D604 Task 2: Sentiment Analysis

Jesse Byers

College of Information Technology, Western Governors University

Dr. Sherin Aly (Instructor)

June 26, 2025

D604 Task 2: Sentiment Analysis

Part I: Research Question

A1. Research Question

To what extent can a neural network make accurate predictions on reviews of live performances?

A2. Objectives Or Goals

This analysis is being performed for a theatrical production company. The company is interested in using a model that can classify reviews of its performances (plays, musicals, and live musical performances) as positive or negative. The classification of the reviews will allow the company to streamline its review analysis and make better decisions around which acts to promote. The neural network described in this paper is a prototype trained on IMDB movie review data. If the prototype performs within a reasonable degree of accuracy, the company can invest in labeling a set of their existing reviews and re-training the model with reviews that more closely match the context of the live performances.

A3. Prescribed Network

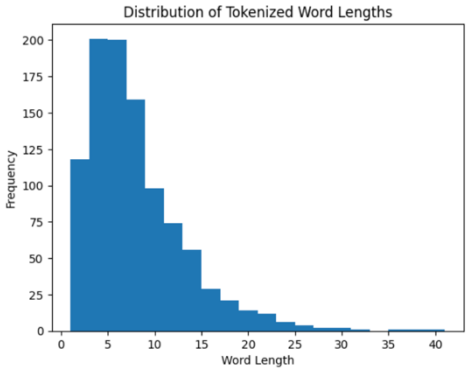
A deep learning binary classification neural network will be used to complete this prototype for sentiment analysis. Specifically, the network will be a hybrid design including a convolutional block (CNN) as well as a Long Short-Term Memory block (LSTM). The convolutional block is advantageous in detecting local patterns within the text input (for example, recognizing adjacent pairs of words such as “not good” or “extremely good”). The Bidirectional LSTM block is advantageous because it is able to remember and use context in both directions (words that are behind or in front of certain text). This network architecture will be described in more detail in a later section.

Part II: Data Preparation

B1. Data Exploration

The IMDB dataset was used for this analysis. In data exploration, the following details were noticed about the data:

- Unusual Characters
    - The dataset included a number of non-text characters that needed to be addressed. These included punctuation marks (such as quotes, periods, commas, dashes, etc), formatting sequences (such as line breaks). There were also a lot of numbers contained within the text.
- Vocabulary Size
    - After data cleaning, including the removal of stop words, the total vocabulary size was 2707 words. As will be described in a later section, the top 1500 words in this vocabulary were included in the network analysis.
- Word Embedding Length
    - A word embedding length of 16 was chosen for this model. This means that each input word is represented as a 16-dimensional vector before it is fed to the convolutional layer of the network. 16 is a fairly small number and was chosen to reduce the chance of overfitting.
- Justification for Maximum Sequence Length
    - A maximum sequence length was chosen for each text input, based on a statistical analysis of the remaining words after data pre-processing. As the histogram and



Min: 1
Mean: 7.55
Median: 6.0
90th percentile: 14.0
95th percentile: 18.0
Max: 41

summary data shows, while the maximum word sequence length was 41, the mean and median were much lower. 20 was chosen as the maximum sequence length because it would include 95% of the full sequences, and would clip content from the end of the longest (outlier) sequences. Ideally, this could reduce noise and overfitting potential by eliminating the frequency of potentially rambling reviews that might veer off-topic towards the end.

B2. Tokenization

Input text was first tokenized into individual words within the data cleaning loop, which will be further described in section B5. The NLTK word_tokenize function was used to accomplish this. This code converted the string of text into a list of word elements, which was then used to eliminate stop words.

```
from nltk import word_tokenize

tokens = word_tokenize(new_line, language='english', preserve_line=True)
filtered_tokens = [word for word in tokens if word not in stop_words]
```

Next, the TensorFlow Keras Tokenizer was used to define how the token words would be used in the network. The Tokenizer was instantiated with 1500 words, which means that it will use only the top 1500 words based on frequency. This number allows the network to ignore the least frequently used words, which are coded as <OOV>, which might represent noise. The 1500 most frequent words are then mapped to a dictionary, with each word encoded to an integer. The tokenizer provides the word index mapping words to integers, as well as word frequency counts for each word.

