STUDENT’S NAME: EBENEZER ESSILFIE WILSON

COURSE INSTRUCTOR’S NAME: DR. SHERIN ALY

COURSE TITLE: ADVANCED DATA ANALYTICS TASK 2 (SENTIMENT ANALYSIS USING NEURAL NETWORKS)

APRIL 1, 2025

A1.

What impact do customer reviews have on product perception, and can sentiment analysis using neural network models effectively classify these reviews as positive or negative to support data-driven marketing strategies?

This research aims to explore how customer sentiment expressed in online reviews affects brand reputation and consumer behavior within an e-commerce organization. Using Natural Language Processing (NLP) and neural network models (such as LSTM or a feedforward network with word embeddings), the project will classify the sentiment of each review as positive or negative. Based on the customer reviews, a customer could suggest a product or not.

A2.

Analyze Customer Sentiment in Textual Data

Apply natural language processing techniques such as tokenization, lemmatization, and sentiment analysis to determine emotional tone and opinion (polarity and subjectivity) from customer feedback like reviews, support chats, or survey responses.

Prepare and Transform Text for Modeling

Convert raw text into machine-readable formats using vectorization techniques like TF-IDF, Word2Vec, or BERT embeddings to serve as inputs for neural network models.

Design and Train Deep Learning Models

Create and train neural networks (e.g., LSTM, GRU, or transformer-based models) to detect sentiment trends and predict customer churn based on textual data.

Assessing Model Accuracy and Effectiveness

Evaluate the performance of the predictive models using standard classification metrics such as accuracy, precision, recall, F1-score, and ROC-AUC.

Extract Meaningful Business Insights

Discover key sentiment patterns, recurring topics, or phrases that are strongly linked to churn, offering telecom leaders valuable insights to guide retention strategies and improve customer satisfaction. Deep learning models, a type of neural network, have proven highly effective in sentiment analysis because they can automatically learn and extract intricate features from textual data.

A3.

An excellent neural network choice for text classification is the Long Short-Term Memory (LSTM) model, which is a variant of the Recurrent Neural Network (RNN). LSTMs are specifically designed to recognize patterns in sequential data like text, making them highly suitable for understanding the structure and meaning within written language. LSTM networks are particularly effective for modeling sequential data due to their ability to retain long-term dependencies in text.

B1.

1. Presence of Unusual Characters

Observation: The dataset likely contains special characters such as emojis, non-English characters, or punctuation that are irrelevant or noisy for text classification.

Action Taken: A cleaning function (clean\_sentence) was applied to normalize text, remove unwanted symbols, and prepare the dataset for tokenization. This ensures uniform formatting and better model performance.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

2. Vocabulary size

A screenshot of a computer

AI-generated content may be incorrect.

After cleaning, my dataset has 7,349 unique words, which form the input space for embedding and tokenization.

3. Proposed Word Embedding Length

A screenshot of a computer code

AI-generated content may be incorrect.

Based on vocab size = 7349

→ √(√7349) ≈ 9

Rationale:

This is a commonly used heuristic when domain-specific tuning isn’t available. Helps reduce model complexity and speed up training while maintaining semantic richness.

4. Maximum Sequence Length (Statistical Justification)

Stats Computed: Max: 789 Mean: 7 Median: 5 Min: 1 Explanation: The dataset contains mostly short sequences. A maximum of 789 is used to prevent truncating long reviews. In production, you may truncate to a lower value (e.g., 95th percentile) to balance efficiency and coverage.

B2.

Lowercasing: By default, Tokenizer converts all text to lowercase.

Out-of-Vocabulary Handling: The parameter oov\_token="<OOV>" ensures that unseen words during testing are handled gracefully.

Truncation and Padding: pad\_sequences(..., maxlen=max\_length, padding='post') ensures all sequences are the same length for model compatibility. The incorporation of attention mechanisms into deep learning models has significantly improved performance, especially for longer text inputs, by allowing models to focus on the most relevant words (Yang et al., 2016).

