**Sentiment Analysis**

Bumjoo Huh

MS Data Analytics, Western Governors University

Advanced-Data Analytics — D213

Instructor: Dr. Festus Elleh

Oct 27, 2024

1. Purpose of Time Series Analysis
   1. Research Question
      1. How accurately can sentiment analysis classify customer feedback into positive and negative categories across multiple platforms using neural network models and natural language processing (NLP) techniques? The goal is to develop a model that can reliably predict sentiment polarity from customer reviews on Amazon, Yelp, and IMDb.
   2. Goal
      1. Train a sentiment classification model using labeled sentiment data from Amazon, Yelp, and IMDb, focusing on distinguishing between positive and negative reviews. Provide visualizations and insights that help stakeholders understand key patterns in customer feedback, including word frequency, sentiment distribution, and how these vary across different platforms.
   3. Type of Neural Network
      1. For this sentiment analysis task, a Recurrent Neural Network (RNN), specifically a Long Short-Term Memory (LSTM) network, will be utilized. LSTM networks are a powerful type of neural network architecture that excels at learning and making predictions based on sequential data, such as text.
2. Data Preparation
   1. EDA
      1. Presence of unusual characters
         1. During EDA, 17 unusual characters (e.g., emojis and non-English characters) in the dataset were identified. These characters are in the following indices: 1150, 1598, 1823, 1915, 2018, 2019, 2117, 2121, 2226, 2271, 2297, 2319, 2359, 2562, 2569, 2646, and 2715.
      2. Vocabulary Size
         1. The original vocabulary size was 5,272. After cleaning, this was reduced to 5,046 words, as non-English characters, punctuation, and irrelevant symbols were removed.
      3. Proposed Word Embedding Length
         1. The word embedding length is set to 100. This size is standard for representing meaningful word embeddings for the size of the words while balancing computational efficiency and accuracy.
      4. Maximum Sequence Length
         1. The maximum sequence length is statistically justified by the 95th percentile of sentence lengths, which is 14. This means that 95% of the sentences in the dataset have 14 or fewer words, making it an appropriate length for padding and truncation.
   2. Tokenization Process
      1. The tokenization process involves transforming the text data into sequences of integers. Each word is assigned a unique integer based on its frequency in the dataset.
      2. Goal
         1. The goal of tokenization is to convert the raw text into a format that can be fed into the neural network. Tokenization helps reduce the complexity of the input data and standardizes it for consistent processing.
      3. Code

from tensorflow.keras.preprocessing.text import Tokenizer

1. *# Initialize the tokenizer*

tokenizer = Tokenizer()

*# Fit the tokenizer on the text data*

tokenizer.fit\_on\_texts(combined\_df['text'])

*# Create the 'tokens' column (list of tokenized words)*

combined\_df['tokens'] = tokenizer.texts\_to\_sequences(combined\_df['text'])

combined\_df.head()

* 1. Padding Process
     1. After tokenization, padding ensures that all input sequences are of the same length. Padding standardizes the sequence length by adding zeros before or after.
     2. In this project, padding occurs after the text sequences to preserve the word order, and the maximum sequence length is set to 14, based on statistical analysis from EDA. Applying padding after the sequence is a more natural order for the input data.
     3. Code

*# Get sentence lengths*

combined\_df['sentence\_length'] = combined\_df['tokens'].apply(len)

*# Get basic statistics of sentence length*

print(combined\_df['sentence\_length'].describe())

*# Propose max sequence length based on 95th percentile*

max\_seq\_length = int(combined\_df['sentence\_length'].quantile(0.95))