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

# create tokenizer and padded_sequence
```

```
words = df.review.values
tokenizer = Tokenizer(num_words=1500, oov_token='<OOV>')
tokenizer.fit_on_texts(words)
vocab_size = len(tokenizer.word_index) + 1
encoded_docs = tokenizer.texts_to_sequences(words)
padded_sequence = pad_sequences(encoded_docs, maxlen=20)
word_index = tokenizer.word_index
word_counts = tokenizer.word_counts
```

B3. Padding Process

Next, pad_sequences was used to convert each input into a standard size regardless of each review's starting word length. This is done by adding zeros to the beginning of each sequence, followed by the encoded integer values of each word in the sequence. Below is a screenshot of the first review after cleaning, tokenization, and padding, represented by a list of 20 values:

```
[   0    0    0    0    0    0    0    0    0    0    0    0  995
  996    3  997  611  309   79]
```

This represents the following original negative review:

original	review
A very, very, very slow-moving, aimless movie about a distressed, drifting young man. \t0\n	[slowmoving, aimless, movie, distress, drift, young, man]

B4. Categories of Sentiment

Two categories of sentiment are used in this model. Positive reviews are represented by 1, and negative reviews are represented by 0. The following activation function is used to generate this binary classification output. The first parameter represents the final node of the network with a binary output, which will be a float value between 0 and 1.

```
model.add(Dense(1, activation='sigmoid', kernel_regularizer=regularizers.l2(0.01)))
```

B5. Steps to Prepare the Data

The following steps were used to prepare the data prior to splitting into training (80%), testing (10%), and validation sets (10%). The training set was kept at 80% because of the relatively small size of the overall dataset.

1. Load Data

```
# load data
with open("imdb_labelled.txt", "r", encoding="utf-8") as file:
    lines = file.readlines()

# print a sample of raw data
lines[0:5] # list of individual reviews with sentiment label
```

```
['A very, very, very slow-moving, aimless movie about a distressed, drifting young man.  \t0\n',
 'Not sure who was more lost - the flat characters or the audience, nearly half of whom walked out.  \t0\n',
 'Attempting artiness with black & white and clever camera angles, the movie disappointed - became even more ridiculous - as the acting was poor and the plot and lines almost non-existent.
\t0\n',
 'Very little music or anything to speak of.  \t0\n',
 'The best scene in the movie was when Gerardo is trying to find a song that keeps running through his head.  \t1\n']
```

2. Data Cleaning Loop (see code below)
    a. Remove leading and trailing whitespace from each review
    b. Convert all text to lowercase
    c. Remove punctuation marks
    d. Remove the sentiment label from the end of each review
    e. Replace numerals 1-10 with text
        i. *Note: While numbers are generally removed from text, I noticed that the numbers in the reviews were meaningful in relation to the overall sentiment. For example, "10/10" represented a positive sentiment. After experimentation, I found that converting these numbers to text led to higher accuracy.*
    f. Remove remaining numbers and non-word characters
    g. Remove escape sequences ("\n", "\t")
    h. Remove stop words
    i. Use lemmatization to convert words to their root form

