DATA MINING II - D213
Task 2
Eric D. Otten
Student ID: 011399183

A1. Summarize one research question that you will answer using neural network models and NLP techniques. Be sure the research question is relevant to a real-world organizational situation and sentiment analysis captured in your chosen data set(s).

My research question for D213 Task 2 is “How accurately can a neural network model predict the sentiment (positive or negative) of customer reviews using NLP techniques?”

In the e-commerce industry, understanding customer sentiment from reviews can help companies improve their products and services. Accurate sentiment analysis can assist in identifying key areas of customer satisfaction or dissatisfaction, enabling companies to make data-driven decisions to enhance customer experience and increase sales.

By employing accurate sentiment analysis, companies can discern positive and negative feedback, which allows them to pinpoint strengths and weaknesses in their products or services. Accurate sentiment analysis reached through these models can directly lead to improved customer experiences and increased sales.

- Sentiment analysis has profound implications for public relations (PR). What would happen if companies used sentiment analysis to manage their public image more effectively? Some organizations may consider censoring negative reviews to maintain a positive online presence. However, a more ethical approach involves reaching out to dissatisfied customers privately, allowing them to vent their frustrations without publicly impacting the brand's reputation. This method ensures customer concerns are addressed competently, fostering a more loyal customer base. Stakeholders should not hesitate to recognize the value in proactive customer engagement strategies.

- Sentiment analysis isn't just for customers; it can also be applied to employees. Monitoring employee sentiment can help organizations make better decisions about retention and workforce planning. According to HRForecast, understanding employee sentiment is essential for effective workforce planning. By accurately analyzing employee sentiment, companies can identify areas of dissatisfaction early and implement changes to improve workplace morale and reduce turnover.

Additionally, sentiment analysis has several use cases outside of business contexts:

- Political parties use sentiment analysis to gauge public opinion on candidates, issues, and policies. Accurate sentiment analysis helps political entities tailor their messaging to resonate with different demographic segments, thereby optimizing their campaign strategies. With trillions of dollars and the future of global politics at stake, political parties and other non-governmental organizations certainly leverage the insights provided by sentiment analysis when creating political campaigns.

- Sentiment analysis also has applications in warfare. In a news segment on Ticker News earlier this year, I discussed how machine learning can be used to monitor and influence sentiment on social media regarding conflicts such as the Russia-Ukraine war or the Hamas-Israel war. Accurate sentiment analysis can inform psychological operations and strategic planning, ultimately impacting the outcome of conflicts.

- Similarly, sentiment analysis can be and has been used for propaganda purposes outside of warfare. Although the goals of propaganda are similar to public relations, and the two are closely linked through the work of Edward Bernays in the early 20th century, state propaganda can be far more impactful than corporate public relations. The Chinese government employs sophisticated sentiment analysis techniques to monitor public sentiment on social media platforms such as Weibo and WeChat. Once dissenting sentiments are identified, the Chinese government employs various strategies to suppress these voices. The state can

competently censor online content, remove critical posts, and block accounts that spread dissent. Additionally, individuals expressing dissent may be subject to offline repercussions such as surveillance, harassment, or arrest.

A2. Define the objectives or goals of the data analysis. Be sure the objectives or goals are reasonable within the scope of the research question and are represented in the available data.

I've chosen to divide my data analysis into a four-step plan for simplicity:

1. Preprocess and clean the Amazon review dataset
Preprocessing and cleaning the review text data are critical first steps in sentiment analysis. This involves removing punctuation, converting text to lowercase, and eliminating stopwords and irrelevant information. Accurately preprocessing the data ensures that the neural network model can focus on the most pertinent information, improving the overall quality of the analysis.

2. Build an train a neural network using TensorFlow
Building and training a neural network model involves selecting the appropriate architecture, such as convolutional neural networks (CNNs) or recurrent neural networks (RNNs), and feeding the cleaned data into the model for training. A well-designed and accurately trained model is competent in distinguishing between positive and negative sentiments in reviews.

3. Evaluate the model's performance using metrics and visualizations
Evaluating the model's performance involves using metrics to provide insights into how well the model is performing and highlight areas that may need improvement. Accurately assessing the model's performance ensures that it meets the required standards for deployment.

4. Run the model on new reviews to analyze sentiment
Once the model is built, trained, and evaluated, it can be used to analyze the sentiment of new customer reviews. Competently running the model on new reviews allows businesses to make data-driven decisions quickly.

The code for step 1 is included in Part B. Step 2 is included in Part C. Step 3 is included in Part D. Step 4 is included in Part F.

A3. 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.

I will be using a Recurrent neural network (RNN) for my text classification analysis. RNNs are typically used in ordinal or temporal problems such as text classification. The reason for this is that the word "recurrent" refers to the model's ability to take information from prior inputs to influence the current input and output. This allows a model to interpret each word within a broader context rather than by itself.

Additionally, I will use TenrorFlow's LSTM method to implement long short-term memory layers.