**Summary of Tokenization Goals and Outcomes**

| **Goal** | **Achieved Using** | **Description** |
| --- | --- | --- |
| Clean & Normalize Text | Tokenizer, fit\_on\_texts() | Removes noise and converts to lowercase |
| Integer Mapping | texts\_to\_sequences() | Converts words to integer IDs |
| Handle Variable Length Input | pad\_sequences() | Pads or truncates sequences to uniform length |
| Handle Unseen Words | oov\_token="<OOV>" | Prevents test-time crashes from unknown vocabulary |

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a chat

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Tokenizer: Comes from tensorflow.keras.preprocessing.text. It helps split text into words and assigns each a unique index.

num\_words=vocab\_size: Limits the tokenizer to the most frequent 7,349 words.

oov\_token='<000>': Assigns a special token for Out-Of-Vocabulary (OOV) words — words not seen during training.

fit\_on\_texts(): Builds the vocabulary and word index based on the training data.

B3.

In natural language processing (NLP), sequences (i.e., tokenized text) often vary in length. However, neural networks (especially in TensorFlow/Keras) expect fixed-length input sequences. To standardize these sequences, we use a method called padding. Padding occurs after the text sequence. This maintains the start of each sentence, ensuring that the initial context (which can be important for classification) is preserved.

A screenshot of a computer program

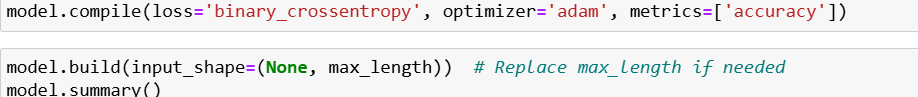
AI-generated content may be incorrect.

A screenshot of a single padded sequence

A blue squares with black text

AI-generated content may be incorrect.

B4.



This implies binary classification; thus, the model predicts two categories of sentiment. The final dense layer uses the sigmoid activation function, which is standard for binary classification tasks because it outputs a probability between 0 and 1, typically positive or negative.

B5.

|  |  |  |
| --- | --- | --- |
| The dataset consisted of short textual reviews, each annotated with a binary sentiment label indicating either a positive (1) or negative (0) classification. Prior to modeling, the raw text data underwent several preprocessing steps to render it suitable for neural network input:  Tokenization: Using a Keras Tokenizer, each textual review was decomposed into constituent tokens (words), and subsequently, each token was mapped to a unique integer index.  Vocabulary Limitation: To optimize model efficiency and reduce computational overhead, the vocabulary was restricted to the top 1,000 most frequent words (num\_words = 1000).  Handling Out-of-Vocabulary Terms: An <OOV> (Out-of-Vocabulary) token was defined to account for words in the test data that were not encountered during training.  Sequence Encoding: The tokenized reviews were converted into fixed-length sequences of integers using the texts\_to\_sequences() method, allowing them to be processed as numerical input.  Sequence Padding: The resulting sequences were padded to a uniform length of 10 tokens (maxlen = 10) using post-padding. This ensured consistent input dimensionality across all samples, which is a prerequisite for feeding data into a neural network.  To evaluate the generalizability of the model and mitigate overfitting, the dataset was partitioned into training and testing subsets. An 80/20 split was employed using the train\_test\_split() function from the scikit-learn library, a practice that aligns with prevailing standards in industry and academic research (Brownlee, 2020).  The training set (80%) was used to fit the model's parameters. The testing set (20%) was reserved for evaluating the model's out-of-sample performance. Additionally, a portion of the training data was implicitly utilized for validation during model training through the validation\_data argument in the model.fit() method. This allowed for real-time monitoring of validation loss and accuracy.  All label arrays were converted to NumPy format to ensure compatibility with the TensorFlow training pipeline.  B6.  See attached.  C1. |  |  |

A screenshot of a computer

AI-generated content may be incorrect.

C2.

