Twitter/X sentiment analysis using RNN

SEP740 - Traditional Deep Learning Project

Group 6 - Jingjing Zhai, Junran (Claudia) Chen

Table of Contents

Abstract

This report discusses about the development and evaluation of a deep learning model for sentiment analysis on Twitter/X data. The project utilizes the sentiment140 dataset, including a balanced set of positive and negative tweets, and focuses on preprocessing raw text to remove noise, standardize language, and convert tweets into a numerical format suitable for modeling. The core of the analysis is an RNN model enhanced with Long Short-Term Memory (LSTM) units and an embedding layer, which together capture the sequential dependencies present in social media text.

Through systematic experimentation, the project explored various hyperparameters—including the number of hidden layers, LSTM units, embedding dimensions, and different activation functions—to optimize model performance. Regularization techniques, such as dropout and recurrent dropout, were applied to reduce overfitting, ensuring that the model generalizes better on unseen data. The final model configuration achieved a training accuracy of 84% and a test accuracy of 75%.

The report also presents detailed error analyses using confusion matrices and word cloud visualizations, providing insights into common misclassification patterns. It concludes with recommendations for future improvements, such as advanced preprocessing methods and data augmentation techniques, to enhance model robustness. Overall, the study provides a comprehensive examination of deep learning strategies for sentiment analysis.

1. Introduction

The topic of our project is to create a Recurrent Neural Networks (RNN) model that analyzes the sentiment expressed in tweets. Nowadays, Twitter (now rebranded as X) is one of the most popular social media platforms around the globe, where millions of users share their thoughts, opinions, and emotions in real time. Understanding the sentiment behind these tweets becomes a crucial task since the public's ideas could be discovered from those tweets. RNN models can handle sequential input as they have loops in the architecture, which makes it a great choice to analyze text data like tweets.

The significance of sentiment analysis on Twitter extends beyond mere academic interest—it has profound practical applications. For example, film studios can use this technology to gauge public reactions to their new movie, adjusting marketing strategies based on viewers' sentiment. Similarly, governments can monitor public opinion on new policies, ensuring that they remain responsive to the concerns and needs of citizens. Moreover, businesses can use sentiment analysis to manage brand reputation, predict market trends, and tailor customer service efforts.

To train our RNN model, we use a set of data that contain actual tweets extracted by the Twitter API, with each tweet labeled as either positive or negative based on the emotion it expressed. Our goal is to train a Deep Learning model that could decide the polarity of the tweets (positive or negative).The objectives we need to achieve include:

• **Preprocessing the Data:** We begin by converting the raw tweet data into a format that can be accepted by the model as input. This step involves standardizing the text by removing unnecessary information such as special characters, URLs, and stop words. The preprocessing also includes normalizing the language to ensure that the model focuses on the key elements of each tweet.

• **Training the Model:** With the processed tweets and their corresponding sentiment labels, the next objective is to train the RNN model. The neural network model will be created by functions in Keras package as it provide better optimization and faster running speed. The mission of the model is clear: perform binary classification by determining whether each tweet expresses a

positive or negative sentiment. The training process involves feeding the model these labeled examples so that it learns to map the tweet content to the correct sentiment.

• **Experimentation and Optimization:** An important part of our project involves conducting various experiments. We adjust meta parameters and introduce regularization techniques to refine the model's performance. By exploring different configurations, we aim to identify a model setup that provides the highest accuracy and learning efficiency. This approach ensures that the final model is robust.

• **Error Analysis and Future Improvements:** After the training phase, we focus on analyzing the model’s errors. This involves a detailed examination of the cases where the model misclassified the sentiment, allowing us to identify potential weaknesses. By discussing these shortcomings, we can propose future ways to enhance the model, such as refining the preprocessing steps or exploring alternative model architectures.

Each of these steps builds upon the last, creating a comprehensive workflow that not only trains an effective sentiment analysis model but also lays the groundwork for continuous improvement and deeper insights into the sentiment dynamics of tweets.

2. Methodology

A. Recurrent Neural Network

To analyze sentiment in tweets, we designed and trained a deep learning model based on a Recurrent Neural Network (RNN) architecture.

• **Recurrent Neural Network (RNN) & Long Short-Term Memory (LSTM):** Traditional RNNs update hidden states iteratively to capture contextual relationships across words in a sequence. However, their inability to retain long-term dependencies—due to vanishing gradients—poses challenges for analyzing tweets containing sarcasm, multi-phrase negations, or extended emotional expressions [1]. To address this, we integrated Long Short-Term Memory (LSTM) units, which utilize input, forget, and output gates to selectively preserve or discard information. This gating mechanism enables the model to maintain relevant context over longer text spans, making LSTMs particularly effective for social media language analysis [2].

• **Embedding Layer:** In addition to the LSTM layer, we included an embedding layer at the beginning of our model to transform each word into a dense vector representation. The purpose of the embedding layer is to reduce the dimensionality of the input text while preserving semantic relationships between words. It enables the model to capture syntactic and semantic information, which is especially helpful when working with large vocabularies [3]. Furthermore, we included fully connected hidden layers after the LSTM to help extract and combine features learned from the sequential output.

After hyperparameter tuning, we obtain a final model architecture beginning with an embedding layer with a vocabulary size of 10,000 and an embedding dimension of 128. This is followed by a single LSTM layer with 8 units, which captures sequential dependencies in the embedded tweet sequences. The output of the LSTM is then passed through two dense hidden layers, each comprising 16 units with parametric ReLU (PReLU) activation. The model concludes with a sigmoid-activated output layer for binary sentiment classification, determining whether a tweet conveys a positive or negative sentiment. The training process was conducted over 10 epochs using binary cross-entropy as the loss function, optimized with the RMSprop algorithm at a learning rate of 0.001. A mini-batch size of 64 was used to update model weights during training.

B. Data Processing

Before training our RNN model, we carried out several preprocessing steps to prepare the tweet dataset for effective learning.

• **Data Sampling:** To reduce computational cost and ensure a manageable dataset size, we randomly sampled 20,000 tweets from the original corpus.

• **Target Label Generation:** Each tweet in the dataset came with a sentiment label—either "positive" or "negative." These labels were mapped to binary targets, with positive mapped to 1 and negative to 0.