B1. Perform exploratory data analysis on the chosen data set, and include an explanation of each of the following elements

I chose to analyze the Amazon reviews dataset found within amazon_cells_labelled.txt.

Code

```
from collections import Counter
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from pylab import rcParams
```

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import train_test_split
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.preprocessing.text import Tokenizer
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import re
import string

plt.style.use("fivethirtyeight")
plt.rcParams["lines.linewidth"] = 1.5
dark_style = {
    "figure.facecolor": "#212946",
    "axes.facecolor": "#212946",
    "savefig.facecolor": "#212946",
    "axes.grid": True,
    "axes.grid.which": "both",
    "axes.spines.left": False,
    "axes.spines.right": False,
    "axes.spines.top": False,
    "axes.spines.bottom": False,
    "grid.color": "#2A3459",
    "grid.linewidth": "1",
    "text.color": "0.9",
    "axes.labelcolor": "0.9",
    "xtick.color": "0.9",
    "ytick.color": "0.9",
    "font.size": 12,
}
plt.rcParams.update(dark_style)

rcParams["figure.figsize"] = (18, 7)

# Load dataset
data = pd.read_csv('amazon_cells_labelled.txt', sep='\t', header=None, names=['sentence', 'score'])

# Check for unusual characters
def find_unusual_characters(text):
    return re.findall(r'[^a-zA-Z0-9\s,.!?\'\"-]', text)

# B1.  Perform exploratory data analysis on the chosen data set, and include an explanation of each of
the following elements
# presence of unusual characters (e.g., emojis, non-English characters)
# vocabulary size
# proposed word embedding length
# statistical justification for the chosen maximum sequence length

# - presence of unusual characters (e.g., emojis, non-English characters)
data['unusual_chars'] = data['sentence'].apply(find_unusual_characters)
unusual_chars = data['unusual_chars'].explode().unique()
print("Unusual characters found:", unusual_chars)

# - vocabulary size
all_words = ' '.join(data['sentence']).split()
vocab_size = len(set(all_words))
print("Vocabulary size:", vocab_size)
```

```
# - statistical justification for the chosen maximum sequence length
review_lengths = data['sentence'].apply(lambda x: len(x.split()))
max_seq_length = int(np.percentile(review_lengths, 95))  # 95th percentile
print("Maximum sequence length:", max_seq_length)
```

Output

```
Unusual characters found: [nan '+' '/' ':' ')' '(' '&' '$' '*' ';' '%' '#' '[' ']']
Vocabulary size: 2815
Maximum sequence length: 23
```

The presence of unusual characters in the dataset was identified using a regular expression to find characters outside the typical alphanumeric range. The unusual characters include punctuation marks, symbols, and possibly non-English characters. These characters can introduce noise into the text data, potentially affecting the model's performance. Cleaning or transforming these characters is a crucial preprocessing step to ensure data quality and model accuracy.

The vocabulary size was determined by tokenizing all the words in the dataset and counting the unique tokens. The vocabulary size represents the total number of unique words in the dataset. A larger vocabulary size indicates a more diverse set of words, which can improve the richness of the language model. However, it also means that the model must handle a larger feature space, which can increase computational complexity.

Word embedding length refers to the size of the dense vector representation for each word. I plan to implement a word embedding length of 100 as this is a common choice in NLP tasks. It strikes a balance between capturing sufficient word semantics and maintaining computational efficiency. The embedding length should be long enough to encode meaningful word relationships but not so long as to introduce unnecessary complexity.

The maximum sequence length was determined by calculating the 95th percentile of the review lengths in the dataset. This means that 95% of the reviews have a length of 23 words or fewer. Choosing the 95th percentile as the maximum sequence length ensures that the majority of reviews are fully captured while limiting the number of excessively long sequences. This approach balances the need to capture enough information from most reviews without overly increasing the model's computational load.

B2. Describe the goals of the tokenization process, including any code generated and packages that are used to normalize text during the tokenization process.

Code

```
# B2.  Describe the goals of the tokenization process, including any code generated and packages that
are used to normalize text during the tokenization process.
stop_words = set(stopwords.words('english'))
punctuation = set(string.punctuation)

def tokenize_and_normalize(text):
    tokens = word_tokenize(text.lower())
    tokens = [word for word in tokens if word not in stop_words and word not in punctuation]
    return tokens

data['tokens'] = data['sentence'].apply(tokenize_and_normalize)


max_seq_length = 23 # Determined from EDA
tokenizer = Tokenizer()
tokenizer.fit_on_texts(data['tokens'])
```

