of more diverse training data may be necessary to improve overall model performance.

Evaluation Metric The evaluation metric that the model is using is the validation accuracy, this ensures that the model performs well on unseen data and is the standard for binary classification

problems. The validation accuracy was 81.33% which indicates improved performance on unseen data. The training accuracy with a score of 97.13% indicates overfitting, as the model performed exceptionally well on the training data. The test accuracy had a score of 65.33%, meaning the model performs pretty well on the testing data. The metrics are outputted below.

[2996]:
```
# evaluating the model on the training dataset
train_loss, train_accuracy = top_model.evaluate(padded_train, y_train,␣
 ↪verbose=0)
print(f"Training Accuracy: {train_accuracy:.4f}")
print(f"Training Loss: {train_loss:.4f}")

# evaluating the model on the validation dataset
val_loss, val_accuracy = top_model.evaluate(padded_val, y_val, verbose=0)
print(f"Validation Accuracy: {val_accuracy:.4f}")
print(f"Validation Loss: {val_loss:.4f}")

# evaluating the model on the test dataset
test_loss, test_accuracy = top_model.evaluate(padded_test, y_test, verbose=0)
print(f"Test Accuracy: {test_accuracy:.4f}")
print(f"Test Loss: {test_loss:.4f}")
```

```
Training Accuracy: 0.9713
Training Loss: 0.6901
Validation Accuracy: 0.8133
Validation Loss: 0.7748
Test Accuracy: 0.6533
Test Loss: 0.7997
```

0.6 D. Model Evaluation

Stopping Criteria As mentioned in previous sections, the stopping criteria was used to prevent overfitting by the model while also assisting the tuning process to prevent the use of redundant epochs. This enabled training to stop at epoch number 3 which can be seen below.

[3013]:
```
# training the top model
history = top_model.fit(padded_train, y_train,
                        epochs=10,
                        validation_data=(padded_val, y_val),
                        callbacks=[early_stop_cb, model_check_cb])
```

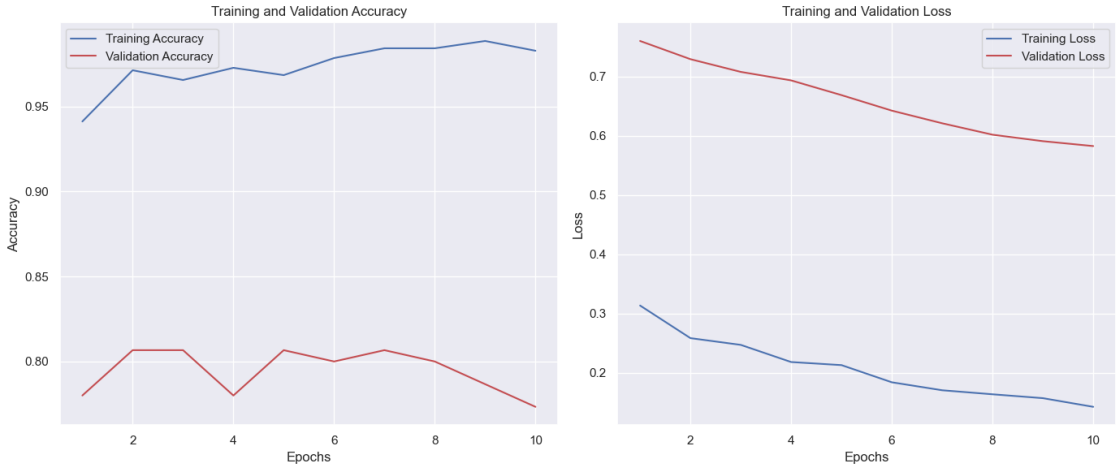
```
Epoch 1/10
22/22             0s 14ms/step -
accuracy: 0.9817 - loss: 0.1199 - val_accuracy: 0.7867 - val_loss: 0.5413
Epoch 2/10
22/22             0s 15ms/step -
accuracy: 0.9965 - loss: 0.0937 - val_accuracy: 0.7933 - val_loss: 0.5468
Epoch 3/10
22/22             0s 11ms/step -
accuracy: 0.9950 - loss: 0.0896 - val_accuracy: 0.7933 - val_loss: 0.5569
```

0.6.1 D2. Model Fitness

The model showed signs of overfitting, as indicated by the high training accuracy score of 97.13%. The model's performance was assessed by evaluating it on both the testing and validation sets. Early stopping was implemented to mitigate overfitting, and its criteria were adjusted to see if there was any improvement. A dropout layer was added which randomly deactivates a fraction of neurons during training, to help prevent overfitting by making the network less reliant on specific neurons. However, further adjustments to the model may be necessary such as providing more data for training or optimizing the network further to enhance its fitness.