• **Data Pruning:** We applied a custom preprocessing method. This method handled negation standardization, removed user mentions, hashtags, URLs, special characters, and repetitive punctuation. The main objective of this pruning process was to remove uninformative tokens that would otherwise waste capacity in our tokenizer's word index.

• **Tokenization:** We initialized a Keras Tokenizer and fit it on the pruned dataset to build a word frequency table. After fitting, we inspected the top N (based on the word size parameter) most frequent words and confirmed that they reflected natural language and sentiment-bearing terms. Using the tokenizer, each tweet was then transformed into a sequence of integers, where each integer represented a token's frequency-based index in the tokenizer's vocabulary.

• **Padding:** Since tweet lengths varied greatly, we padded all sequences to a uniform length of 200. Sequences shorter than 200 tokens were pre-padded with zeros, while longer ones were truncated from the front. This ensured that all input data matched the expected input shape of the model.

C. Regularization

To address the risk of overfitting and improve the model's generalization capability, we explored several regularization techniques.

• **Batch Normalization:** Batch normalization was applied to normalize the activations of intermediate layers, aiming to reduce internal covariate shift and accelerate convergence. However, in our case, it showed limited impact on performance, possibly due to the relatively shallow architecture and small dataset.

• **Dropout:** Dropout was employed by randomly deactivating a fraction of neurons in the fully connected layers during training. This prevents units from co-adapting too much and encourages the network to learn more distributed and robust feature representations.

• **Recurrent Dropout:** Recurrent dropout, a variant of dropout specifically applied to the recurrent connections within the LSTM units, was also tested. This method helps regularize temporal dependencies by adding stochasticity to the memory updates, which is especially effective in recurrent architectures[4].

• **L1 & L2 Regularization:** L1 regularization encourages sparsity by penalizing the absolute value of the weights, potentially leading to simpler models. L2 regularization, on the other hand, penalizes the squared magnitude of weights to discourage extreme parameter values and promote smoother solutions.

After extensive experimentation, we found that the combination of dropout in the dense layers and recurrent dropout within the LSTM layer offered the most favorable balance between training performance and generalization to unseen data.

D. Model Evaluation – Learning curve and accuracy plotting

To evaluate the performance and generalization capability of our model, we partitioned the dataset into a training set (80%) and a test set (20%) using a stratified sampling approach to ensure the sentiment class distribution remained balanced across both subsets.

We plotted the resulting training and test loss curves, as well as the corresponding accuracy curves, for the entire training period. These visualizations provide intuitive insights into how well the model is learning over time and help identify potential training issues such as underfitting or overfitting.

• **Underfitting:** Underfitting is typically indicated when both training and test loss remain high and accuracy stays low across epochs, suggesting the model lacks the capacity or training time to capture patterns in the data.

• **Overfitting:** Overfitting occurs when training loss continues to decrease while test loss starts to increase, and the gap between training and test accuracy widens. This indicates that the model is fitting too closely to the training data and failing to generalize.

E. Error Analysis

To understand where our model struggled, we used two simple methods: confusion matrices and word clouds of misclassified tweets. These helped us see patterns in the mistakes and figure out how to improve the model.

• **Confusion Matrix:** We built a confusion matrix to compare the model's predictions with the actual labels. This table splits results into four categories: correctly predicted positive/negative tweets and incorrectly predicted ones. By looking at these numbers, we could notice if the model was biased toward one class [5].

• **Misclassified Samples Word Cloud Visualization:** We created word clouds from the misclassified tweets. We collected all the text from wrong predictions (e.g., negative tweets labeled as positive) and turned them into a visual cloud where bigger words meant they appeared more often. This showed us common words or phrases the model misunderstood [6].

3. Data

A. Description of the Dataset

The dataset we used is the sentiment140 dataset from Kaggle [7]. It contains 1,600,000 tweets extracted using the Twitter API. The tweets are labeled as 0 if they express negative emotion and as 4 if they express positive emotion. The dataset includes the following fields:

- **polarity**: The polarity of the tweet (0 = negative, 4 = positive)
- **id**: The tweet's unique identifier
- **date**: The date of the tweet
- **query**: The query; if there is no query, this value is "NO_QUERY"
- **user**: The user who tweeted
- **text**: The text content of the tweet

The dataset is balanced, with 800,000 tweets labeled as positive and 800,000 tweets labeled as negative. Examples of the dataset are as follows:

	polarity	id	date	query	user	text
1586564	4	2190822209	Tue Jun 16 03:49:24 PDT 2009	NO_QUERY	Alex_Clough	...went to New Emperor as it was still open V...
1593258	4	2191903014	Tue Jun 16 06:10:21 PDT 2009	NO_QUERY	LeahChantelle	@lickmycupcakes Yay!!! It worked!! And thank y...
250171	0	1983150308	Sun May 31 12:36:04 PDT 2009	NO_QUERY	YesDiva32	@Mister_82 don't b hatin on my Mira Mira music
774617	0	2321582236	Wed Jun 24 21:15:45 PDT 2009	NO_QUERY	_WorldsApart_	Where were u when #BPD broke 1,000 users onlin...
219640	0	1976442803	Sat May 30 17:20:29 PDT 2009	NO_QUERY	greatbrytton	I really just want to be out of Dothan...more ...
364854	0	2048346997	Fri Jun 05 15:03:16 PDT 2009	NO_QUERY	MsCherrylicious	Fml got suckered into working til 930 and I'm ...
1081037	4	1968400415	Fri May 29 21:21:32 PDT 2009	NO_QUERY	pennamico	@aalmishal ???? ?? ????? ????? ????? ?????? ?...
352747	0	2031275962	Thu Jun 04 09:35:12 PDT 2009	NO_QUERY	darlingroland	Today I could go get my new phone, probably wo...
804759	4	1468569254	Tue Apr 07 02:28:17 PDT 2009	NO_QUERY	Scyranth	@SexySubKaylee heh heh dirty is always good gu...
903760	4	1694686706	Mon May 04 03:43:12 PDT 2009	NO_QUERY	LmaonadeStand	No school today - Not because of "H1N1&qu...