```
sequences = tokenizer.texts_to_sequences(data['tokens'])
```

The tokenization process is part of step 1 of my data analysis and involves the following substeps:

1. Define stopwords and punctuation. These sets are used to filter out common words and punctuation that do not add meaningful information to the model.

2. Tokenize the dataset. The function tokenize_and_normalize tokenizes the input text, converts it to lowercase, and removes stopwords and punctuation, resulting in a cleaned and normalized list of tokens. This allows us to focus only on meaningful content and ensure that text data is easily comparable.

3. Convert tokens to sequences. Initialize a Tokenizer object, fit it on the tokenized text to create a vocabulary index, and convert the tokens to sequences of numerical indices. The max_seq_length is used later for padding the sequences to a uniform length.

B3. Explain the padding process used to standardize the length of sequences. Include the following in your explanation:

Code

```
# B3.  Explain the padding process used to standardize the length of sequences. Include the following
in your explanation:
# if the padding occurs before or after the text sequence
# a screenshot of a single padded sequence

padded_sequences = pad_sequences(sequences, maxlen=max_seq_length, padding='post')
print("Padded sequence example:", padded_sequences[0])
```

Result

```
Padded sequence example: [137  87 442 443 159 695   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0]
```

The parameter padding='post' specifies that padding should be added after the text sequence. This means that if a sequence is shorter than the specified maximum length (max_seq_length), zeros will be appended to the end of the sequence until it reaches the desired length.

The maxlen parameter is set to max_seq_length, which is determined from exploratory data analysis (EDA) to be 23. This means every sequence will be adjusted to have exactly 23 elements.

The padded sequence output shows a sample sequence with zeros appended at the end to meet the required length.

B4. Identify how many categories of sentiment will be used and an activation function for the final dense layer of the network.

In my sentiment analysis task, I'm focused on classifying customer reviews into two distinct categories: positive and negative sentiment. This binary classification is fundamental to understanding customer feedback and making data-driven decisions to enhance customer experience. By accurately distinguishing between positive and negative reviews, we can pinpoint areas of customer satisfaction and dissatisfaction more effectively.

For this binary classification, the sigmoid activation function is the ideal choice for the final dense layer of the neural network. The sigmoid function outputs a probability value between 0 and 1. This output range is perfect for binary classification problems because it allows us to interpret the results as the likelihood of a review being positive. If the

model outputs a value closer to 1, it indicates positive sentiment, while a value closer to 0 indicates negative sentiment.

The code for this implementation is included in Part C1.

B5. Explain the steps used to prepare the data for analysis, including the size of the training, validation, and test set split (based on the industry average).

Preparing the data for analysis involves several key steps to ensure it is ready for training a neural network model. First, I tokenized the text data, converting each sentence into sequences of numerical indices using the Tokenizer class from Keras. This conversion is essential because neural network models require numerical input. Following tokenization, I padded the sequences to ensure they all have the same length, which is necessary for uniform input size across the dataset.

Next, I split the data into training, validation, and test sets. Using the train_test_split function from Scikit-learn, I initially split the data into a training set and a temporary set with an 80-20 split. This split is based on industry standards, where typically 80% of the data is used for training the model. Here, the temporary set accounts for 20% of the data, ensuring that we have a sufficient amount for both validation and testing.

After creating the training set, I split the temporary set further into validation and test sets, each comprising 50% of the temporary data. This results in 10% of the total data for validation and 10% for testing. The validation set is used during model training to tune hyperparameters and prevent overfitting, while the test set is reserved for evaluating the model's performance on unseen data. Specifically, my dataset consists of 800 samples for training, 100 samples for validation, and 100 samples for testing, which aligns with common practices in the industry to ensure robust model evaluation and reliable performance metrics.

By following these steps, I ensure that the data is properly prepared for training, validation, and testing, providing a solid foundation for building an accurate and reliable sentiment analysis model. This systematic approach to data preparation is crucial for achieving high-quality results and making data-driven decisions based on the model's predictions.

Code

```
# B5.  Explain the steps used to prepare the data for analysis, including the size of the training, validation, and test set split (based on the industry average).

X_train, X_temp, y_train, y_temp = train_test_split(padded_sequences, data['score'], test_size=0.2, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

print("Training set size:", len(X_train))
print("Validation set size:", len(X_val))
print("Test set size:", len(X_test))
```

Result

```
Training set size: 800
Validation set size: 100
Test set size: 100
```

B6. Provide a copy of the prepared data set.

See padded_sequences.csv, X_train.csv, X_test.csv, X_val.csv, y_train.csv, y_test.csv, and y_val.csv for each file respectively.

C1. Provide the output of the model summary of the function from TensorFlow.

The code below creates a machine learning model using the Sequential function from the TensorFlow library and outputs its summary.

Code

```
model = Sequential([
    Embedding(input_dim=vocab_size, output_dim=100, input_length=max_seq_length),
    LSTM(100, return_sequences=True),
    Dropout(0.5),
    LSTM(100),
    Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

early_stopping = EarlyStopping(monitor='val_loss', patience=2, restore_best_weights=True)
history = model.fit(X_train, y_train, epochs=15, validation_data=(X_val, y_val),
callbacks=[early_stopping])
model.summary()
```

Result