```
# clean / pre-process data
def replace_num_with_text(line):
    new_line = ""
    nums = {"0": "zero",
            "1": "one",
            "2": "two",
            "3": "three",
            "4": "four",
            "5": "five",
            "6": "six",
            "7": "seven",
            "8": "eight",
            "9": "nine",
            "10": "ten",
            }
    for i, char in enumerate(line):
        if char in nums.keys():
            if char == "1" and line[i+1] == "0":
                new_line = new_line + nums["10"]
            else:
                new_line = new_line + nums[char]
        else:
            new_line = new_line + char
    return new_line

stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

df = pd.DataFrame(columns=["original", "review", "label", "word_length"])

new_lines = []
labels = []

for line in lines:
    # remove leading and trailing whitespace
    new_line = line.strip()
    # convert all text to lowercase
    new_line = new_line.lower()
    # remove punctuation
    punct_remover = str.maketrans(' ', ' ', string.punctuation)
    new_line = new_line.translate(punct_remover)
    # extract label from end
    label = int(new_line[-1])
    new_line = new_line[:-3] # remove label
    labels.append(label)
```

```
    # replace 1-10 numbers to text
    new_line = replace_num_with_text(new_line)
    # remove numbers and other non-word characters
    new_line = re.sub(r'\d+', ' ', new_line)
    # remove escape sequences
    new_line = new_line.replace("\n", " ").replace("\t", " ")
    # remove stop words
    tokens = word_tokenize(new_line, language='english', preserve_line=True)
    filtered_tokens = [word for word in tokens if word not in stop_words]
    # lemmatization
    for i, word in enumerate(filtered_tokens):
        new_word = lemmatizer.lemmatize(word, pos='v')
        filtered_tokens[i] = new_word
    new_lines.append(filtered_tokens)  # append to new_lines list
```

	original	review	label	word_length
13	It Was So Cool. \t1\n	[cool]	1	1
420	It just blew. \t0\n	[blow]	0	1
406	I liked it. \t1\n	[like]	1	1
92	It was horrendous. \t0\n	[horrendous]	0	1
105	Very disappointing. \t0\n	[disappoint]	0	1
...	...	...	...	...
428	The use of slow-motion needlessly repeats itse...	[use, slowmotion, needlessly, repeat, througho...	0	29
469	A cheap and cheerless heist movie with poor ch...	[cheap, cheerless, heist, movie, poor, charact...	0	31
421	This movie is excellent!Angel is beautiful and...	[movie, excellentangel, beautiful, scamp, ador...	1	35
390	Though The Wind and the Lion is told largely t...	[though, wind, lion, tell, largely, eye, son, ...	1	37
620	This is a masterful piece of film-making, with...	[masterful, piece, filmmaking, many, theme, si...	1	41

3. Tokenize and Pad Data
    a. See code and explanation in sections B2 and B3
4. Split Data into Training (80%), Validation (10%), and Testing (10%) Data Sets

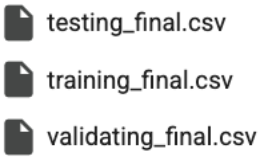
```
# split dataset into training(80%), validating(10%), and testing(10%) datasets

X = padded_sequence
y = df["label"]

# 80% training
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.2, random_state=19)
# split temp into 50% validating and 50% testing
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=19)
```

```
print(f"Length of Training Set: {len(X_train)}")
print(f"Length of Validating Set: {len(X_val)}")
print(f"Length of Testing Set: {len(X_test)}")
```

```
Length of Training Set: 800
Length of Validating Set: 100
Length of Testing Set: 100
```

B6. Prepared Dataset

The training, validation, and testing datasets are included with the submission.



Part III: Network Architecture

C1. Model Summary

Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 20, 16)	43,312
spatial_dropout1d (SpatialDropout1D)	(None, 20, 16)	0
conv1d (Conv1D)	(None, 18, 32)	1,568
max_pooling1d (MaxPooling1D)	(None, 9, 32)	0
bidirectional (Bidirectional)	(None, 32)	6,272
dropout (Dropout)	(None, 32)	0
dense (Dense)	(None, 1)	33

Total params: 51,185 (199.94 KB)
Trainable params: 51,185 (199.94 KB)
Non-trainable params: 0 (0.00 B)
None

C2. Network Architecture