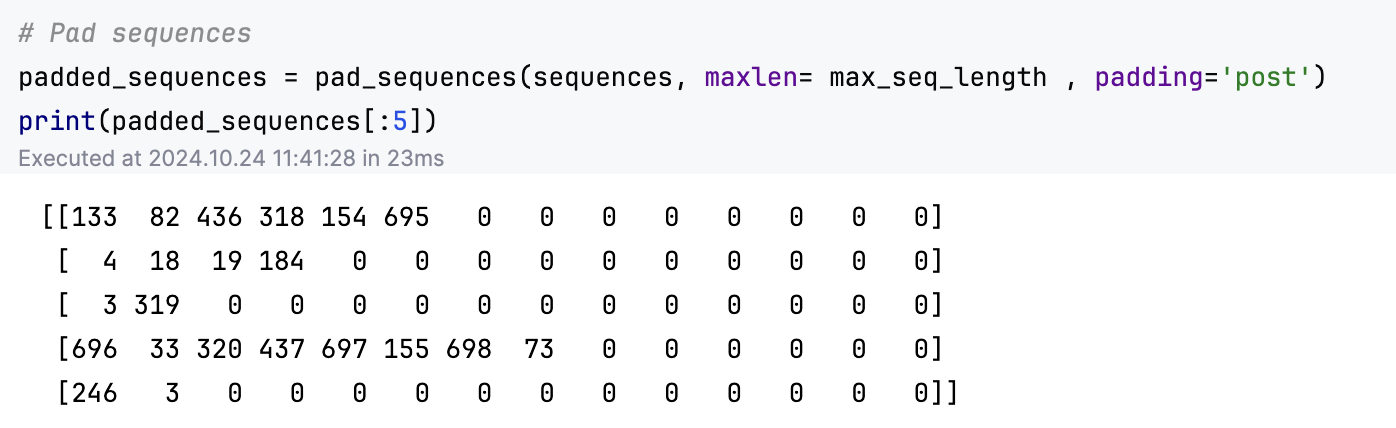
print(f"Proposed max sequence length (95th percentile): {max\_seq\_length}")

Max\_seq\_length

*# Pad sequences*

padded\_sequences = pad\_sequences(sequences, maxlen= max\_seq\_length , padding='post')

print(padded\_sequences[:5])

* + 1. Screenshot
  1. Categories of Sentiment
     1. The dataset contains two categories of sentiment: positive and negative. This makes it a binary classification task, with 1 representing positive sentiment and 0 representing negative sentiment.
     2. The final dense layer of the neural network uses a sigmoid activation function to output a probability between 0 and 1 for binary classification.
  2. Data Preparation for Analysis
     1. **Identification and Removal of Non-Alphanumeric Characters**: We identified sentences containing unusual, non-alphanumeric characters, which are often unnecessary for sentiment analysis. Using a custom function, we searched for non-ASCII characters to flag sentences with unusual symbols or formatting that might disrupt text processing:

*# Function to find non-alphanumeric characters*

def find\_unusual\_chars(text):

return re.findall(r'[^\x00-\x7F]+', text)

*# Apply the function and find sentences with unusual characters*

combined\_df['unusual\_chars'] = combined\_df['text'].apply(find\_unusual\_chars)

df\_unusual = combined\_df[combined\_df['unusual\_chars'].map(len) > 0]

* + 1. Conversion to Lowercase: All text was converted to lowercase to eliminate inconsistencies due to capitalization. This step helps prevent the model from treating words like "Good" and "good" as separate tokens, reducing vocabulary size and improving model efficiency:

def clean\_text(text):

text = text.lower()

* + 1. Removal of Punctuation and Non-Alphabetic Characters: Punctuation and special characters were removed to streamline the text and focus on relevant content. This was done using a regular expression that replaced any non-alphabetic character with a space, ensuring clean input for the model:

text = re.sub(r"[^a-zA-Z0-9]", " ", text)

* + 1. Stop Word Removal: We removed common English stop words (e.g., "and," "the," "is") using the Natural Language Toolkit (nltk). These words are typically frequent but add minimal semantic value to the model's understanding of sentiment:

nltk.download('stopwords')

stop\_words = set(stopwords.words('english'))

text = ' '.join(word for word in text.split() if word not in stop\_words)

* + 1. Elimination of Extra White Spaces: After removing stop words and punctuation, additional white spaces may remain. By joining the words with single spaces, we ensured a clean and consistent text structure:

text = ' '.join(word for word in text.split() if word not in stop\_words)