```
Epoch 1/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 4s 36ms/step - accuracy: 0.5122 - loss: 0.6936 - val_accuracy: 0.5700 - val_loss: 0.6897
Epoch 2/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.5254 - loss: 0.6865 - val_accuracy: 0.7600 - val_loss: 0.4854
Epoch 3/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.9057 - loss: 0.2969 - val_accuracy: 0.7700 - val_loss: 0.5375
Epoch 4/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.9487 - loss: 0.1660 - val_accuracy: 0.8100 - val_loss: 0.7660
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(32, 23, 100)	281,500
lstm_2 (LSTM)	(32, 23, 100)	80,400
dropout_1 (Dropout)	(32, 23, 100)	0
lstm_3 (LSTM)	(32, 100)	80,400
dense_1 (Dense)	(32, 1)	101

```
Total params: 1,327,205 (5.06 MB)
Trainable params: 442,401 (1.69 MB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 884,804 (3.38 MB)
```

C2. Discuss the number of layers, the type of layers, and the total number of parameters.

The model consists of several carefully chosen layers designed to effectively process and classify text data. Each layer represents a substep in building and training the neural network.

1. Embedding Layer: This layer converts input tokens into dense vectors of fixed size, with an output shape of (32, 23, 100) and 281,500 parameters. The embedding layer helps the model understand the context of words within the sequences, providing a rich representation of the input text.

2. First LSTM Layer: With an output shape of (32, 23, 100) and 80,400 parameters, this LSTM layer captures temporal dependencies in the data. The LSTM (Long Short-Term Memory) architecture is well-suited for handling sequences and remembering long-term dependencies, which is crucial for understanding the context in text data.

3. Dropout Layer: This layer, with an output shape of (32, 23, 100) and no parameters, helps prevent overfitting by randomly setting a fraction of input units to zero during training. This regularization technique ensures that the model generalizes better to unseen data.

4. Second LSTM Layer: This layer outputs a shape of (32, 100) with 80,400 parameters. It further processes the sequential data output from the first LSTM layer, allowing the model to capture more complex patterns and dependencies.

5. Dense Layer: The final dense layer, with an output shape of (32, 1) and 101 parameters, uses a sigmoid activation function to output a single probability value for binary classification. This layer helps in deciding whether the input text is of positive or negative sentiment.

The total number of parameters in the model is 1,327,205, with 442,401 trainable parameters. This combination of layers and parameters ensures that the model has enough capacity to learn from the data while also being efficient in terms of computation.

C3. Justify the choice of hyperparameters, including the following elements

The choice of hyperparameters in this model is grounded in common practices for natural language processing (NLP) tasks, ensuring that the model is both effective and efficient.

- Embedding and LSTM Layers: These layers use their default activation functions, which are suitable for capturing complex patterns and temporal dependencies in sequential data.

- Dense Layer: The sigmoid activation function is used in the final dense layer. This function is ideal for binary classification tasks as it outputs a probability between 0 and 1, making it easy to interpret the model's predictions as the likelihood of the input belonging to one of the two classes (positive or negative sentiment).

- LSTM Layers: Each LSTM layer has 100 units. This number of units allows the model to capture complex patterns and long-term dependencies in the text data, which are crucial for understanding sentiment.

- Dense Layer: The final dense layer has 1 unit, which is standard for binary classification tasks. This single unit outputs the probability of the input text being positive or negative.

The binary cross-entropy loss function is used for this model. This loss function is suitable for binary classification problems as it measures the performance of the model by comparing the predicted probabilities to the actual class labels, penalizing incorrect predictions more heavily.

You could find it beneficial to understand why the Adam optimizer is selected. The Adam optimizer is chosen for its efficiency and ability to handle sparse gradients on noisy problems. Adam combines the advantages of two other extensions of stochastic gradient descent—AdaGrad and RMSProp—making it well-suited for training deep neural networks.

Early stopping is employed to prevent overfitting, a critical aspect you should not overlook. The training process stops if the validation loss does not improve for a specified number of epochs, and the best weights are restored.

Accuracy is used as the primary evaluation metric. This metric is straightforward and measures the proportion of correctly predicted instances out of the total instances. For binary classification tasks like sentiment analysis, accuracy provides a clear indication of the model's performance.

D1. Discuss the impact of using stopping criteria to include defining the number of epochs, including a screenshot showing the final training epoch.