```
# create LSTM model
embedding_vector_length = 16
model = Sequential()
model.add(Embedding(vocab_size, embedding_vector_length))
model.add(SpatialDropout1D(0.2))
model.add(Conv1D(filters=32, kernel_size=3, activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(Bidirectional(LSTM(16, dropout=0.3, recurrent_dropout=0.5)))
model.add(Dropout(0.4))
model.add(Dense(1, activation='sigmoid', kernel_regularizer=regularizers.l2(0.01)))
model.build(input_shape=(None, 20))
model.compile(loss='binary_crossentropy',optimizer=Adam(learning_rate=0.0001),
metrics=['accuracy'])
print(model.summary())
```

This model has 7 layers and 51,585 parameters, as described in the model summary above. The embedding layer converts each word in the input to a 16-dimensional vector. Then, the dropout layer causes the network to randomly drop 20% of the feature maps in each run, which helps for generalization and reduces the chance of overfitting. The convolutional layer is a locally-connected layer that applies 32 filters to detect local patterns across adjacent words in the input. The pooling layer then reduces the input size by half, keeping only the strongest features or patterns from each region of the text. After this, the bidirectional LSTM layer processes the input, using context it gains from looking both forward in the sequence as well as backward in the sequence, which can help detect sentiment patterns across longer distances than the convolutional layer alone. Another dropout layer is used, which again drops random outputs from the LSTM to further avoid overfitting. Finally, the last layer is the dense layer, which uses the sigmoid activation function to output the binary classification result, a float between 0 and 1, representing a prediction of positive or negative sentiment.

C3. Hyperparameters

- Activation functions
    - Activation functions are used to introduce non-linearity, which allows the network to detect and learn more complex patterns in the data. The convolutional layer uses a Rectified Linear Unit (ReLu) activation function. The ReLu function ensures that only positive signals are passed on to the next layer of the model. This helps to ensure that the most prominent features are detected, as well as more complex patterns. The dense layer uses a sigmoid activation layer to output a probability between 0 and 1, which represents the sentiment prediction. The sigmoid activation function is appropriate because the network is performing binary classification predictions.
- Number of nodes per layer
    - In general, the model architecture was chosen in an effort to progressively increase the number of channels at each layer (depth), while decreasing the input size. This progression allows the model to discover more meaningful, complex patterns in the image while conserving resources. The number of nodes per layer is calculated from the output shape, multiplying the sequence length by the number of features. The embedding layer starts with 320 nodes per input (20 tokens times 16 features). The convolutional layer increases to 576 nodes, but not all are used due to dropout. The pooling layer reduces the nodes by half, to 288. The LSTM reduces it further to 32 nodes, and the dense layer reduces it to 1 node to produce the binary classification output.

- Loss function
    - Binary Cross-Entropy was used as the loss function. This is appropriate for binary classification tasks because it computes the difference between the actual label with the predicted label, and uses this difference to adjust the weights during backpropagation, which penalizes incorrect predictions.
- Optimizer
    - The Adam optimizer was used for this model. This optimizer was a good choice because it is more flexible and adaptable than other optimizers, such as SGD. The Adam optimizer has an adaptive learning rate, meaning that it can adjust its learning rate for individual parameters. This adaptability can also lead to faster training times.
- Stopping criteria
    - Early stopping was used to stop training and restore and save the optimal weights. The code below uses the minimum validation loss as a stopping criterion and uses a patience of 7 epochs to determine whether the criterion has been met.

```
model_checkpoint_callback = ModelCheckpoint(
                            filepath="checkpoint.model.keras",
                            save_weights_only=False,
                            monitor='val_loss',
                            mode='min',
                            save_best_only=True,
                            verbose=1)
early_stopping_callback = EarlyStopping(
                            monitor='val_loss',
                            mode='min',
                            patience=7,
                            restore_best_weights=True,
                            verbose=1)
callbacks = [early_stopping_callback, model_checkpoint_callback]
```

Part IV: Neural Network Evaluation

D1. Stopping Criteria

Training was run with a maximum of 75 epochs, but the stopping criteria ended training early and restored the weights from epoch 45. The screenshot below shows the training output for the last epochs, showing the impact of the stopping criteria and patience, and shows that the model was last saved at the end of epoch 45.