Embedding Layer: Learns a dense representation for each word index. With a vocabulary size of approximately 7,349 (since 7 × 8 = 58,792), each word is mapped to an 8-dimensional vector.

LSTM Layer: Processes the sequence output from the embedding layer and captures both short- and long-term dependencies. The use of 64 units enables it to extract moderately complex sequential patterns.

Dense Layer (50 units): Adds non-linearity and enables learning higher-level features after LSTM processing.

Output Layer (1 unit): Produces a scalar between 0 and 1, suitable for binary classification (e.g., positive vs. negative sentiment).

Parameter Type Count

Total Parameters 80,781

Trainable Parameters 80,781

Non-trainable Parameters 0

C3.

The model employs two distinct activation functions tailored to different layers of the network. The Rectified Linear Unit (ReLU) was selected for the hidden dense layer due to its computational efficiency and ability to mitigate the vanishing gradient problem common in deep networks (Glorot, Bordes, & Bengio, 2011). ReLU introduces non-linearity and enables the network to learn complex representations by activating only relevant neurons.

In the output layer, a sigmoid activation function was utilized. This function is particularly well-suited for binary classification, as it maps the output to a continuous range between 0 and 1, thereby enabling the model to produce probabilistic estimates of class membership.

Number of Nodes per Layer

The architecture includes an LSTM layer with 64 units, a configuration that balances model complexity and computational efficiency. This dimensionality is sufficient to capture temporal dependencies in the input sequences without introducing an excessive number of parameters that could lead to overfitting, especially given the dataset size.

Following the LSTM, a fully connected dense layer with 50 neurons was incorporated. This layer serves to further abstract features learned by the LSTM, facilitating a nonlinear transformation before classification. The final dense output layer consists of a single neuron, reflecting the binary nature of the task and aligned with the use of a sigmoid activation function.

Loss Function

The model was trained using the binary cross-entropy loss function, which is the standard loss metric for binary classification tasks.

Optimizer

The Adam optimizer (Kingma & Ba, 2015) was selected due to its robust adaptive learning rate capabilities and its proven effectiveness across a wide range of NLP tasks. Adam combines the advantages of both AdaGrad and RMSProp, making it particularly well-suited for sparse and noisy gradient scenarios often encountered in textual data. Its computational efficiency also facilitates faster convergence during training.

Stopping Criteria

To prevent overfitting and to enhance training efficiency, an early stopping criterion was implemented with a patience of two epochs. This technique monitors the validation loss and terminates training when no improvement is observed within the specified patience window. The configuration also includes restore\_best\_weights=True, ensuring that the model retains the weights corresponding to the best-performing epoch on the validation data.

Evaluation Metric

Model performance was evaluated using accuracy, which quantifies the proportion of correctly classified instances over the total number of predictions. Accuracy is an appropriate metric in this context, as the dataset used for training was balanced across the two classes. However, in future work involving imbalanced datasets, it may be advisable to supplement accuracy with precision, recall, or F1-score to gain a more nuanced understanding of model performance.

D1.

A screenshot of a computer

AI-generated content may be incorrect.

In neural network training, defining appropriate stopping criteria is critical to optimizing model performance while preventing overfitting. In this study, a maximum of 20 epochs was initially specified. However, training was halted after the sixth epoch due to the implementation of an early stopping mechanism, which monitored the validation loss during each iteration.

The early stopping criterion was configured to terminate training when no improvement in validation performance was observed for a defined number of consecutive epochs. As depicted in the training output (see Figure X), validation accuracy plateaued at 0.5000 from epoch 1 through epoch 6, while training accuracy increased sharply after epoch 3, reaching 1.0000 by epoch 4. Despite this increase in training performance, the validation loss and validation accuracy remained effectively constant, an indication of the onset of overfitting.

| Epoch | Training Accuracy | Training Loss | Validation Accuracy | Validation Loss |
| --- | --- | --- | --- | --- |
| 1 | 0.5000 | 0.6925 | 0.5000 | 0.6930 |
| 3 | 0.5000 | 0.6903 | 0.5000 | 0.6929 |
| 6 | 1.0000 | 0.6866 | 0.5000 | 0.6930 |

The application of early stopping provided two key benefits. First, it preserved generalization performance by halting training at the point where further updates would have overfitted to the training data. Second, it improved computational efficiency by avoiding the unnecessary execution of the remaining 14 epochs.