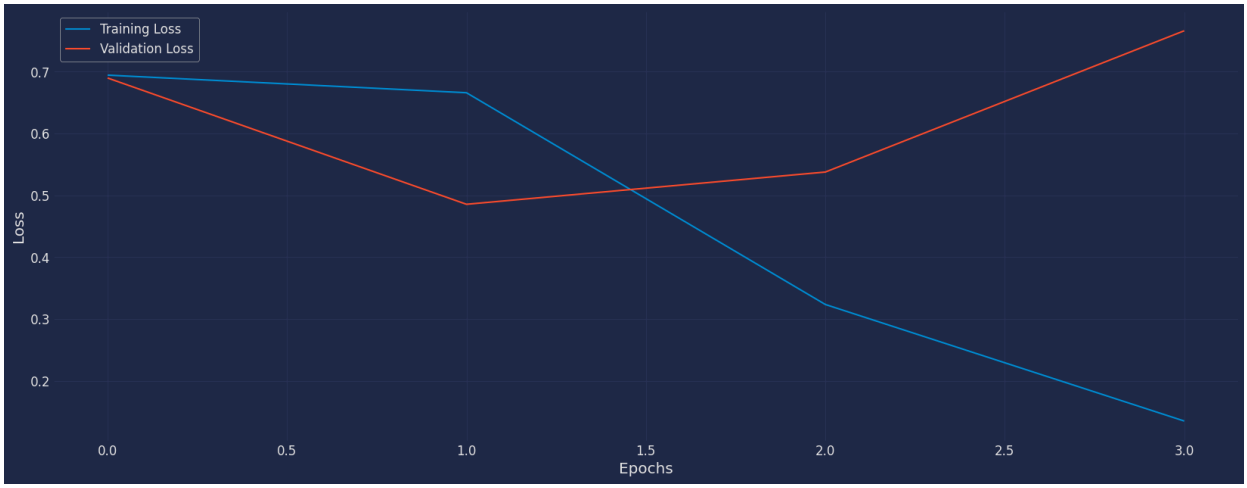
Code

```
# D1.  Discuss the impact of using stopping criteria to include defining the number of epochs, including a screenshot showing the final training epoch.
final_epoch = len(history.history['loss'])
print("Training stopped after {} epochs.".format(final_epoch))
```

Result

Training stopped after 4 epochs.

The following plot showing training and validation loss is generated in Part D3:

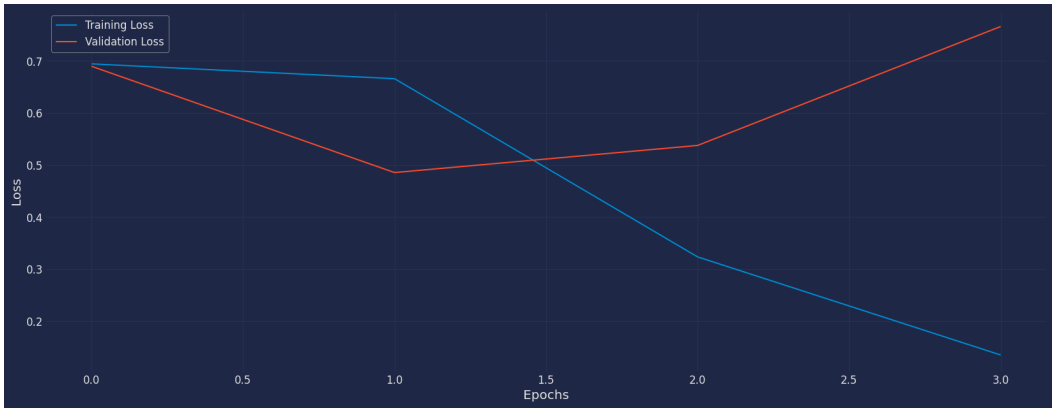


The training loss consistently decreases over the epochs, which is expected as the model learns from the data. A decreasing training loss indicates that the model is fitting the training data well.

Initially, the validation loss decreases, suggesting that the model is learning useful patterns that generalize to unseen data. However, after a certain point, the validation loss starts to increase, signaling the beginning of overfitting. The model starts to memorize the training data, including the noise, rather than learning the generalizable patterns.

Early stopping helps in capturing the optimal model performance on the validation set, avoiding the detrimental effects of overfitting. By stopping early, we ensure that the model maintains a balance between learning and generalizing, which is critical for its performance on new, unseen data. Any competent data analyst would agree that stopping at the right time can significantly improve the model's generalization ability.

D2. Assess the fitness of the model and any actions taken to address overfitting.