```
Epoch 45/75
49/50 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.8887 - loss: 0.3306
Epoch 45: val_loss improved from 0.55377 to 0.54365, saving model to checkpoint.model.keras
50/50 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.8896 - loss: 0.3299 - val_accuracy: 0.7500 - val_loss: 0.5437
Epoch 46/75
49/50 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9266 - loss: 0.2789
Epoch 46: val_loss did not improve from 0.54365
50/50 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.9260 - loss: 0.2794 - val_accuracy: 0.7700 - val_loss: 0.5498
Epoch 47/75
47/50 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9293 - loss: 0.2700
Epoch 47: val_loss did not improve from 0.54365
50/50 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9283 - loss: 0.2708 - val_accuracy: 0.7500 - val_loss: 0.5507
Epoch 48/75
49/50 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9457 - loss: 0.2265
Epoch 48: val_loss did not improve from 0.54365
50/50 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.9450 - loss: 0.2275 - val_accuracy: 0.7600 - val_loss: 0.5547
Epoch 49/75
47/50 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9268 - loss: 0.2451
Epoch 49: val_loss did not improve from 0.54365
50/50 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9267 - loss: 0.2454 - val_accuracy: 0.7700 - val_loss: 0.5537
Epoch 50/75
46/50 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9397 - loss: 0.2269
Epoch 50: val_loss did not improve from 0.54365
50/50 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9396 - loss: 0.2269 - val_accuracy: 0.7700 - val_loss: 0.5583
Epoch 51/75
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9350 - loss: 0.2219
Epoch 51: val_loss did not improve from 0.54365
50/50 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.9349 - loss: 0.2221 - val_accuracy: 0.7600 - val_loss: 0.5606
Epoch 52/75
47/50 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9189 - loss: 0.2441
Epoch 52: val_loss did not improve from 0.54365
50/50 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.9201 - loss: 0.2424 - val_accuracy: 0.7600 - val_loss: 0.5583
Epoch 52: early stopping
Restoring model weights from the end of the best epoch: 45.
```

D2. Fitness

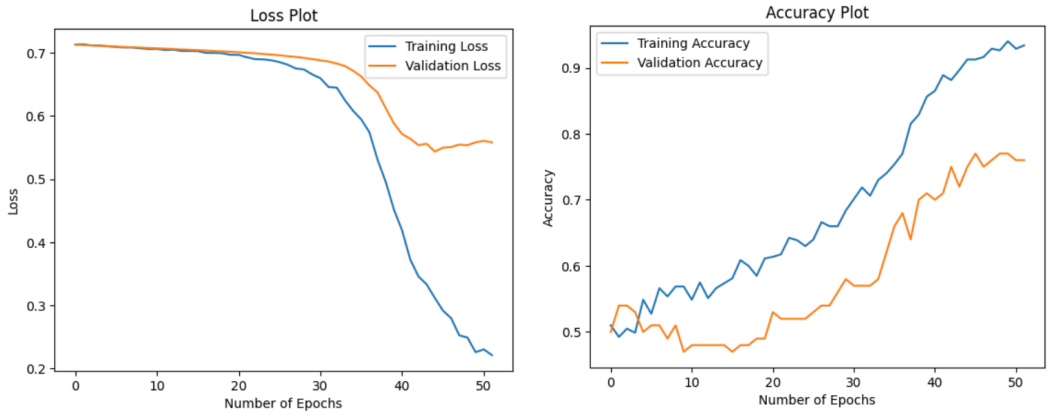
Many actions were taken to address overfitting, in particular in this model. First, the stopping criteria were used to ensure that the model was not overtrained, as evidenced by the loss plot in the section below. The plot shows that the minimum loss was achieved at epoch 45, and afterwards the started to rise again. Continued training would result in continued improvement in training accuracy, but decreased accuracy in validation accuracy due to overfitting.

Other actions to avoid overfitting include the following:

- Reducing the vocabulary size to 1500 to reduce noise from infrequent words
- Reducing sequence length to 20 to minimize the impact of padded values and clip rambling reviews
- Using a low learning rate (0.0001) to avoid jumping over the minimum
- Using dropout layers to make the model more robust and increase generalization when learning patterns

Overall, these design choices helped to reduce overfitting; however, overfitting is difficult to avoid with small datasets, such as this training set with only 800 samples. Overfitting could be further avoided by using a larger dataset or by artificially augmenting this dataset through creating new samples using synonyms of existing phrases or translating back and forth across languages to generate new samples.

D3. Training Process



D4. Predictive Accuracy

Overall, the trained model performed reasonably well on the test dataset during evaluation. First, the model was evaluated with the test data and achieved an accuracy of about 82%.

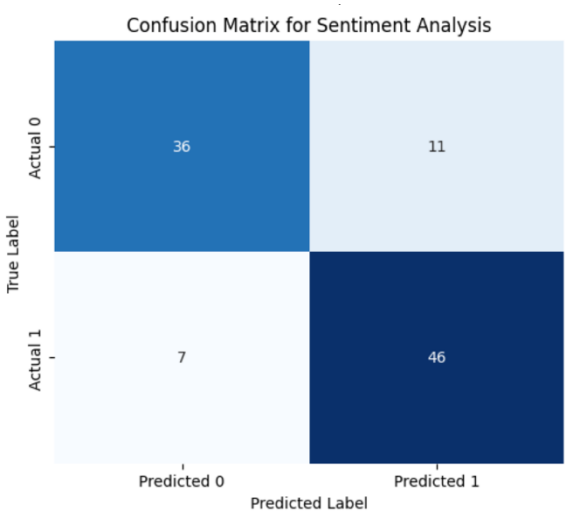
```
# load and test best model
model = tf.keras.models.load_model('checkpoint.model.keras')