Figure 1. 10 examples of sentiment140 dataset

B. Preprocessing Steps

To prepare the data, we randomly selected 20,000 entries from the dataset since the original dataset was too large for our equipment to run. To ensure that experiments could be repeated with the same set of samples, a random seed was applied. In the original dataset, the label 4 indicates positive emotion; for ease of understanding, we changed all instances of label 4 to 1. After these modifications, the sampled dataset contains 9,880 positive tweets and 10,120 negative tweets.

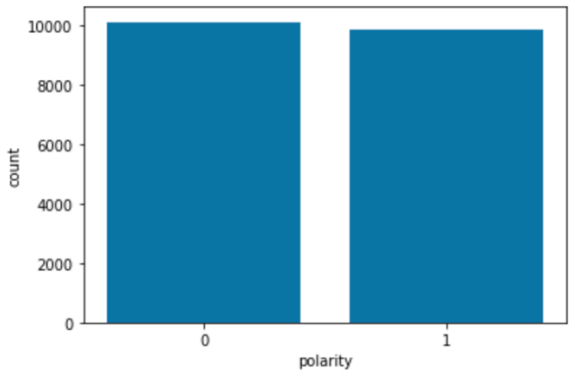


Figure 2. Histogram of counts of two polarities

After that, the id, date, user, and query columns were removed from the dataset, as they did not provide information relevant to the model's learning. Next, we processed the tweet texts to enhance model performance by applying the following steps:

- Convert all letters to lower case.
- Remove URLs from the text.
- Remove usernames (e.g., @SandyM) from the text.
- Remove non-alphanumeric symbols such as special characters and emojis, while keeping apostrophes (e.g., in “I’m” and “don’t”).
- Expand abbreviations for negative expressions into their full forms (e.g., "don’t" → "do not", "can’t" → "cannot", "won’t" → "will not", "ain’t" → "is not").
- Apply lemmatization to reduce verbs and nouns to their root form (e.g., "running" → "run", "better" → "good", "mice" → "mouse", "cars" → "car") [8].

These steps ensure that words with the same meaning are not learned as different words due to variations in their formats. In addition to these preprocessing steps, we considered removing stop words such as "the", "is", "at", and "and" as they typically do not provide information about the polarity of the tweets. However, this step was not applied because we observed that removing stop words lowered the test accuracy. This may be because RNNs are designed to model sequential dependencies, and the order and structure of words are crucial [9]. For example, removing stop words from a sentence like “I am not happy” to “I not happy” caused the RNN to lose the grammatical context necessary for understanding the sentence. Finally, we stored the processed tweets in the dataset.

	polarity	text	processed_tweets
1586564	1	...went to New Emperor as it was still open V...	go to new emperor a it be still open very nice...
1593258	1	@lickmycupcakes Yay!!! It worked!! And thank y...	yay it work and thank you blush
250171	0	@Mister_82 don't b hatin on my Mira Mira music	do not hatin on my mira mira music
774617	0	Where were u when #BPD broke 1,000 users onlin...	where be when bpd break 1000 user online 628pm...
219640	0	I really just want to be out of Dothan...more ...	really just want to be out of dothanmore than ...
364854	0	Fml got suckered into working til 930 and I'm ...	fml get suckered into work til 930 and back in...
1081037	1	@aalmishal ???? ?? ????? ????? ????? ?????? ?...	
352747	0	Today I could go get my new phone, probably wo...	today could go get my new phone probably will ...
804759	1	@SexySubKaylee heh heh dirty is always good gu...	heh heh dirty be always good gurl like how you...
903760	1	No school today - Not because of "H1N1&qu...	no school today not because of quoth1n1quot bu...

Figure 3. 10 examples of origin text and processed text

Then, the tweets were converted into sequences of integer indices based on the word index built by the tokenizer, since deep learning models ultimately work with numbers. While creating the tokenizer, we limited it to the top 10,000 most frequent words to reduce noise from uncommon words and improve model performance [10]. The model requires inputs of the same shape, so we set a maximum word length of 200. If a sequence exceeded this length, it was truncated from the beginning; if it was shorter, it was padded with zeros at the beginning [11]. This process resulted in an array of integers with the shape (20,000, 200).

```
[[   0    0    0 ...  105 7058 3774]
 [   0    0    0 ...   46    7 2727]
 [   0    0    0 ... 4802 4802  290]
 ...
 [   0    0    0 ...   46    7  105]
 [   0    0    0 ...    1  492  189]
 [   0    0    0 ...   78   17  269]]
```

Figure 4. The sequence of indices converted from the tweets

Eventually, the processed dataset was split to train data and test data with a test size of 0.2 with a specific seed. Now the data were available for training.

C. Data Normalization

By processing the tweets in the dataset, we performed basic normalization on the text data. The data used as input for the model were the indices generated by the tokenizer, and the embedding layer in the RNN model maps these indices to dense vectors for training. Therefore, after converting the text data into sequences, no additional data normalization was required.

4. Experiments, Results and Discussion

Our group conducted systematic experiments to optimize the model's performance by tuning key hyperparameters. Each experiment was evaluated using training accuracy and validation accuracy to identify the best balance between model accuracy and generalization.

A. Hidden Layer

In this experiment, we tested 16 combinations of hidden layers (1–4) and hidden units (4–32). As shown in the graphs, training accuracy generally increased with more layers and units, peaking around 16 units, but dropped slightly with 4 layers and 32 units. Test accuracy, however, varied more: too few or too many layers led to poor generalization. The best performance was achieved with 2 hidden layers and 16 units, reaching 95% training accuracy and 72.5% test accuracy.

The results show that deeper and wider networks generally improve training accuracy, but can hurt test performance due to overfitting, as the model may memorize noise rather than generalize well to unseen data. Conversely, networks with too few layers or hidden units may underfit, leading to low accuracy on both training and test sets.