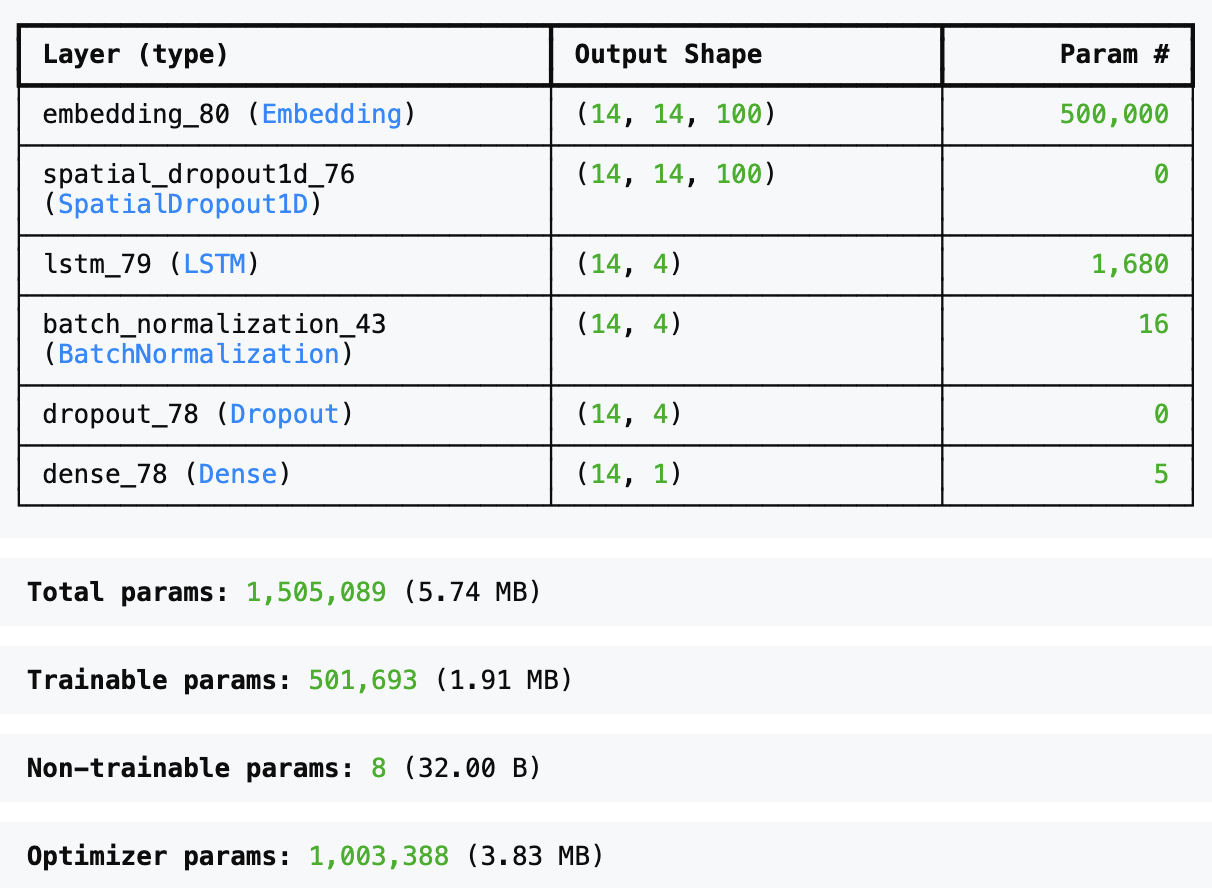
return text

* + 1. Implementation and Application: After defining the cleaning function, we applied it across all text entries in the dataset:

combined\_df['text'] = combined\_df['text'].apply(clean\_text)

* + 1. Dataset split: The dataset is split into training and test sets, with an 80%-20% split, which is standard in the industry.
    2. This split ensures that 80% of the data is used for training the model, while the remaining 20% is reserved for testing its performance on unseen data.
  1. Copy of Prepared Dataset
     1. X\_train.csv
     2. X\_test.csv
     3. Y\_train.csv
     4. Y\_train.csv