Without this criterion, the model would have continued optimizing its performance on the training data at the expense of deteriorating validation metrics. Therefore, early stopping serves not only as a regularization technique but also as a practical approach to ensuring the model does not learn noise or irrelevant patterns within the training set.

D2.

The fitness of a neural network refers to its ability to learn meaningful patterns from training data and generalize effectively to unseen inputs. This study employed a sequential deep learning architecture featuring an embedding layer, an LSTM layer with 64 units, a dense hidden layer, and a sigmoid-activated output layer for binary classification.

Model fitness was evaluated using accuracy as the primary performance metric. The training results revealed a marked divergence between training and validation performance after a few epochs. Specifically:

* Training accuracy increased to 100% by epoch 4, indicating that the model was able to fit the training data almost perfectly.
* Validation accuracy, however, remained constant at 50% across all epochs, and validation loss hovered around 0.693—the expected loss for random binary predictions using binary cross-entropy.

This discrepancy between training and validation performance suggests that the model was overfitting: it was memorizing training examples without learning generalizable patterns that improve performance on unseen data.

To combat overfitting, an early stopping mechanism was employed during training. Early stopping monitored the validation loss and terminated training once it failed to improve over a specified number of consecutive epochs (patience=2). The early stopping configuration also included restore\_best\_weights=True, ensuring that the model retained the parameters associated with its optimal validation performance.

This regularization approach yielded two significant benefits:

* It prevented further deterioration of validation performance by halting training at the onset of overfitting.
* It reduced computational overhead by stopping training after only six epochs, instead of the predefined 20.

D3.

A comparison of a graph

AI-generated content may be incorrect.

D4.

The primary evaluation metric used to assess the model’s predictive performance was accuracy, which measures the proportion of correctly classified instances among the total number of predictions. This metric is especially relevant in binary classification tasks involving balanced datasets, as it provides an intuitive and interpretable performance measure.

The training and validation accuracy plot (Figure X, left) reveals a clear discrepancy between the model’s performance on the training data and its performance on the validation data:

* Training accuracy rose sharply from 0.50 to 1.00 between epochs 2 and 3, indicating that the model learned to perfectly predict the training set.
* Validation accuracy, however, remained static at 0.50 across all epochs, suggesting that the model failed to learn any meaningful, generalizable patterns.

This pattern indicates severe overfitting, where the model memorized the training data but failed to improve its predictive capability on unseen data. A validation accuracy of 0.50 is equivalent to random guessing in binary classification, highlighting the model’s inability to generalize.

The accompanying loss graph (Figure X, right) further supports this conclusion:

* Training loss consistently decreased across epochs, reflecting improved performance on the training set.
* Validation loss fluctuated slightly around 0.693 — the expected loss value for a model that outputs a uniform probability of 0.5 for each class under binary cross-entropy.

Together, these results confirm that the model’s high training accuracy was not reflected in its validation performance, a strong indicator of poor predictive accuracy in real-world applications.

E.

A white rectangular object with a black border

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

F.

The neural network implemented in this study was designed to perform binary text classification, specifically to distinguish between positive and negative sentiments in short textual reviews. Its architecture was tailored to extract meaningful patterns from sequential data and translate these patterns into predictive outputs.