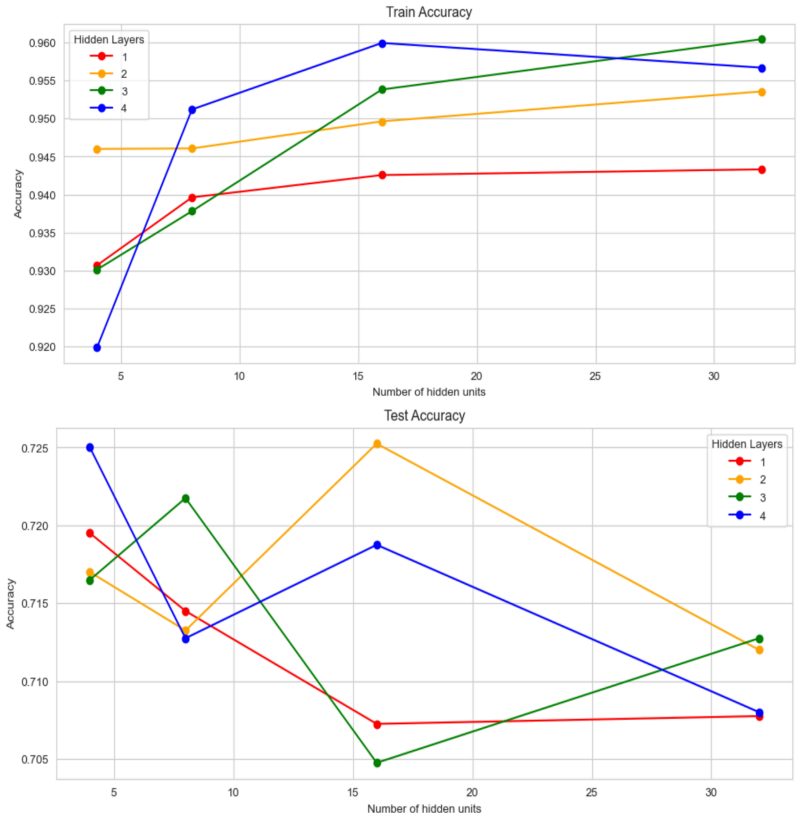


Figure 5. Comparison with different combinations of the number of hidden layers and hidden units

B. LSTM Layer

The experiment explored combinations of LSTM layers (1, 2, 3) and LSTM units (8, 16, 32, 64). As the number of units increases, both training and test accuracy generally followed a U-shaped trend—first decreasing and then increasing. With smaller units, fewer layers had higher test accuracy while maintaining strong training accuracy. When units exceed 16, deeper networks began to perform better on the test set. The best combination was 1 LSTM layer with 8 units, achieving 95.5% training accuracy and 72.2% test accuracy.

More LSTM units allow the model to capture richer sequential patterns. Meanwhile, stacking multiple LSTM layers helps the model learn deeper and more abstract temporal features.

This suggests that for short texts like tweets, smaller models can be both effective and efficient. Interestingly, deeper LSTM networks only began to show advantages when combined with more units, which also made them more complex and harder to train. We conclude that there's a trade-off: while deeper and wider LSTMs can potentially improve performance, they require careful tuning to avoid overfitting.

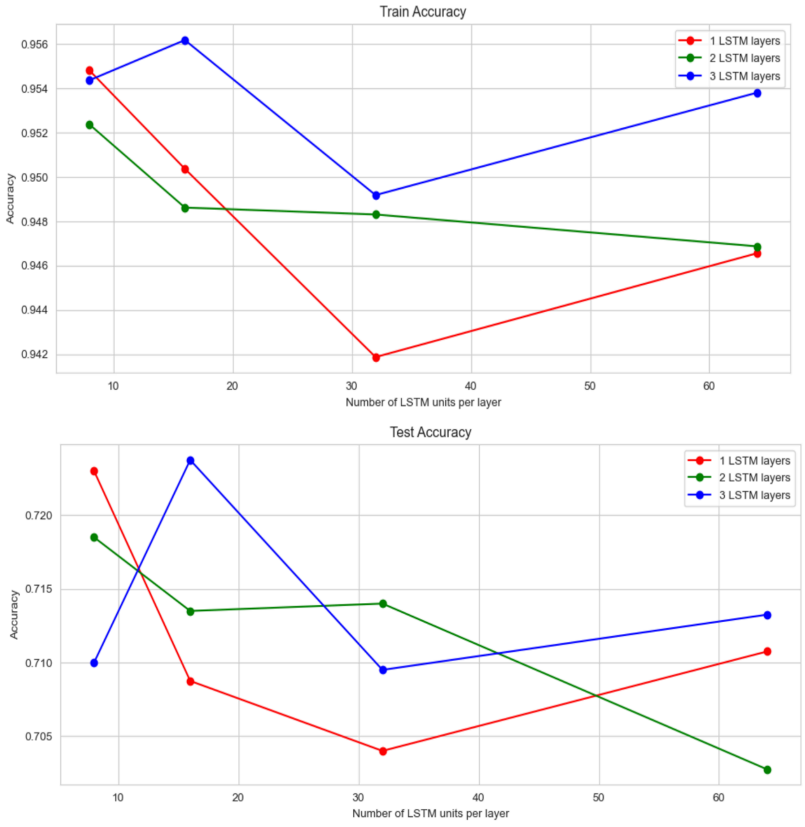


Figure 6. Comparison with different combinations of the number of LSTM layers and LSTM units

C. Embedding Layer

We explored the impact of different embedding configurations by varying the word size (word size = 1,000 to 10,000) and embedding dimensions (64 to 512). As expected, training accuracy generally increased with larger word sizes. When increasing the embedding dimension, training accuracy improved up to 256 and then plateaued. Test accuracy showed more complex behavior. With smaller word sizes, test accuracy dropped when increasing dimensions up to 128, then gradually recovered. In contrast, higher word sizes (5,000–10,000) benefited more from larger dimensions, especially in the 64–256 range. Beyond that, only the smallest (2,000) and largest (10,000) word sizes showed improvements. The best result was achieved with word size 10,000 and embedding dimension 128, reaching 96.5% training accuracy and 72.6% test accuracy.