1. Network Architecture
   1. Model Summary Output



* 1. Discussion on Layer and Parameter
     1. Number of Layers 6
        1. Embedding Layer: This layer converts input words into dense vector representations of size 100. It maps each word in a vocabulary of size 5000 to a 100-dimensional vector. This shape output is (14, 14, 100), where 14 is the batch size and sequence length.
     2. SpatialDropout1D: This layer randomly drops entire 1D feature maps in the embedding space to reduce overfitting. It does not have any trainable parameters.
     3. LSTM Layer: This layer captures the sequential dependencies in the text, processing the sequences of words into a 4-dimensional output. It has 1680 parameters.
     4. BatchNormalization: This normalizes the outputs from the LSTM layer, helping to stabilize the training process and improve the model's speed and performance.
     5. Dropout Layer: This regularization technique randomly drops 50% of the units during training to prevent overfitting. It doesn't add any trainable parameters.
     6. Dense Layer: The final output layer contains one neuron for binary classification (positive or negative sentiment). It uses sigmoid activation to output probabilities. It has five parameters.
     7. Total Parameters: The model has 1,505,089 parameters. However, only 501,693 parameters are trainable, and eight are non-trainable (from BatchNormalization).
  2. Justification of Hyperparameters
     1. Activation Function
        1. Sigmoid Activation: The final dense layer uses a sigmoid activation function because it is a binary classification task. The sigmoid function outputs a probability between 0 and 1, ideal for distinguishing between two classes (positive or negative sentiment).
     2. Number of Nodes per Layer
        1. LSTM Layer (4 units): The LSTM layer has four units, a relatively small number of units to balance computational efficiency and prevent overfitting. A small LSTM can capture essential sequential information while reducing the risk of overfitting.
        2. Dense Layer (1 unit): The final dense layer has only 1 neuron since this is a binary classification task, with an output of either positive or negative sentiment.
     3. Loss Function
        1. Binary Cross-Entropy: The model uses the binary cross-entropy loss function, the standard for binary classification problems. It measures the difference between the predicted probabilities and the actual labels.
     4. Optimizer
        1. Adam Optimizer: The Adam optimizer is used because it combines the benefits of both RMSProp and Stochastic Gradient Descent (SGD). It adapts the learning rate during training, making it effective for most deep-learning tasks without requiring extensive tuning.
     5. Stopping Criteria
        1. Early Stopping: Early stopping is implemented using the EarlyStopping callback. This monitors the validation accuracy (val\_accuracy), and if the validation accuracy doesn't improve for five consecutive epochs (patience=5), trailing stops and the best weights are restored (restore\_best\_weights=True). This helps to avoid overfitting, as the model stops training when it no longer improves on the validation set.
     6. Evaluation Metric
        1. Accuracy
           1. The model achieved an accuracy of 81.0% on the test set. Accuracy is the ratio of correct predictions to the total number of forecasts, providing a general measure of the model’s performance.
           2. Test Accuracy: 0.809 (approximately 81%).
        2. Confusion Matrix
           1. True Positives (TP): 197
           2. True Negatives (TN): 248
           3. False Positives (FP): 38
           4. False Negatives (FN): 67
           5. This shows that the model is relatively good at correctly predicting positive and negative sentiments but makes some errors, particularly with false negatives (incorrectly predicting negative sentiment when it was actually positive).
        3. Precision, Recall, and F1-score
           1. Precision: Precision measures how many of the optimistic predictions made were correct.

Precision for Class 0 (negative sentiment): 0.79

Precision for Class 1 (positive sentiment): 0.84

* + - * 1. Recall: Recall measures how many positive examples were correctly predicted.

Recall for Class 0 (negative sentiment): 0.87

Recall for Class 1 (positive sentiment): 0.75

* + - * 1. F1-score: The F1-score is the harmonic mean of precision and recall, balancing both metrics.

F1-score for Class 0 (negative sentiment): 0.83

F1-score for Class 1 (positive sentiment): 0.79

* + - * 1. The model performs slightly better in predicting negative sentiment than positive sentiment, as indicated by the higher recall and F1-score for Class 0 (negative sentiment). The model performs reasonably well overall, with good precision and recall. The slight imbalance between the precision for Class 1 (positive sentiment) and recall for Class 1 suggests that the model is conservative in predicting positive sentiment but is generally reliable.

1. Model Evaluation
   1. Impact of using stopping criteria
      1. The initial training was set to 50 epochs, but during this run, the model achieved 92% training accuracy with a validation accuracy of around 78%, indicating signs of overfitting. This means that while the model performed well on the training data, it needed help generalizing to the validation set.
      2. To address this, we examined the epoch where the model performed best without significant overfitting. At epoch 22/50, the training accuracy was approximately 90%, and the validation accuracy was 81%. This showed a better balance between training and validation performance than the 50 epochs.
      3. Therefore, the number of epochs was reduced to 20, allowing the model to balance learning and generalization. Additionally, early stopping was applied with a patience of 5 epochs to prevent the model from continuing to train if the validation accuracy did not improve for five consecutive epochs. This ensured the model did not waste training time and avoided getting stuck at a plateau.  
         
   2. Assessment of Model Fitness
      1. **Training and Test Set Evaluation**:

The model achieved a **training accuracy of 94.68%** with a **training loss of 0.3496**, reflecting effective learning on the training data. For the test set, the model achieved an **accuracy of 80.91%**, demonstrating that it generalizes well to unseen data. The close alignment of training and test metrics indicates that the model was able to learn generalizable features without excessive memorization of the training data.

