Sentiment Analysis on the Sentiment140 Dataset Using BiLSTM with and without Pretrained Word Embeddings

Mohammad Mohsen Hajari Taheri
PSU ID: 902241477

1 Problem Definition and Dataset Curation

1.1 Problem Definition

In this project, we explore a binary sentiment classification task using the Sentiment140 dataset. The goal is to predict whether a given tweet expresses a positive or negative sentiment. Specifically:

- Input: A tweet (text).
- Output: A binary label indicating sentiment (0 for negative, 1 for positive).

1.2 Dataset Curation

The Sentiment140 dataset, containing 1.6 million tweets automatically labeled using emoticons, provides sentiment labels of 0 for negative and 4 for positive (which we map to 1). The dataset is balanced, consisting of 800,000 positive tweets and 800,000 negative tweets. For this project, we focus exclusively on distinguishing positive and negative sentiments, disregarding other labels.

1.2.1 Data Preprocessing

We read the data, mapped label 4 to 1 for binary classification (0 = negative, 1 = positive), and used an 80-20 split to ensure both a sufficiently large training set and a representative test set. Although inherently large and noisy due to its social media origins, the dataset remains a widely used resource that provides a strong foundation for demonstrating the effectiveness of deep learning architectures.

2 Word Embeddings, Algorithm, and Training Details

2.1 Word Embeddings

In one approach, we train embeddings from scratch so they learn directly from our dataset. In the other, we use pretrained GloVe embeddings, which already contain broader language knowledge and stay fixed during training.

2.1.1 Model with Self-Trained Embeddings

We use a TensorFlow/Keras Embedding layer that is randomly initialized and trained from scratch on our dataset. Here are configurations for this self-trained embeddings:

- Vocabulary Size: 1,000 words.
- Embedding Dimension: 16.
- Trainable: True.
- Maximum length: 280 tokens.

2.1.2 Model with Pretrained GloVe (100d) Embeddings

We load the GloVe 100-dimensional embeddings trained on a corpus of 6B tokens. We create an embedding matrix that maps each word in our tokenizer's vocabulary to its corresponding GloVe vector. If the word does not exist in GloVe, its embedding remains zero-initialized. We set trainable to False to use the GloVe embeddings as-is and avoid altering them during training. Here are configurations for this pretrained GloVe embeddings:

- Vocabulary Size: 10,000 words.
- Embedding Dimension: 100.
- Trainable: False.
- Maximum length: 280 tokens.

We set the maximum sequence length to 280 tokens, using the typical 280-character tweet limit as an upper bound even though tokens do not map exactly one-to-one with characters.

2.2 Neural Network Architecture

We adopt a Bidirectional LSTM architecture for both the self-trained and pretrained approaches. First, an embedding layer either initializes word vectors randomly (self-trained) or loads them from GloVe (pretrained). Next, a 64-unit Bidirectional LSTM processes the embedded sequences from both forward and backward directions, capturing broader contextual dependencies within the text. A Dense layer with 16 units and a tanh activation then refines the extracted features, and finally, a single-unit output layer with a sigmoid activation maps these features to a binary classification for sentiment.

A 64-unit Bidirectional LSTM combines two LSTM layers that read the text in opposite directions. One layer reads from left to right, while the other reads from right to left. Each layer has 64 hidden units, capturing patterns and context across the entire tweet. By merging their outputs, the model gains a richer understanding of the text for more accurate sentiment classification.

2.3 Optimization Details

2.3.1 Optimizer

Here, we used Adam optimizer and set the learning rate to 0.001, a commonly used value that balances learning speed and stability. Additionally, we use clipnorm=1.0, which prevents the gradients from growing too large (exploding gradients), ensuring more stable updates and helping the model converge smoothly.

2.3.2 Loss Function

For this binary sentiment classification task, we use binary crossentropy as the loss function, which is well-suited for problems with two possible output classes (positive or negative sentiment). Binary crossentropy measures the difference between the predicted probability (output of the sigmoid activation) and the actual label (0 or 1). It is defined as:

$$L = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right]$$

By minimizing binary crossentropy, the model improves its ability to assign accurate sentiment probabilities to tweets.

2.3.3 Metric

For this sentiment classification task, we use accuracy as the primary evaluation metric. Accuracy measures the proportion of correctly classified tweets out of the total number of tweets in the dataset. It is defined as:

$$Accuracy = \frac{Number\ of\ Correct\ Predictions}{Total\ Number\ of\ Predictions}$$

2.3.4 Early Stopping

We use EarlyStopping as a callback during training to prevent overfitting and improve generalization. This technique monitors the validation loss at the end of each epoch and stops training if the loss does not improve for three consecutive epochs (patience=3). Once training is halted, the model automatically restores the best weights, ensuring that the final model is the one with the lowest validation loss, rather than one that may have overfit in later epochs. This helps avoid unnecessary computations, reduces training time, and ensures better performance on unseen data.