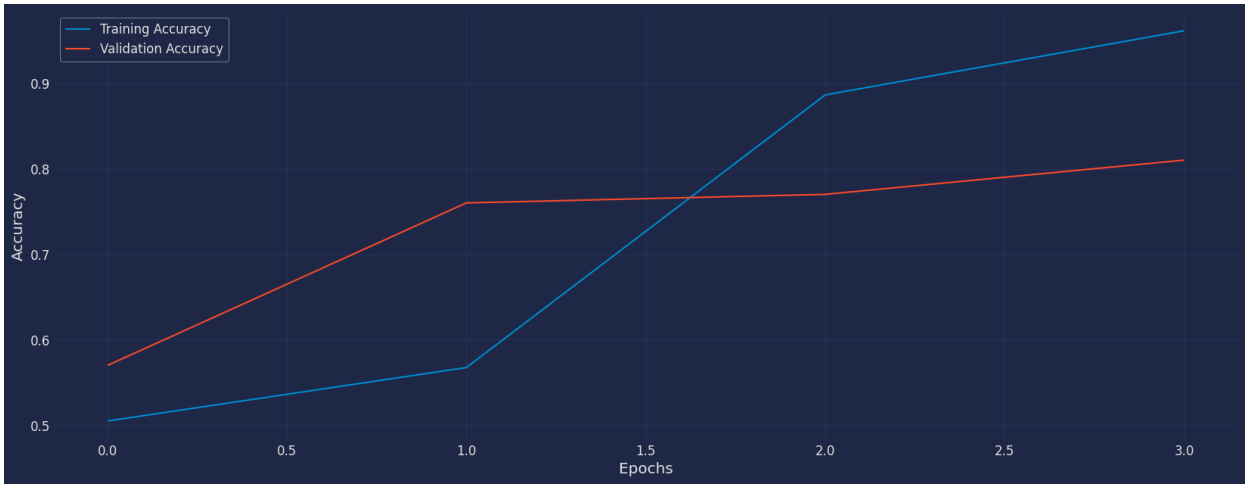


As described in Part D1, early stopping was employed to mitigate overfitting by halting the training process once the model's performance on the validation set stopped improving. This decision was based on the observation that while training loss consistently decreased, validation loss began to increase after a certain point, indicating the onset of overfitting.

By leveraging early stopping, competent data analysts ensure that the model retains a balance between learning and generalization. This helps in capturing the optimal model performance on the validation set and prevents the detrimental effects of overfitting.

Moreover, dropout layers were included in the model architecture to randomly set a fraction of input units to zero during training. This regularization technique helps in making the model more robust by preventing it from relying too heavily on any specific neurons, thus reducing overfitting.

The following plot showing training and validation accuracy was created in Part D3.



The plot provided shows the training accuracy (blue line) and validation accuracy (red line) over the epochs. Initially, both the training and validation accuracies start relatively low, with the training accuracy around 0.5 and the validation accuracy slightly higher. As the epochs progress, the training accuracy steadily increases, indicating that the model is effectively learning from the training data. By the time we reach about 1.5 epochs, the training and validation accuracy lines intersect, which signifies a pivotal moment in the model's learning process.

At this crossing point, the training accuracy surpasses the validation accuracy. This is an important observation because it typically marks the onset of overfitting. Before this intersection, the model is generalizing well, learning

patterns that apply to both the training and validation data. However, after this point, the validation accuracy begins to plateau and shows only a slight improvement, while the training accuracy continues to rise sharply. This divergence suggests that the model is starting to overfit to the training data, learning specific details and noise that do not generalize well to new, unseen data.

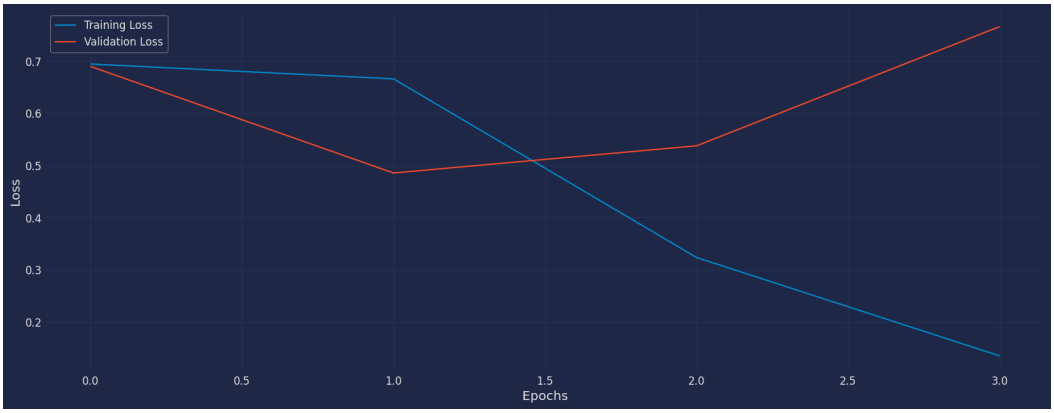
Overall, this plot demonstrates the importance of using techniques like early stopping, which was discussed previously. Early stopping can prevent the model from training too long and overfitting, ensuring it maintains its ability to generalize. In this scenario, observing the crossing of the training and validation accuracy lines around 1.5 epochs highlights the delicate balance between underfitting and overfitting, and the necessity for vigilant monitoring during the training process to achieve optimal model performance.