**Confusion Matrix**: The training data confusion matrix showed that the model effectively identified both positive and negative classes, with high recall for both classes (97% for class 0 and 93% for class 1). The classification report confirms balanced precision, recall, and F1-scores across classes, contributing to a high overall training accuracy.

* + 1. Early Stopping

An early stopping callback monitored validation accuracy, halting training when no improvement was observed over five epochs. This prevented the model from overfitting by stopping at the optimal point before training accuracy could surpass generalizable levels. Weights from the best-performing epoch on the validation data were restored, further supporting generalization.

*# Early stopping callback*

early\_stopping = EarlyStopping(monitor='val\_accuracy', patience=5, restore\_best\_weights=True)

* + 1. Embedding Dimension

The embedding dimension was increased from 10 to 100, creating richer representations that allowed the model to understand more nuanced word relationships. This change improved model performance, contributing to a higher validation accuracy without causing overfitting.

* + 1. Kernel Regularizer (L2 Regularization) and Dropout:

SpatialDropout1D and Dropout layers with dropout rates of 0.5 forced the model to generalize by deactivating 50% of neurons during each forward pass. This discouraged reliance on specific neurons and patterns, making the model more robust.

The L2 regularization value was increased from 0.01 to 0.1. This penalizes large weights, further helping the model to avoid overfitting by encouraging smaller weight values.

* + 1. Learning Rate Scheduler

A learning rate scheduler dynamically adjusted the learning rate over training epochs, reducing it by 50% at regular intervals. Lower learning rates in later epochs allowed the model to fine-tune weights without drastic updates, contributing to a smoother convergence and helping mitigate overfitting.

*# Define a function for the scheduler*

def scheduler(epoch, lr):

if epoch == 0: *# Every 5 epochs*

return lr \* 0.1 *# Reduce learning rate by 0.5*

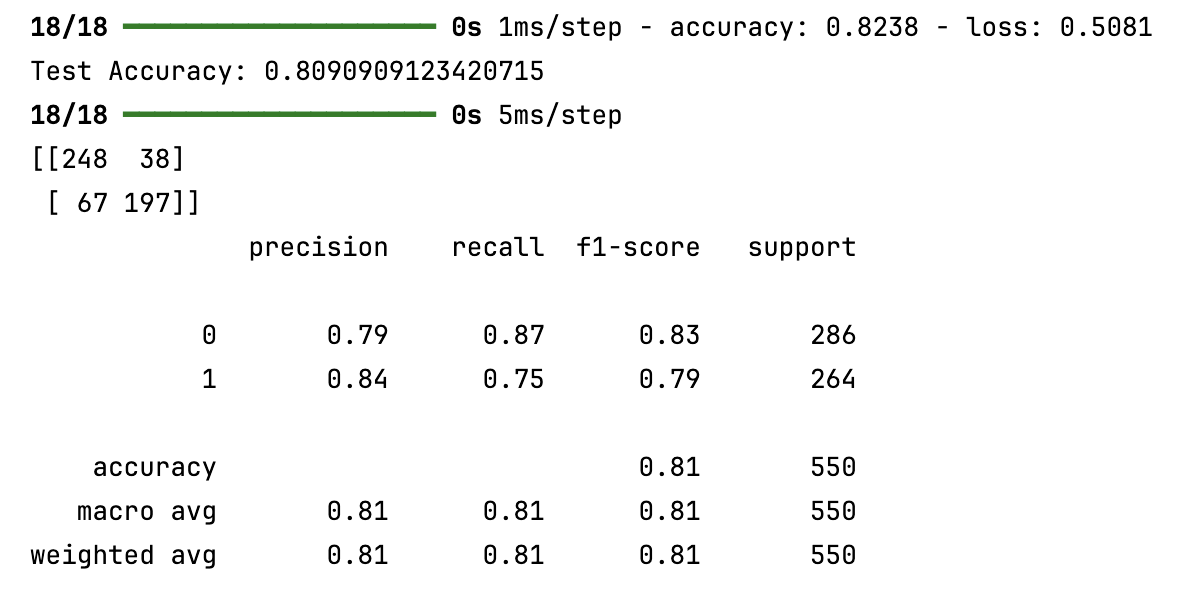
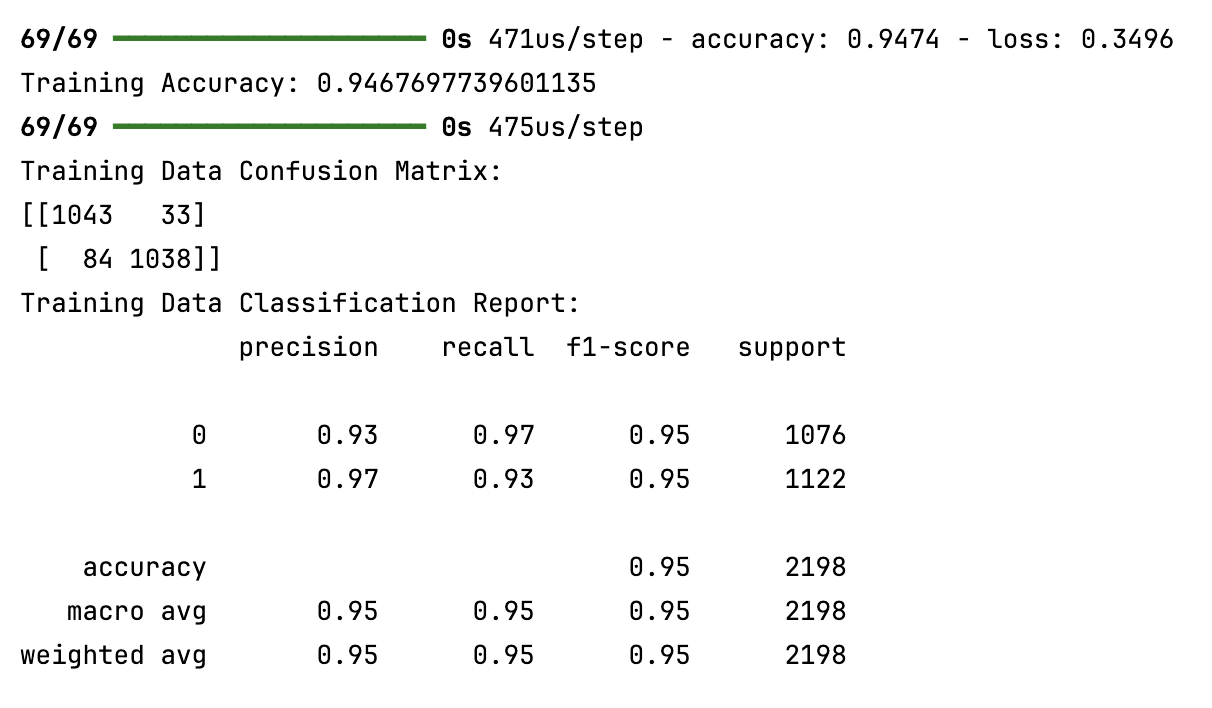
return lr