Word size controls the number of distinct tokens the model can recognize. A larger word size helps capture more informative and diverse vocabulary, which is crucial for tweets where word choice varies widely. However, using too many rare or irrelevant words may introduce noise. Embedding dimension defines how much information each word vector can represent—higher dimensions enable richer representations but also increase the risk of overfitting. In our results, increasing the embedding size generally improved training accuracy, but test accuracy peaked when using 128-dimensional vectors. This suggests that overly large embeddings may not generalize well on limited data. Also, word sizes below 5,000 lost too much semantic information, while 10,000 struck the right balance between coverage and noise.

We conclude that using a moderate embedding dimension (128) with a large but reasonable word size (10,000) allows the model to learn expressive words for sentiment classification.

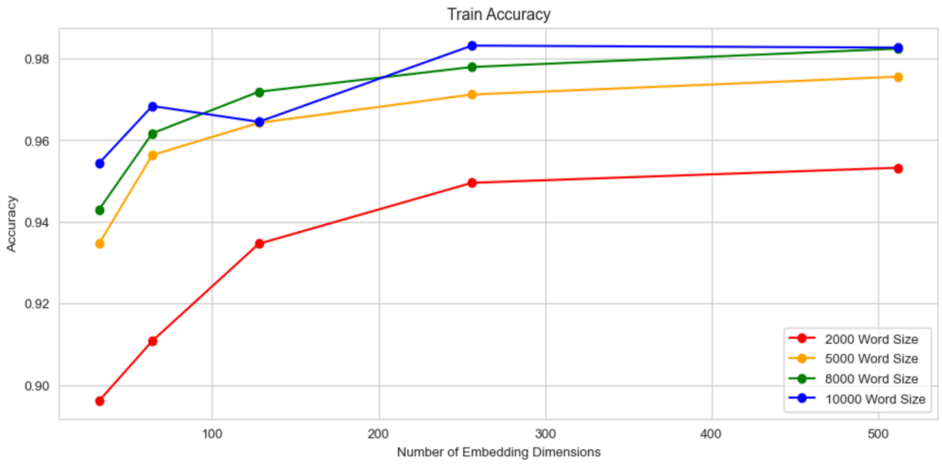


Figure 7. Comparison with different combinations of word size and the number of embedding dimensions

D. Activation Function

We tested various combinations of activation functions for the hidden and output layers. Overall, the output layer's activation functions (Sigmoid and Tanh) showed minimal differences in train and test accuracy, except for PReLU, which performed best with Sigmoid and worst with Tanh. For the hidden layer, activation functions under Sigmoid showed similar performance, with PReLU achieving the highest test accuracy. The best combination was PReLU for the hidden layer and Sigmoid for the output layer, achieving a test accuracy of 71.8% and a train accuracy of 97.8%.

In the hidden layer, PReLU stood out under Sigmoid, achieving the highest test accuracy due to its learnable parameters that adapt to data distribution. Under Tanh, activation functions like ReLU, LeakyReLU, and ELU had identical test accuracy but progressively lower train accuracy, indicating Tanh's symmetric range may not align well with these functions. In conclusion, PReLU in the hidden layer and Sigmoid in the output layer balanced adaptability and probabilistic output mapping to optimize performance.

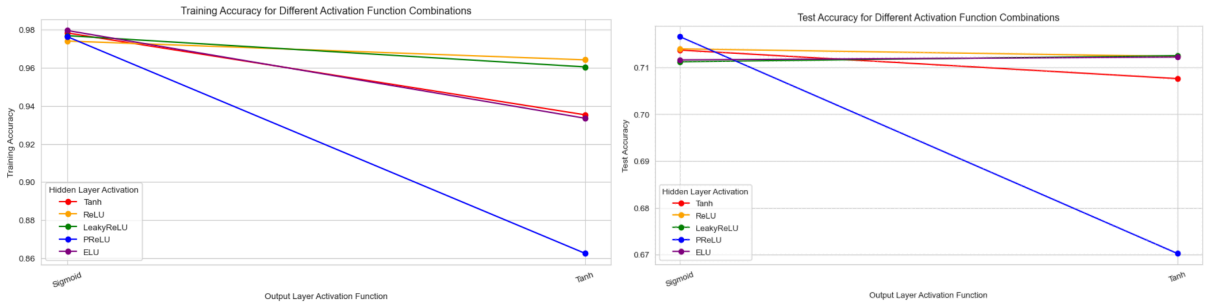


Figure 8. Comparison with different combinations of the activation functions in hidden layers and output layer

E. Mini-Batch Size

The mini-batch size affects not only the model's training speed but also its performance. We tested the current model using four different batch sizes: 8, 32, 64, and 128. After plotting the accuracies and training times, we found that the batch size of 64 resulted in the shortest training time. Although it did not produce the highest training accuracy, we selected 64 as the final batch size because the test accuracy it provided was relatively high. Prioritizing test accuracy over training accuracy helps improve generalization and reduce the risk of overfitting.

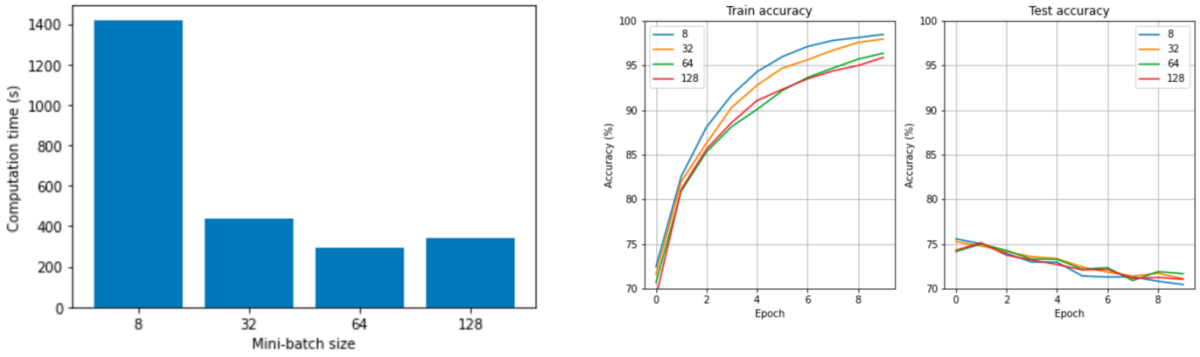


Figure 9. Histogram of running time of different batch size and learning curve

F. Optimization Function and Learning Rate

Choosing an appropriate optimization function and learning rate is crucial when building a deep learning model. The performance of some optimizers can be highly sensitive to the learning rate. To identify the best combination, we implemented a nested loop to test each pairing of optimizer—[SGD, RMSprop, Adam]—with learning rates—[0.0001, 0.0004, 0.001, 0.006, 0.02]. After running the experiments and plotting the final accuracies, we observed that RMSprop with a learning rate of 0.0004 yielded the highest test accuracy. However, it also resulted in a final training accuracy below 90%. To achieve a better balance between training and test accuracy, we selected RMSprop with a learning rate of 0.001 as the final configuration for this model.