2.3.5 Epoch

We set the number of training epochs to 10, meaning the model goes through the entire dataset ten times during training.

2.4 Training Procedure

2.4.1 Text Tokenization

We use the Keras Tokenizer to convert text into sequences of integer indices, where each integer represents a specific word, and an out-of-vocabulary token (<OOV>) to handle unknown words.

2.4.2 Padding/Truncation

We ensure all sequences have a fixed length of 280 tokens, using it as the maximum length based on the typical character limit of a tweet and the original dataset's setup. To maintain consistency, we apply a post padding/truncation strategy, meaning shorter sequences are padded at the end, and longer sequences are truncated from the end.

2.4.3 Training

We split the data into training and testing sets with an 80/20 ratio, ensuring a sufficient amount of data for both model training and evaluation. The model is then trained on the training set and validated on the testing set to assess its generalization performance. Finally, we evaluate the accuracy and loss curves to analyze the model's learning progress and detect any signs of overfitting or underfitting.

3 Results

3.1 Self-Trained Embeddings

3.1.1 Accuracy

Figure 1 shows that in the self-trained embedding approach, validation accuracy stabilizes after epoch 4, indicating that the model has learned most of the useful patterns from the data. However, training accuracy continues to improve, suggesting that the model is overfitting the training data between epochs 4 and 10. This overfitting occurs because the model continues to memorize patterns in the training set while failing to generalize better to unseen data. The gap between training and validation accuracy suggests that additional regularization techniques, such as dropout or early stopping with a stricter patience setting, could help prevent overfitting.

3.1.2 Loss

Figure 2 shows the training and validation loss for the self-trained embedding approach over 10 epochs. Initially, both losses decrease, indicating that the model is learning. However, after epoch 3, validation loss stabilizes and stops improving significantly, while training loss continues to decrease. This aligns with what we observed in Figure 1 (Accuracy vs. Epochs), where validation accuracy remained steady after epoch 4 while training accuracy continued to improve. Similarly, in Figure 2, the validation loss stops decreasing after epoch 3, while training loss keeps declining, indicating that the model is no longer improving on unseen data and is instead memorizing patterns from the training set.

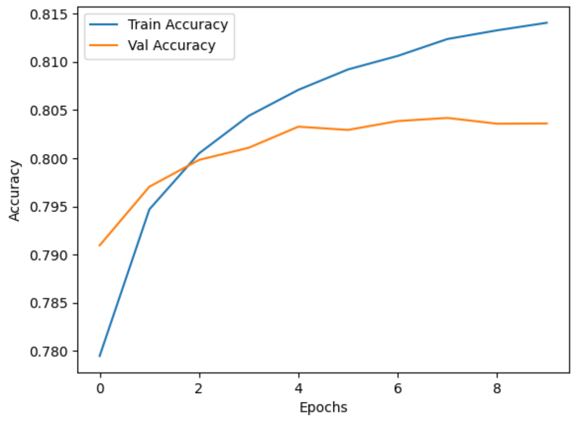

Figure 1: Accuracy vs. epochs for the model with self-trained embeddings.

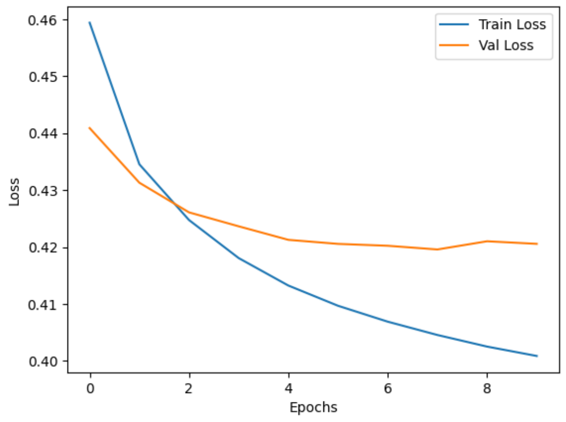

Figure 2: Validation loss vs. epochs for the model with self-trained embeddings.

3.2 Pretrained GloVe

3.2.1 Accuracy

In Figure 3, we used pretrained GloVe embeddings without training them further during model training. We observe that the model converges faster, as validation accuracy stabilizes within the first 3 epochs. This is expected since GloVe embeddings already capture meaningful word relationships, allowing the model to start with a strong representation of language rather than learning embeddings from scratch.