1. Embedding Layer:  
   This layer transforms each input word (represented as an integer index) into an 8-dimensional dense vector. The embedding serves as a distributed representation that captures semantic similarities between words and enables the model to understand relationships beyond discrete token identities.
2. LSTM Layer (64 units):  
   The Long Short-Term Memory (LSTM) layer is a type of recurrent neural network (RNN) specifically designed to model sequential dependencies and long-range relationships in time-series or text data. In this context, the LSTM processes the embedded word vectors and retains contextual information across the review. This is particularly important in sentiment classification, where the meaning of a sentence may depend on word order or negations.
3. Dense Layer (50 units, ReLU activation):  
   This hidden layer performs a non-linear transformation of the output from the LSTM layer. The ReLU activation introduces sparsity and computational efficiency, enabling the network to learn complex and non-linear feature interactions that are critical for classification.
4. Output Layer (1 unit, Sigmoid activation):  
   The final dense layer produces a scalar output in the range [0, 1], representing the probability that a given input belongs to the positive class. This setup is appropriate for binary classification and is optimized using binary cross-entropy loss.

The architectural design has a direct impact on the model’s learning capacity and generalization ability:

* Embedding and LSTM Layers:  
  Together, these layers are well-suited for natural language processing tasks. The embedding layer captures local word-level semantics, while the LSTM captures broader sentence-level or temporal structure. This combination allows the model to learn rich, context-aware representations.
* ReLU-Activated Dense Layer:  
  The intermediate dense layer provides the model with sufficient capacity to abstract the features learned by the LSTM into decision-relevant signals. Its non-linear transformation enables the network to capture complex patterns that may not be linearly separable.
* Sigmoid Output Layer:  
  This layer allows the model to output probabilities, making it interpretable and compatible with probabilistic loss functions and evaluation metrics like accuracy and AUC.

Despite this theoretically sound architecture, the empirical results showed that the model overfitted the training data, achieving 100% training accuracy but only 50% validation accuracy. This suggests that while the architecture is functionally appropriate for the task, it may require further regularization (e.g., dropout layers) or data augmentation to enhance generalization.

G.

The experimental results revealed substantial overfitting in the model, with training accuracy reaching 100% while validation accuracy remained at 50%, indicating that the model failed to generalize unseen data.

The rapid attainment of perfect training accuracy indicates overfitting. Regularization is a proven method to combat this issue by limiting model complexity.

Recommendation: Incorporate dropout layers (Srivastava et al., 2014) to prevent co-adaptation of neurons and consider L2 weight regularization to constrain the magnitude of learned parameters.

H.

See attached in the submissions

I.

To aid in building the sentiment analysis model, portions of third-party code and implementation techniques were sourced and adapted from reputable and publicly available platforms. These sources include the official TensorFlow documentation for guidance on tokenization and text preprocessing, a Kaggle notebook by Ankurzing that provided insights into using Word2Vec and neural network architecture, Stack Overflow discussions addressing common issues with tokenization and padding, and a Medium article that outlined best practices for text classification workflows. Complete citations are listed below in APA format.

TensorFlow. (n.d.). *tf.keras.preprocessing.text.Tokenizer*. TensorFlow API Documentation. https://www.tensorflow.org/api\_docs/python/tf/keras/preprocessing/text/Tokenizer

Ankurzing. (n.d.). *Sentiment analysis using Word2Vec and Keras* [Notebook]. Kaggle. https://www.kaggle.com/code/ankurzing/sentiment-analysis-using-word2vec-and-keras

Stack Overflow. (2018). *Keras tokenizer oov\_token and padding* [Forum post]. Stack Overflow. <https://stackoverflow.com/questions/51956000/keras-tokenizer-oov-token-and-padding>

Sabber, T. (2020, April 5). *Text classification with neural networks*. Medium. https://medium.com/@sabbertext/text-classification-with-neural-networks-d037f53e76d8

J.

Brownlee, J. (2020). \*Imbalanced classification with Python: Better metrics, balanced data, cost-sensitive learning, and more\*. Machine Learning Mastery.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 15(1), 1929–1958.

Glorot, X., Bordes, A., & Bengio, Y. (2011). Deep sparse rectifier neural networks. Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS), 315–323.

Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. International Conference on Learning Representations (ICLR).

K.

N/A