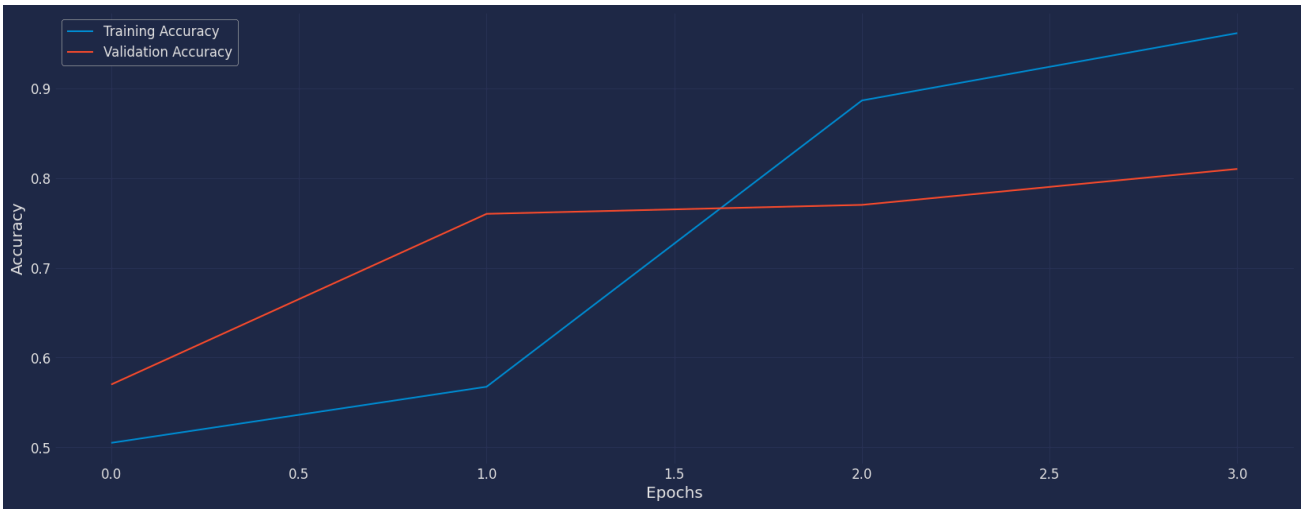
D3. Provide visualizations of the model's training process, including a line graph of the loss and chosen evaluation metric.

Code

```
# D3.  Provide visualizations of the model's training process, including a line graph of the loss and chosen evaluation metric.

plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

test_loss, test_accuracy = model.evaluate(X_test, y_test)
print("Test Accuracy:", test_accuracy)

model.save('sentiment_analysis_model.h5')
```

Result





Test Accuracy: 0.800000011920929

D4. Discuss the predictive accuracy of the trained network using the chosen evaluation metric from part D3.

The predictive accuracy of the trained network, as measured by the test accuracy, is 0.8000. This means the model correctly classifies 80% of the test instances, reflecting its competence in handling unseen data. For a sentiment analysis task, this level of accuracy is promising, suggesting that the model has effectively learned to differentiate between positive and negative sentiments based on the training it received.

Many competent data analysts might find it helpful to pause and reflect on the training and validation accuracy to understand the model's learning behavior better. As the accuracy plot shows, both training and validation accuracies initially increase, indicating that the model is effectively learning from the data. Around 1.5 epochs, the training and validation accuracy lines intersect. This intersection often marks a critical point where the model begins to overfit.

After this intersection, training accuracy continues to rise sharply, reaching above 0.9, while validation accuracy starts to plateau, stabilizing at around 0.81 by the fourth epoch. What would happen if the training continued without early stopping? The model might learn to capture noise and specific patterns in the training data that do not generalize well to new, unseen data.

It’s often useful to highlight that early stopping was employed to prevent overfitting. The early stopping technique ensured that training stopped at the optimal point where the model achieved the best validation performance, thus maintaining its generalization ability. How might the use of early stopping have influenced the final test accuracy? This approach helped achieve a balance between underfitting and overfitting, resulting in a robust model with a commendable test accuracy of 0.8000.

By reflecting on these points, competent data analysts can appreciate the importance of monitoring training and validation accuracies to optimize model performance. This thorough analysis underscores the significance of implementing strategies like early stopping to enhance the model's predictive accuracy and generalization capability.

E. Provide the code you used to save the trained network within the neural network.

See main.py.

F. Discuss the functionality of your neural network, including the impact of the network architecture.

The neural network used for sentiment analysis has demonstrated its functionality and effectiveness through its architecture and performance. The model comprises an embedding layer, LSTM layers, dropout layers, and a dense layer with a sigmoid activation function, designed specifically to handle text data and perform binary classification.