lr\_scheduler = LearningRateScheduler(scheduler)

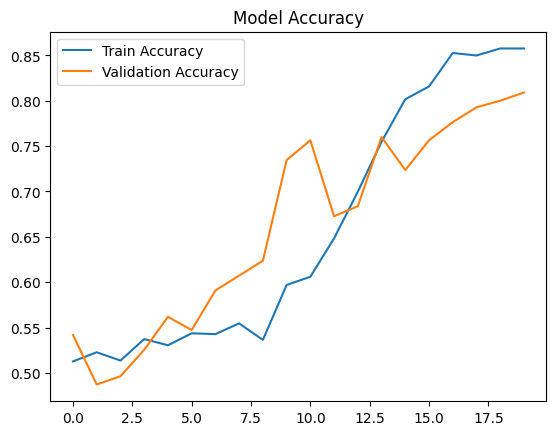
* + 1. Results of Overfitting Mitigation

The final evaluation metrics confirmed that the overfitting mitigation strategies were effective. The small gap between training accuracy (94.68%) and test accuracy (80.91%) indicates the model’s capacity to generalize. Regularization techniques, early stopping, and dynamic learning rate adjustments allowed the model to maintain strong performance on unseen data.

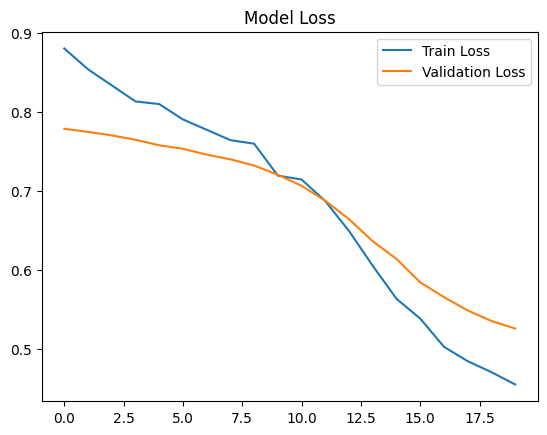
The balanced precision, recall, and F1 scores across classes in the training set support the model’s robustness. By preventing drastic divergence between training and validation losses, these measures collectively ensured optimal model fitness without overfitting.