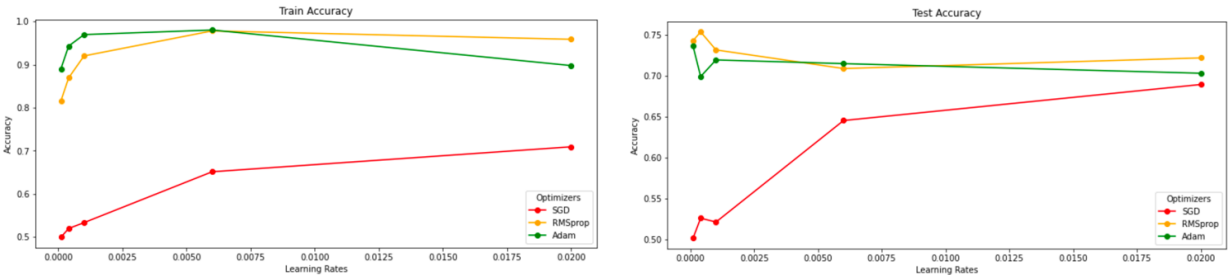


Figure 10. Comparison with different combinations of the optimizers and the learning rates

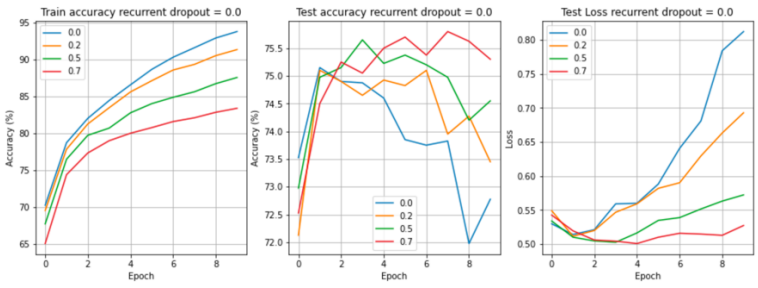
G. Regularization

At this stage, the model achieved a final test accuracy of approximately 73% and a training accuracy above 90%, indicating the presence of overfitting. To address this issue, regularization techniques were applied to the model.

Specifically, different values of dropout were tested. Since the model is based on an RNN architecture, we implemented both standard dropout and recurrent dropout, which randomly drops units in the recurrent connections of the LSTM layer at each time step [12]. To determine the optimal combination of dropout and recurrent dropout rates in LSTM layer, we experimented with dropout rates of [0, 0.2, 0.5, 0.7] and recurrent dropout rates of [0, 0.1, 0.3]—keeping the recurrent dropout generally lower due to its higher sensitivity.

After running these experiments and analyzing the learning curves for each configuration, we observed that although training accuracy was generally increasing across all settings, test accuracy often declined and test loss increased over epochs. This indicated the model was failing to generalize well. These issues were minimized when the regular dropout was set to 0.7 and the recurrent dropout to 0.1. This combination not only yielded the highest final test accuracy but also produced smoother and more stable learning curves.

Since we have a relatively small size of the dataset for sentiment analysis, overfitting was more likely to occur. In this context, a high dropout rate helped the model focus on the most significant patterns by forcing it to zero out a large number of neurons, thereby improving generalization and overall performance.



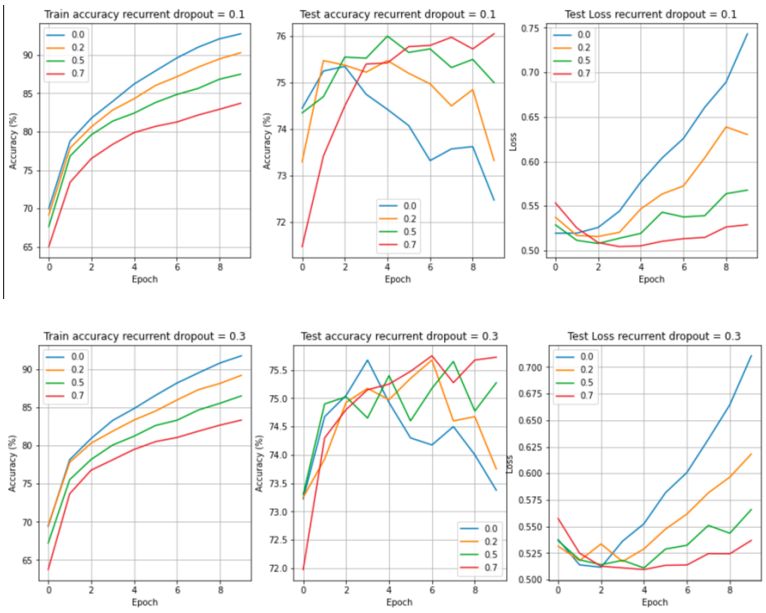


Figure 11. Comparison with different combinations of the dropout rates and recurrent dropout rates in LSTM layer

In addition to dropout regularization, we also experimented with L1 and L2 regularization to evaluate whether penalizing model complexity could further enhance performance. We applied L1 and L2 regularization with a lambda value of 0.001 to each layer individually and analyzed the results.

The experiments showed that L1 regularization could break the model and L2 regularization did not increase the performance of current model. This might happen because a high dropout regularization was already applied to the model, more regularization could harm the model's ability to learn from the data. As a result, neither L1 nor L2 regularization was added to the model.