This architecture was applied to test new responses, producing the following outputs:

Code

```
# F.  Discuss the functionality of your neural network, including the impact of the network
architecture.
reviews = [
    "Absolutely love this product! It works wonders.",
    "Worst purchase ever, completely disappointed.",
    "Pretty decent, but I've seen better.",
    "An amazing buy, very satisfied with the quality!",
    "Did not meet my expectations at all.",
    "Okay product, not what I was hoping for.",
    "Incredible performance and great value for money.",
    "Not worth the price at all.",
    "Could be improved, lacks some important features.",
    "Best product I've used in years!",
    "Total waste of money, do not recommend.",
    "Fairly good, could be cheaper though.",
    "Love it! It’s exactly what I needed.",
    "Broke after one use, not durable.",
    "Surprisingly good for the price point."
]

# Assuming 'tokenizer' and 'max_seq_length' are loaded or defined elsewhere as they were during model
training
tokens = [tokenize_and_normalize(review) for review in reviews]  # Use the same normalization function
from your training
sequences = tokenizer.texts_to_sequences(tokens)
padded = pad_sequences(sequences, maxlen=max_seq_length, padding='post')

# Predict sentiments
predictions = model.predict(padded)
predicted_sentiment = ['Positive' if pred > 0.5 else 'Negative' for pred in predictions.flatten()]

# Output
results = pd.DataFrame({
    'Review': reviews,
    'Sentiment': predicted_sentiment
})

print(results)
```

Result

```
                                              Review Sentiment
0     Absolutely love this product! It works wonders.  Positive
1       Worst purchase ever, completely disappointed.  Negative
2                 Pretty decent, but I've seen better.  Positive
3    An amazing buy, very satisfied with the quality!  Positive
4                 Did not meet my expectations at all.  Negative
5            Okay product, not what I was hoping for.  Positive
```

```
6   Incredible performance and great value for money.  Positive
7                        Not worth the price at all.  Positive
8   Could be improved, lacks some important features.  Negative
9                    Best product I've used in years!  Positive
10           Total waste of money, do not recommend.  Negative
11             Fairly good, could be cheaper though.  Positive
12              Love it! It’s exactly what I needed.  Positive
13                 Broke after one use, not durable.  Negative
14            Surprisingly good for the price point.  Positive
```

While the model performs well in identifying positive and negative sentiments, there are a few false positives where the model incorrectly classified a negative sentiment as positive. Additionally, the binary nature of the model means it cannot account for neutral responses, which could provide a more comprehensive understanding of customer feedback.

Competent data analysts might find it useful to highlight these limitations when considering improvements. Addressing these aspects could lead to even more nuanced and accurate sentiment analysis, providing deeper insights into customer opinions and improving decision-making processes based on this feedback.

By leveraging this neural network architecture, the model effectively analyzes sentiment in text data, but there is always room for refinement to capture the full spectrum of sentiments expressed in customer reviews. Understanding these strengths and limitations helps in guiding future enhancements and achieving more robust sentiment analysis solutions.

G. Recommend a course of action based on your results.

Given that my dataset focuses on Amazon reviews, the course of action I would recommend would be tailored to Amazon sellers. The sentiment analysis model has demonstrated a commendable ability to classify customer reviews into positive and negative sentiments with an accuracy of 80%. However, there are a few areas for improvement and strategic applications that can greatly benefit sellers on Amazon.

Firstly, sellers should actively monitor the sentiment of their product reviews using this model. Positive reviews indicate areas where the product is meeting or exceeding customer expectations, providing valuable insights into strengths that can be highlighted in marketing efforts. On the other hand, negative reviews should be closely analyzed to identify common issues or areas for improvement. Addressing these concerns promptly can help improve customer satisfaction and reduce negative feedback in the future.

The insights gained from sentiment analysis can be used to refine product descriptions, images, and specifications to better align with customer expectations. Sellers can emphasize features that receive positive feedback and adjust or clarify aspects that customers find lacking. By making these data-driven adjustments, sellers can enhance the appeal of their product listings.

Proactively engaging with customers who leave negative reviews, as previously described in Part A1, is another crucial strategy. This approach can not only turn a negative experience into a positive one but also demonstrate a commitment to customer satisfaction, fostering brand loyalty and encouraging repeat purchases.

Additionally, a competent data analyst may wish to extrapolate these findings and apply them to sentiment analysis for purposes outside of Amazon e-commerce.

For companies looking to manage their online reputation, the model can be retrained to analyze sentiment across social media posts, news articles, and other online content. This can help in identifying and addressing negative sentiments quickly, potentially averting PR crises.

Organizations can use a similar model to monitor employee sentiment through internal surveys, emails, and feedback forms. By understanding the overall mood and concerns of their workforce, companies can take appropriate actions to improve employee satisfaction and retention.

Political parties can retrain the model on datasets containing public opinion on candidates, issues, and policies. This can help in tailoring campaign messages to resonate with different demographic segments.

In the context of warfare, sentiment analysis can be applied to social media data to gauge public opinion on conflicts. Governments and organizations can use these insights to understand the public mood and possibly influence narratives.

J. Web Sources

What are Recurrent Neural Networks, IBM – https://www.ibm.com/topics/recurrent-neural-networks

How can employee sentiment analysis improve your business? – https://hrforecast.com/the-importance-of-employee-sentiment-in-workforce-planning/

Text classification with an RNN – https://www.tensorflow.org/text/tutorials/text_classification_rnn

tf.keras.layers.Dense – https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense

K. Works Consulted

None