* 1. Visualization of the model

MA

* + 1. Model Accuracy Plot: This plot shows the train and validation accuracy over each epoch. It indicates how accuracy improves over time, and the gap between training and validation accuracy helps assess overfitting.

ML

* + 1. Model Loss Plot: This displays the train and validation loss over each epoch. The convergence of the loss curves indicates that the model is learning effectively without significant divergence between training and validation loss.
  1. Predictive Accuracy
     1. The model achieved a test accuracy of 81%. This means that the model correctly classified 81% of the samples on unseen test data.
     2. Precision, Recall, and F1-Score were calculated for both classes (positive and negative sentiment):

Class 0 (Negative Sentiment)

* + - * 1. Precision: 0.79
        2. Recall: 0.87
        3. F1-Score: 0.83

Class 1 (Positive Sentiment)

Precision: 0.84

Recall: 0.75

F1-Score: 0.79

* + 1. The weighted average F1 Score of 0.81 indicates a balanced performance across both classes. The model shows a slight preference for accurately predicting negative sentiment (higher recall for Class 0) while maintaining strong precision for positive sentiment predictions.
    2. These metrics and the confusion matrix show that the model performs well in identifying both classes but can improve in reducing false negatives for positive sentiment.

1. Code

# Save the trained model   
model.save('sentiment\_model.keras')

1. Functionality
   1. Network Architecture Impact
      1. Embedding Layer: An embedding layer with a dimension of 100 helps transform words into dense vector representations, capturing semantic similarities between words. This allows the model to understand context and relationships within the text.
      2. SpatialDropout1D Layer: By applying 0.5 dropouts to the embeddings, this layer helps to prevent over-reliance on specific words or patterns, making the model more robust and reducing the risk of overfitting.
      3. LSTM Layer: The LSTM layer, with four units, captures temporal dependencies between words in a sequence. This is crucial for understanding the context in sentences where word order impacts sentiment, such as differentiating "not good" from "good."
      4. BatchNormalization: By normalizing the output from the LSTM layer, BatchNormalization stabilizes training and allows the model to converge faster by reducing internal covariate shifts.
      5. Dropout Layer: The additional dropout layer (0.5) further reduces the risk of overfitting by randomly dropping 50% of the LSTM outputs during training, encouraging the network to learn generalized patterns.
      6. Dense Layer with Sigmoid Activation: The final dense layer with a sigmoid activation outputs probabilities for binary classification, making it well-suited for the sentiment analysis task. The L2 regularization (0.1) helps control the model's complexity, ensuring that the weights do not grow too large, which could lead to overfitting.
   2. Performance Impact
      1. The architecture balances training and validation accuracy by incorporating regularization techniques like dropout and L2 regularization, leading to better generalization on unseen data.
      2. Early stopping ensures that training halts when further epochs do not improve validation accuracy, making the training process efficient and preventing overfitting.
      3. Overall, the neural network's design allows it to effectively learn and generalize from the training data, achieving satisfactory accuracy while minimizing the risk of overfitting. The choice of architecture is aligned with the complexity of the problem, ensuring a good balance between model complexity and performance.
2. Recommendation
   1. Model Optimization
      1. Consider fine-tuning the hyperparameters further, such as adjusting the dropout rates or LSTM units, to see if the model can better balance accuracy and validation performance.
      2. Experiment with larger embedding dimensions or pre-trained embeddings (e.g., GloVe or Word2Vec) to see if they boost performance by leveraging pre-learned semantic relationships between words.
   2. Data Augmentation
      1. Incorporating additional training data could help the model learn more robustly and improve generalization if more labeled data is available. This is especially important if the model encounters edge cases or uncommon phrases in the test data.
      2. Implement data augmentation techniques like synonym replacement or paraphrasing to increase the diversity of the training set artificially.
   3. Implement SEntiment analysis for Customer Feedback
      1. Use the trained sentiment analysis model to analyze customer feedback across various platforms such as surveys, social media, and product reviews. This will help the organization quickly understand customer sentiment trends without manually sifting through thousands of reviews.
      2. Automate the analysis of incoming feedback using the model to categorize it into positive or negative sentiments. This will enable the company to address negative feedback and capitalize on positive trends rapidly.
3. Reporting Jupyter Notebook
   1. Task\_2\_Sentiment\_Analysis.pdf