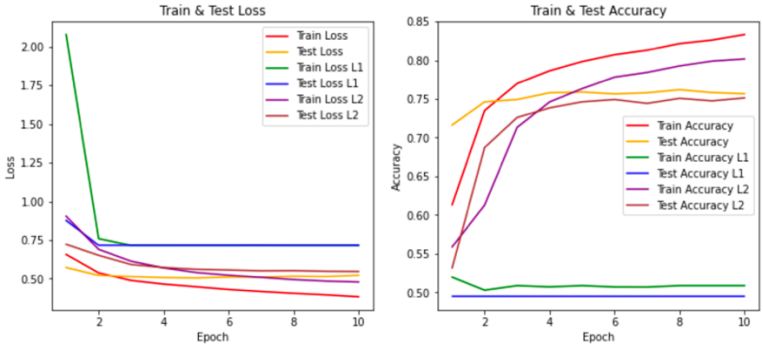


Figure 12. Comparison of accuracies with L1 and L2

H. Loss Function

Since the goal of our model was binary classification, Binary Cross-Entropy (BCE) was the most common and appropriate choice for the loss function. In addition to standard BCE, another option is BCE with logits, which combines the sigmoid activation function with BCE loss. This method offers improved numerical stability and can help address vanishing gradient issues by mitigating the effect of extreme logit values.

However, based on our current learning curves, the training loss consistently decreased, indicating that vanishing gradients were not a concern. Moreover, there were no signs of numerical instability or extreme output values. Therefore, the use of BCE with logits was unlikely to address the primary issue our model faced—low generalization.

As demonstrated in the experimental results, BCE with logits did not lead to performance improvements. Consequently, we retained the standard Binary Cross-Entropy as the loss function for our final model.

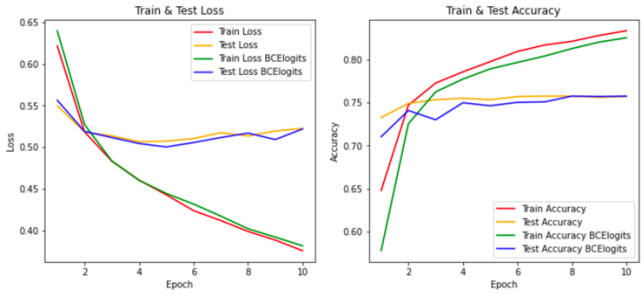


Figure 13. Comparison of accuracy with standard BCE loss and BCE with logits

I. Batch normalization

As a form of regularization, batch normalization reduces the risk of overfitting. Therefore, we did experiment to test whether use batch normalization improve the model. However, the result showed that batch normalization didn't benefit our model and only provided a similar accuracy. The reason why this happen might be the batch normalization computed mean and deviation for each batch but did not consider the recurrent part of RNN model. As a result, batch normalization was not applied in our final model.

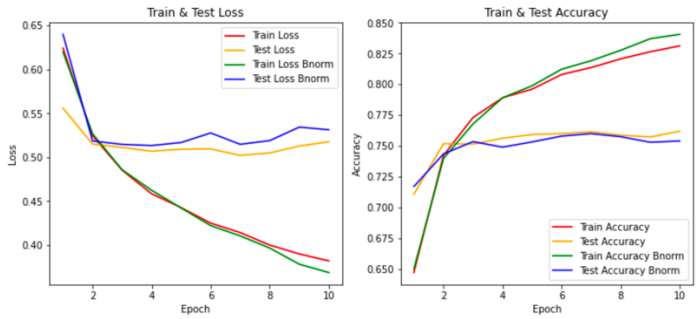


Figure 14. Comparison of accuracy with and without batch normalization

J. Final Model Evaluation

As training epochs increased, the train loss consistently decreased, indicating that the model effectively learned from the training data. However, the test accuracy plateaued after the second epoch, stabilizing around 75%. Similarly, the train accuracy followed a steep curve, eventually reaching 84%, while the test accuracy remained lower, highlighting a gap between training and testing performance.

The results suggest that the model may be overfitting the training data. The consistent decrease in train loss and high train accuracy indicates strong learning on the training set, but the stagnant test accuracy implies limited generalization to unseen data. This could be due to insufficient regularization or a lack of diversity in the training data. To address this, we could explore techniques like dropout, early stopping, or data augmentation. Overall, while the model performs well on the training set, improving its generalization remains a key challenge.

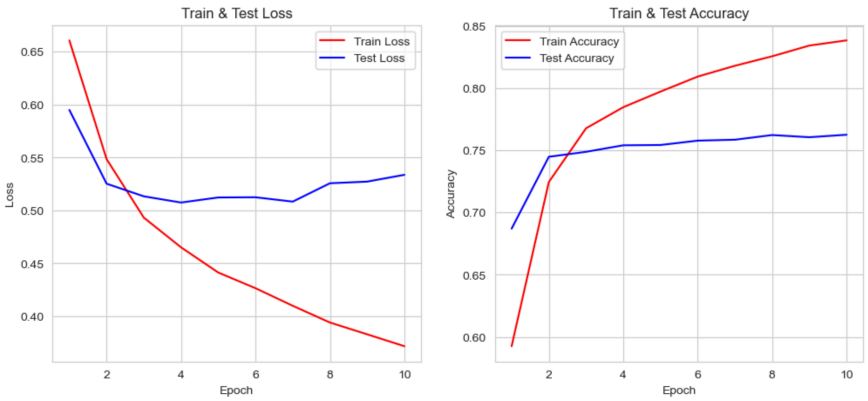


Figure 15. Learning curve of the final model

5. Conclusion and Future Work

Conclusion

In this project, we successfully designed and implemented a Recurrent Neural Network (RNN) model to perform sentiment analysis on tweets. By leveraging an LSTM-based architecture, we addressed the challenges of sequential data processing and captured long-term dependencies in text. Through systematic experimentation, we optimized key hyperparameters, including the number of layers, units, activation functions, embedding dimensions, and regularization techniques.

Our final model achieved a training accuracy of 84% and a test accuracy of 75%, based on a training set of 16,000 samples and a test set of 4,000 samples, demonstrating its ability to classify tweet sentiments effectively. However, the gap between training and test performance highlighted the need for further improvements to enhance generalization.