3.2.2 Loss

Figure 4 shows the training and validation loss for the model using pretrained GloVe embeddings. We can see that the validation loss decreases rapidly in the first few epochs, confirming that the pretrained embeddings help the model converge faster. However, after around epoch 3, validation loss stabilizes and fluctuates slightly, while training loss continues to decrease. This behavior suggests that the model is starting to overfit, as it keeps improving on the training data but no longer generalizes better to the validation set.

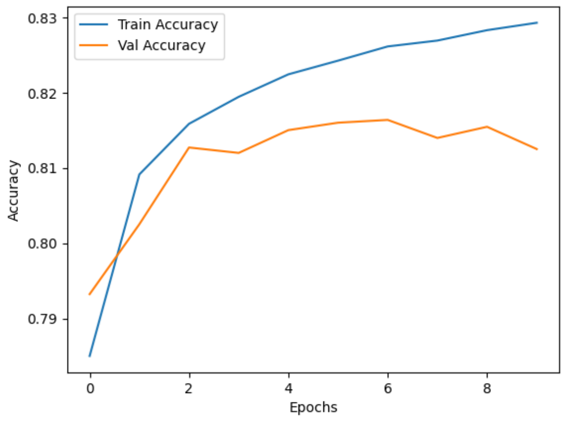

Figure 3: Accuracy vs. epochs for the model with GloVe embeddings.

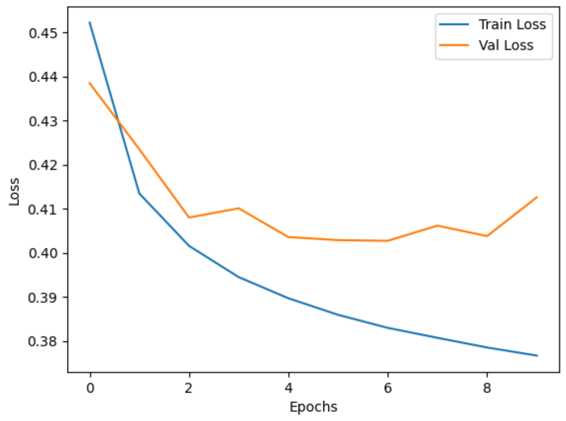

Figure 4: Validation loss vs. epochs for the model with GloVe embeddings.

4 In-Depth Analyses and Experiments

4.1 Hyperparameter Tuning

We experimented with different embedding dimensions (16 vs. 100). Larger pretrained embeddings capture richer semantic information but may require a larger vocabulary size and longer training time to fully leverage their potential. We also fine-tuned optimization parameters, particularly the learning rate and gradient clipping, to ensure stable training.

Initially, we used the Adam optimizer with a learning rate of 0.01, but this resulted in exploding gradients, where the model's weight updates became excessively large, causing instability. To address this, we reduced the learning rate to 0.001 and applied clipnorm=1.0, which restricts the maximum gradient norm. This adjustment successfully stabilized training, prevented extreme weight updates, and led to more reliable convergence across epochs.

4.2 Bidirectional LSTM vs. Unidirectional LSTM

The bidirectional variant provides context from both past and future tokens, which is beneficial for capturing nuances in short texts like tweets.

4.3 Vocabulary Size Trade-offs

A smaller vocabulary (1,000 words) can speed up training but may miss many nuances of language, leading to a less expressive model. A larger vocabulary (10,000 words) captures more words and richer context but increases model size and training time.

5 Lessons Experience Learned

5.1 Expectation vs. Reality: The Impact of Pretrained vs. Self-Trained Embeddings

Before obtaining the results, I expected that using pretrained embeddings like GloVe would significantly improve accuracy, rather than just a 1-3% increase. While pretrained embeddings provide more stable and general semantic representations, self-trained embeddings can better adapt to the specific language and style of tweets, capturing domain-specific nuances that pretrained embeddings may not fully represent. Fine-tuning the pretrained embeddings or using domain-specific (Tweets) pretrained embeddings could potentially improve performance further.

5.2 Recognizing and Addressing Overfitting in Model Training

One key lesson learned is that the slight upward trend in validation loss and the slight downward trend in accuracy toward the last two epochs in both the pretrained and self-trained models indicate that the models are beginning to overfit, and the model start memorizing training patterns rather than generalizing effectively. To mitigate overfitting, future work could explore applying regularization techniques such as dropout, L2 regularization, or early stopping with a lower patience value to enhance generalization and maintain performance on unseen data.

5.3 Model Complexity vs. Data Size

With 1.6 million tweets, a more complex model (like a deeper BiLSTM or a CNN+LSTM hybrid) might yield better performance, but also demands more computational resources.