0.6.2 D3. Training Visualizations

[3001]:
```
import matplotlib.pyplot as plt

# retrieving the accuracy and loss values from the history object
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(acc) + 1)

# plotting the training and validation accuracy
plt.figure(figsize=(14, 6))
plt.subplot(1, 2, 1)
plt.plot(epochs, acc, 'b', label='Training Accuracy')
plt.plot(epochs, val_acc, 'r', label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# plotting the training and validation loss
plt.subplot(1, 2, 2)
plt.plot(epochs, loss, 'b', label='Training Loss')
plt.plot(epochs, val_loss, 'r', label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
```



0.6.3 D4. Predictive Accuracy

Training Accuracy: 0.9713 Training Loss: 0.6901 Validation Accuracy: 0.8133 Validation Loss: 0.7748 Test Accuracy: 0.6533 Test Loss: 0.7997 - Training Accuracy and Loss: The training accuracy of 97.13

- Validation Accuracy and Loss: The validation accuracy of 81.33

0.7 E. Model Code

```
[ ]: top_model.save("top_model.keras")
```

0.8 F. Model Functionality

The neural network selected for this assignment is a GRU (Gated Recurrent Unit), a type of recurrent neural network specifically designed to handle sequential data. GRU was chosen because of its ability to overcome the vanishing gradient problem, which can prevent standard RNNs from retaining information from earlier steps in long sequences. This model was implemented to analyze the sentiment of reviews in the IMDB dataset, but it can also be applied to any review analysis task where the goal is a binary outcome.

The network consists of five layers:

- Embedding Layer: Converts sequences into vectors, allowing the model to learn the semantic meaning of words.
- GRU Layer: This layer maintains memory over sequences by updating or forgetting parts of the hidden state through a series of gates. This layer also includes L2 regularization to reduce overfitting.
- Batch Normalization Layer: This layer stabilizes the learning process by normalizing the output from the GRU layer, which can improve model performance and convergence.
- Dropout Layer: Randomly deactivates a fraction of neurons during training, which helps prevent overfitting by making the network less reliant on specific neurons.

- Dense Layer: Contains a single neuron utilizing the `sigmoid` activation function, which is ideal for binary classification tasks like sentiment analysis.

The Nadam optimizer was chosen over the Adam optimizer as it can be beneficial in training deep neural networks, potentially leading to faster convergence and improved model accuracy.

0.9 G. Course of Action

A recommended course of action for this analysis is to address the issue of overfitting, as indicated by the exceptionally high training accuracy and the discrepancy between the training and validation loss scores. Further improvements to the model such as including more data or fine-tuning can help mitigate overfitting, making it more useful in a production environment. Once refined, the model can be applied to other datasets for sentiment analysis of reviews. For instance, WGU Hospital could utilize this model to analyze patient reviews, enabling more informed resource allocation decisions based on the sentiment analysis.

H. Third-party Code References

GeeksforGeeks. (2024, March 28). How to handle sequence padding and packing in PyTorch for RNNs? GeeksforGeeks. https://www.geeksforgeeks.org/how-do-you-handle-sequence-padding-and-packing-in-pytorch-for-rnns/

Hanyjr. (2024, May 6). Hyper parameter Tuning in Neural Networks. Kaggle. https://www.kaggle.com/code/hanyjr/hyper-parameter-tuning-in-neural-networks

Mukeshmanral. (2023, October 7). RNN_LSTM_GRU-Implementation+HyperparameterTunin. Kaggle. https://www.kaggle.com/code/mukeshmanral/rnn-lstm-gru-implementation-hyperparametertunin

Satishgunjal. (2020, December 6). Tokenization in NLP. Kaggle. https://www.kaggle.com/code/satishgunjal/tokenization-in-nlp Team, K. (n.d.).

Keras documentation: GRU layer. https://keras.io/api/layers/recurrent_layers/gru/

UCI Machine Learning Repository. (n.d.). https://archive.ics.uci.edu/dataset/331/sentiment+labelled+sentences

I. References

Brownlee, J. (2021, October 11). Gradient descent optimization with NaDAM from scratch. MachineLearningMastery.com. https://machinelearningmastery.com/gradient-descent-optimization-with-nadam-from-scratch/

Burchfiel, A. (2023, May 11). What is NLP (Natural Language Processing) Tokenization? Tokenex. https://www.tokenex.com/blog/ab-what-is-nlp-natural-language-processing-tokenization/

D213 Task 2 Cohort Webinar PPT.pptx. (n.d.). https://westerngovernorsuniversity.sharepoint.com/:p:/r/sites/Da
72F1-4964-9942-1F3790EF1D59%7D&file=D213%20Task%202%20Cohort%20Webinar%20PPT.pptx&action=edi

GeeksforGeeks. (2024, March 28). How to handle sequence padding and packing in PyTorch for RNNs? GeeksforGeeks. https://www.geeksforgeeks.org/how-do-you-handle-sequence-padding-and-packing-in-pytorch-for-rnns/

How many parameters are in a gated recurrent unit (GRU) recurrent neural network (RNN) layer? (n.d.). Cross Validated. https://stats.stackexchange.com/questions/328926/how-many-parameters-are-in-a-gated-recurrent-unit-gru-recurrent-neural-network

Leikin, I. (2024, July 4). Understanding binary Cross-Entropy and log loss for effective model monitoring. Aporia. https://www.aporia.com/learn/understanding-binary-cross-entropy-and-log-loss-for-effective-model-monitoring/

Phi, M. (2020, June 28). Illustrated Guide to LSTM's and GRU's: A step by step explanation. Medium. https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

Sharma, S. (2022, November 20). Activation functions in neural networks - towards data science. Medium. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

Stewart, E. (n.d.). What is a Gated Recurent Unit (GRU) and How Does it Work? | Enterprise Tech News EM360Tech. https://em360tech.com/tech-article/gated-recurrent-unit-gru

What is a recurrent neural network (RNN)? | IBM. (n.d.). https://www.ibm.com/topics/recurrent-neural-networks Word embeddings. (n.d.).

TensorFlow. https://www.tensorflow.org/text/guide/word_embeddings