# evaluate model on test data
test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=1)

print(f"Test Loss: {test_loss}")
print(f"Test Accuracy: {test_accuracy}")
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - accuracy: 0.8457 - loss: 0.4327
Test Loss: 0.47418373823165894
Test Accuracy: 0.8199999928474426
```

Next, predictions were run using the test data, with the results represented by the following confusion matrix.

```
# make sample predictions
y_pred_probs = model.predict(X_test)
y_pred_probs = y_pred_probs.flatten()
y_pred_binary = (y_pred_probs > 0.5).astype(int)
y_pred_binary = y_pred_binary.flatten()

# visualize confusion matrix
confusion_matrix = tf.math.confusion_matrix(labels=y_test, predictions=y_pred_binary)

plt.figure(figsize=(6, 5))
sns.heatmap(confusion_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=['Predicted 0', 'Predicted 1'],
            yticklabels=['Actual 0', 'Actual 1'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix for Sentiment Analysis')
plt.show()
```

This matrix shows that the network had a slight tendency to overpredict positive sentiments as opposed to negative sentiments, but it is not a huge difference. When analyzing



the sequences that led to incorrect predictions, it appears that many of the sequences have <OOV> tokens, which suggests that meaningful words might have been dropped before being fed into the model. For example:

```
'<OOV> <OOV> <OOV> <OOV> macbeth <OOV> time sometimes look like <OOV> girl
<OOV> <OOV> blood evil'
```

In addition, some of the examples include conflicting positive and negative ideas, which may have led to the misclassification. For example, in the sequence below, several words or phrases sound positive ("totally recommend", "special") while others sound negative ("garbage"):

```
'scamp sing ive never feel <OOV> totally recommend <OOV> come special
<OOV> june <OOV> cover scamp garbage angel underneath <OOV>',
```

The screenshot below shows all 18 sequences (out of 100 total) that were misclassified by the network.