G. Third-Party Code Reference

Citations

Keras. (n.d.). *Regularizers*. Keras Documentation. Retrieved October 27, 2024, from<https://keras.io/api/layers/regularizers/>

TensorFlow. (n.d.). *tf.keras.callbacks.EarlyStopping*. TensorFlow Documentation. Retrieved October 27, 2024, from<https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/EarlyStopping>

TensorFlow. (n.d.). *tf.keras.layers.Embedding*. TensorFlow Documentation. Retrieved October 27, 2024, from<https://www.tensorflow.org/api_docs/python/tf/keras/layers/Embedding>

TensorFlow. (n.d.). *tf.keras.layers.LSTM*. TensorFlow Documentation. Retrieved October 27, 2024, from<https://www.tensorflow.org/api_docs/python/tf/keras/layers/LSTM>

TensorFlow. (n.d.). *tf.keras.layers.Dense*. TensorFlow Documentation. Retrieved October 27, 2024, from<https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense>

TensorFlow. (n.d.). *tf.keras.layers.Dropout*. TensorFlow Documentation. Retrieved October 27, 2024, from<https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dropout>

TensorFlow. (n.d.). *tf.keras.layers.SpatialDropout1D*. TensorFlow Documentation. Retrieved October 27, 2024, from<https://www.tensorflow.org/api_docs/python/tf/keras/layers/SpatialDropout1D>

TensorFlow. (n.d.). *tf.keras.layers.BatchNormalization*. TensorFlow Documentation. Retrieved October 27, 2024, from<https://www.tensorflow.org/api_docs/python/tf/keras/layers/BatchNormalization>

TensorFlow. (n.d.). *The Sequential model*. TensorFlow Documentation. Retrieved October 27, 2024, from<https://www.tensorflow.org/guide/keras/sequential_model>

TensorFlow. (n.d.). *tf.keras.utils.pad\_sequences*. TensorFlow Documentation. Retrieved October 27, 2024, from<https://www.tensorflow.org/api_docs/python/tf/keras/utils/pad_sequences>

TensorFlow. (n.d.). *tf.keras.preprocessing.text.Tokenizer*. TensorFlow Documentation. Retrieved October 27, 2024, from<https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/text/Tokenizer>

Bird, S., Klein, E., & Loper, E. (n.d.). *Corpus HOWTO*. Natural Language Toolkit (NLTK) Documentation. Retrieved October 27, 2024, from<https://www.nltk.org/howto/corpus.html>

Dr William Sewell. (n.d.).D213 Sentiment Analysis I. Panopto.Western Governors University. Retrieved October 27, 2024, from <https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=a1e9802c-5808-4287-a7e5-aed601580d89>

Dr William Sewell. (n.d.). D213 Sentiment Analysis Presentation 2. Panopto.Western Governors University. Retrieved October 27, 2024, from <https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=01eb7ceb-0a69-4cbb-a437-aee200d0e439>

Dr William Sewell. (n.d.). D213 Sentiment Analysis Presentation # 3. Panopto.Western Governors University. Retrieved October 27, 2024, from <https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=98eadb8d-b305-477a-9c1e-aee301358a6f>

H. References

Reference

Underfitted. (2023, September 28). *The Wrong Batch Size Will Ruin Your Model*. YouTube.Retrieved October 27, 2024, from<https://www.youtube.com/watch?v=dSqKGNT0qaw&t=202s>

StatQuest. (n.d.). *Word Embedding and Word2Vec, Clearly Explained!!!*. YouTube. Retrieved October 27, 2024, from<https://www.youtube.com/watch?v=viZrOnJclY0&t=619s>

StatQuest. (n.d.). *The Essential Main Ideas of Neural Networks*. YouTube. Retrieved October 27, 2024, from<https://www.youtube.com/watch?v=CqOfi41LfDw&list=PLblh5JKOoLUIxGDQs4LFFD--41Vzf-ME1&index=2>

StatQuest. (n.d.). *Tensors for Neural Networks, Clearly Explained!!!*. YouTube. Retrieved October 27, 2024, from<https://www.youtube.com/watch?v=L35fFDpwIM4&list=PLblh5JKOoLUIxGDQs4LFFD--41Vzf-ME1&index=22>