Error Analysis

The error analysis provided valuable insights into the model's limitations. The confusion matrix revealed that out of 4,000 test samples, there were 1,534 true negatives (correctly predicted negative tweets), 506 false negatives (negative tweets misclassified as positive), 1,516 true positives (correctly predicted positive tweets), and 444 false positives (positive tweets misclassified as negative). This indicates that the model struggled slightly more with false negatives, where negative tweets were misclassified as positive.

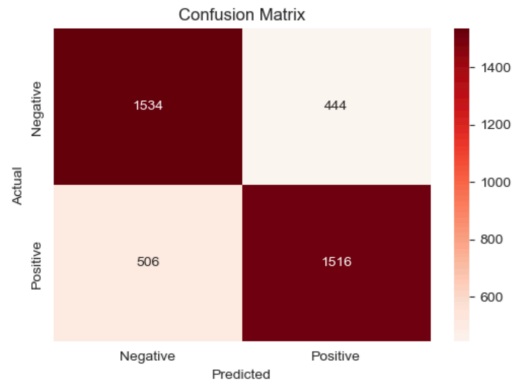


Figure 16. Confusion matrix

The misclassified samples' word clouds further illustrated this issue. Frequent words in false negative tweets included "go," "get," "work," "day," "school," and "home," suggesting that the

model may have difficulty interpreting neutral or context-dependent terms. Similarly, frequent words in false positive tweets, such as "love," "thank," "like," and "really," indicate that the model may overemphasize positive-sounding words that may be in sarcastic contexts and that the model does not consider the overall context.

Figure 17. Misclassified Samples Word Cloud in False Negative Tweets and False Positive Tweets

Future Work

To address these challenges, we propose several future improvements. First, incorporating dropout into the hidden layers, in addition to the LSTM layer, could help mitigate overfitting. In this experiment, only dropout in the LSTM layer was tested and evaluated. For future work, dropout could be added to the hidden layer too. Since different parts of the network could benefit from different dropout strengths, experimenting with varying rates per layer could help us improve generalization further and find a balance between underfitting and overfitting.

Another area for improvement lies in preprocessing and data augmentation. The error analysis suggests that the model struggles with context-dependent words and phrases. Enhancing the preprocessing pipeline to include sentiment-specific features, such as negation handling or phrase-level sentiment scoring[13], could improve the model's understanding of nuanced language. Data augmentation techniques, such as back-translation[14], could also increase the diversity of the training set, helping the model generalize better to unseen data.

By refining regularization techniques, and enhancing preprocessing, we believe we can further optimize the model and achieve better sentiment classification results. This project has provided a solid foundation for future work, and we are excited to continue building on these findings.

References

[1] J. Kaplan et al., "Scaling Laws for Neural Language Models," arXiv preprint arXiv:2001.08361, 2020. [Online]. Available: https://arxiv.org/pdf/2001.08361. [Accessed: Apr. 4, 2025].

[2] A. O. Adebiyi et al., "Utilizing an Attention-Based LSTM Model for Detecting Sarcasm and Irony in Social Media," Computers, vol. 12, no. 11, 2023. [Online]. Available: https://doi.org/10.3390/computers12110231. [Accessed: Apr. 4, 2025].

[3] O. Levy and Y. Goldberg, "Linguistic Regularities in Sparse and Explicit Word Representations," Proceedings of the Eighteenth Conference on Computational Natural Language Learning (CoNLL), pp. 171–180, 2014. [Online]. Available: https://aclanthology.org/W14-1618. [Accessed: Apr. 4, 2025].

[4] S. Semeniuta, A. Severyn, and E. Barth, "Recurrent Dropout without Memory Loss," arXiv preprint arXiv:1603.05118, 2016. [Online]. Available: https://arxiv.org/abs/1603.05118. [Accessed: Apr. 4, 2025].

[5] J. Dodge et al., "Documenting Model Biases via Confusion Matrix Decomposition," Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency (FAccT), pp. 457–468, 2021. [Online]. Available: https://doi.org/10.1145/3442188.3445910. [Accessed: Apr. 4, 2025].

[6] A. C. Müller and S. Guido, Introduction to Machine Learning with Python: A Guide for Data Scientists, 1st ed., O'Reilly Media, 2016. [Online]. Available: https://www.oreilly.com/library/view/introduction-to-machine/9781449369880/ch07.html. [Accessed: Apr. 4, 2025].

[7] Kazanova, "Sentiment140," Kaggle, [Online]. Available: https://www.kaggle.com/datasets/kazanova/sentiment140. [Accessed: Apr. 4, 2025].

[8] M. McLaughlin, "What is lemmatization?," *TechTarget*, [Online]. Available: https://www.techtarget.com/searchenterpriseai/definition/lemmatization. [Accessed: Apr. 4, 2025].

[9] Amazon Web Services, "What is a Recurrent Neural Network (RNN)?," *AWS*, [Online]. Available: https://aws.amazon.com/what-is/recurrent-neural-network/. [Accessed: Apr. 4, 2025].

[10] Keras, "Tokenizer," Keras Hub API, [Online]. Available: https://keras.io/keras_hub/api/tokenizers/tokenizer/. [Accessed: Apr. 4, 2025].

[11] TensorFlow, "tf.keras.utils.pad_sequences," *TensorFlow API Documentation*, [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/utils/pad_sequences. [Accessed: Apr. 4, 2025].

[12] G. Muller and L. Guido, *Machine Learning for Dummies*, 2nd ed., O'Reilly Media, 2018. [Online]. Available: https://www.oreilly.com/library/view/machine-learning-for/9781789136364/ch04s15.html. [Accessed: Apr. 4, 2025].

[13] J. Wei and K. Zou, "EDA: Easy Data Augmentation Techniques for Boosting Performance on Text Classification Tasks," arXiv preprint arXiv:1901.11196, 2019. [Online]. Available: https://arxiv.org/abs/1901.11196. [Accessed: Apr. 4, 2025].

[14] R. Sennrich et al., "Improving Neural Machine Translation Models with Monolingual Data," Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL), pp. 86–96, 2016. [Online]. Available: https://aclanthology.org/P16-1009. [Accessed: Apr. 4, 2025].