```
Decoded Mistakes:
['nothing short <OOV> <OOV> film',
 'simply excuse something poorly do',
 'strong <OOV> take stand exceptional act mostly <OOV> cast <OOV> script doesnt insult audience take easy way come white racism',
 'well keep think bad',
 'dont afraid <OOV> worth little <OOV> <OOV> tenzerotenzero',
 '<OOV> <OOV> <OOV> <OOV> macbeth <OOV> time sometimes look like <OOV> girl <OOV> <OOV> blood evil',
 '<OOV> plot <OOV> <OOV> <OOV> scenes <OOV>',
 'film redeem feature',
 'scamp sing ive never feel <OOV> totally recommend <OOV> come special <OOV> june <OOV> cover scamp garbage angel underneath <OOV>',
 'place good film garbage',
 'think <OOV> <OOV> fat better <OOV> cheesy clichés throw <OOV> <OOV> ready poorly <OOV> <OOV> <OOV> <OOV> <OOV> john <OOV>',
 'fact <OOV> <OOV> like <OOV> release',
 'movie lack visual interest drama expression feel celebration patriotism underline narrative',
 'scene debate whether sack trumpeter falsely accuse murder pure horror really stupid',
 'scene strong <OOV>',
 '<OOV> film fail level',
 'recommend friends',
 'bad everyone else involve didnt share <OOV> level dedication quality wed far better film hand <OOV> mess']
```

D5. Ethical Standards Compliance

According to Kazim and Koshiyama (2021), there are seven main themes under the umbrella of Trustworthy AI. These include human agency and oversight, safety, privacy, transparency, fairness, and accountability. Human agency revolves around the idea that AI should be used for the betterment of humankind while mitigating its potential risks. Safety is

focused on preventing harm in AI systems, such as preventing the misuse of a tool that was developed for a benign purpose from being used in the future for a malignant purpose. Privacy is concerned with protecting personal information, including keeping the amount of information that is collected and used to the bare minimum. Transparency is closely tied to communication. AI systems must be explainable in terms of how they work, engineers must be able to communicate about how decisions are made, and AI tools must be easily identifiable when they are in use. Fairness is focused on equity: AI tools and systems should minimize the chance for bias, and should be accessible for diverse populations. Finally, the theme of accountability concerns making sure systems and processes are well-documented, and tools are audited and evaluated on an ongoing basis to ensure fair use (Kazim & Koshiyama, 2021).

In terms of this analysis, efforts have been made to comply with these standards and mitigate bias. Since the dataset was provided and required, I was unable to confirm that there was no bias in the data collection, but an assumption is made that WGU had these standards in mind when choosing potential datasets. All reviews are anonymous, and no personal information is included in the dataset. This reporting clearly states the goals of the analysis and notes how the neural network developed and trained on the movie review data would be used by the production company (as a prototype to guide future development). Transparency and accountability are reflected by the choices made in determining fair splits of data for training, validation, and testing, and in tuning and evaluating the network prior to using it for classification on new inputs. The report transparently shows examples of the network's mistakes, which allows users to better understand its level of performance and potential risks in misclassifications. In addition, references are included in the reporting to support the tools used in building the network.

Part V: Summary and Recommendations

E. Code

The following code was used to save the best model (based on stopping criteria), and then reload the model with the optimal parameters.

```
# set early stopping criteria (including saving model)
model_checkpoint_callback = ModelCheckpoint(filepath="checkpoint.model.keras",
save_weights_only=False, monitor='val_loss', mode='min', save_best_only=True,
verbose=1)
early_stopping_callback = EarlyStopping(monitor='val_loss', mode='min', patience=7,
restore_best_weights=True, verbose=1)
callbacks = [early_stopping_callback, model_checkpoint_callback]

# fit the model on training data
history = model.fit(X_train, y_train, validation_split=0.0, validation_data=(X_val,
y_val), callbacks=callbacks, epochs=75, batch_size=16)

# load best model
model = tf.keras.models.load_model('checkpoint.model.keras')
```

F. Functionality

The bullet points below describe the functionality of the trained network, from input to output, when used to make binary sentiment classifications on a new piece of text.

1. **Input**: A new review sequence is the input. Before interacting with the model, the raw text should be pre-processed (as described in section B5) and entered into the network as a tokenized and embedded sequence of 20 elements.
2. **Embedding**: This layer converts each token (or word) in the input to a 16-dimensional vector to be used in the Convolutional Layer.
3. **Convolutional Layer**: This locally-connected layer applies 32 filters to the inputs to detect local patterns across adjacent words in the input. Because dropout is used, a portion of the feature maps will not be used in each run.

4. **Pooling**: This layer reduces the size of the input by half because it only keeps the strongest features or patterns from each region of the text.
5. **Long Short-Term Memory**: The bidirectional LSTM layer analyzes content before and after a point in the text to learn its context. It remembers the meaningful context and uses it to inform its outputs, as well as uses it to revise the context it remembers and guide what it can forget.
6. **Dense Layer**: The final layer is a fully-connected layer, which uses the sigmoid activation function to output the binary classification result at the final node.
7. **Output**: The final output is a float value between 0 and 1, which represents the probability that the text sample has a positive sentiment. A value of 0.5 and above can be interpreted as a prediction of a positive sentiment, while a value below 0.5 can be interpreted as a prediction of negative sentiment.

This network architecture supports the goals of the analysis because it leverages multiple types of neural networks to accomplish the task. The use of a convolutional block enables the network to use the local connections to find patterns in spatially close tokens of text, which is important for differentiating sentiments from phrases such as “not good” versus “exceptionally good”. In addition, the Long Short-Term Memory block allows the network to use context from farther distances ahead or behind in the text sample, which allows for better classification by detecting more complex patterns, especially with longer text samples.

G. Recommendations

As stated earlier, the goal of this project was to develop a prototype of a neural network that could be used to classify the sentiment of performance reviews (plays, musicals, musical acts, etc). As a prototype, it can be used as a proof of concept to show the production company what is possible in relation to sentiment analysis, demonstrate how it could be used to streamline their analysis of qualitative review data, and support faster decision-making, ultimately impacting the bottom line.

With an accuracy level of over 80%, I recommend presenting this prototype to the client as a demonstration of what can be achieved if they choose to invest in developing a training dataset better suited to their context of live performance reviews. It is likely that the current network could perform reasonably well on live performance reviews, but accuracy would be higher if the model were trained on domain-specific reviews related to theater and live music as opposed to Hollywood movies. Thus, with approval from the client, I recommend moving forward with the development of a similar neural network trained on a set of domain-specific review data.

Part VI: Reporting

H. Reporting

A PDF file of the Jupyter notebook is included with this submission.

I. Sources For Third-Party Code

The code for creating the CNN and LSTM architecture was strongly influenced by and adapted from the following courses:

Udacity. (n.d.). *Convolutional neural networks.* https://learn.udacity.com/paid-courses/cd1821

Udacity. (n.d.). *Advanced computer vision and deep learning.* https://learn.udacity.com/paid-courses/cd0361

The following documentation resources were used to develop new code, troubleshoot and debug, and better understand how the code works:

NLTK. (2009). *Natural Language Toolkit — NLTK 3.4.4 documentation*. Nltk.org. https://www.nltk.org/

scikit-learn. (2018). *sklearn.model_selection.train_test_split — scikit-learn 0.20.3 documentation*. Scikit-Learn.org. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

Module: tf | TensorFlow Core v2.4.1. (n.d.). TensorFlow. https://www.tensorflow.org/api_docs/python/tf

Team, K. (n.d.). *Keras documentation: Keras API reference*. Keras.io. https://keras.io/api/

J. Sources

Kazim, E., & Koshiyama, A. S. (2021). *A high-level overview of AI ethics. Patterns (New York, N.Y.), 2(9), 100314.* https://doi.org/10.1016/j.patter.2021.100314