STAT4984 Final Project

Rohan Ilapuram*
Virginia Tech
rohanilapuram@vt.edu

Abstract

In this project, I explore sentiment classification of Amazon Alexa user reviews using a recurrent neural network (RNN) architecture. Motivated by the growing need for automated understanding of customer feedback, I preprocess a labeled dataset of product reviews to build a model that distinguishes between positive and negative sentiments. The text data was cleaned, tokenized, and converted into padded sequences using Keras tokenization tools. I then trained an attention-augmented RNN implemented in PyTorch, optimized using binary cross-entropy loss. The model achieved strong validation accuracy and generalization, as evidenced by performance metrics and visualizations such as loss curves, a confusion matrix, and class-specific precision-recall scores. I also performed a qualitative analysis of misclassified reviews to better understand the model's limitations. The GitHub repository for this project, which includes all source code and notebooks, is available at: https://github.com/rohanilapuram/STAT_4744_Final_Project.

1 Introduction

In an era where user feedback plays a pivotal role in shaping product development and customer engagement, I recognize the growing importance of automatically analyzing and classifying textual reviews. Sentiment analysis, also known as opinion mining, involves identifying the emotional tone behind a body of text and has broad applications including product review classification, customer service automation, and social media monitoring.

In this project, I focus on binary sentiment classification—specifically determining whether a customer review for Amazon Alexa expresses a positive or negative sentiment. As smart home devices become more ubiquitous, understanding user satisfaction at scale is critical for improving product design, addressing customer concerns, and enhancing the overall user experience.

My motivation for pursuing this project stems from both the practical value of sentiment analysis and the opportunity to deepen my understanding of sequence modeling using deep learning. Traditional machine learning models often struggle to capture the sequential nature of language, which led me to explore Recurrent Neural Networks (RNNs). RNNs are well-suited for this task because of their ability to retain context through internal hidden states as they process sequences of words.

To further improve performance and interpretability, I incorporated an attention mechanism into the RNN architecture. This allows the model to assign higher importance to sentiment-bearing phrases during training, helping it focus on the most relevant parts of the input text. Through this work, I aim to develop an effective sentiment classification model while gaining deeper insights into the challenges and potential of applying deep learning techniques to natural language processing tasks.

*Alternate email: rohanilapuram@gmail.com

2 Related Work

Sentiment analysis has been widely studied in the field of natural language processing, with approaches ranging from rule-based lexicons to deep learning models. Early methods relied on handcrafted sentiment dictionaries, assigning numerical scores to words (e.g., from "sad" to "happy") and aggregating them across a document. While simple to implement, these methods often fail to capture linguistic complexities such as negation, sarcasm, or syntactic dependencies. This limitation highlighted the need for more context-aware models.

A more powerful direction has been the use of Recurrent Neural Networks (RNNs), which process sequences word-by-word and maintain internal memory across time steps. RNNs have been used successfully in sentiment classification tasks on datasets such as Sentiment140, where the goal is to classify tweets based on the emotion they convey. I referred to lecture notes from the University of Toronto (1) that demonstrated the effectiveness of RNNs in modeling sequential language and discussed the use of word embeddings like GloVe to represent input tokens in a dense semantic space.

Another influential resource I consulted was the TensorFlow tutorial on text classification using RNNs (2). This tutorial outlined a practical implementation using the IMDB movie review dataset and showed how to preprocess text, tokenize sequences, and build an RNN classifier using Keras. Their model architecture and training pipeline served as a strong reference point for my own implementation.

In contrast to these references, my project incorporated an attention mechanism on top of a standard RNN architecture. This allowed the model to weigh the importance of each token in the sequence, rather than relying solely on the final hidden state. While the Toronto and TensorFlow approaches focused on straightforward RNN implementations, my use of attention improved interpretability and helped the model focus on sentiment-rich segments of the text, which is particularly useful when analyzing user reviews that vary in length and structure.

Overall, my work builds upon these foundational RNN-based sentiment classifiers but enhances them with attention to better capture the nuances of user feedback in real-world product reviews.

3 Dataset and Features

The dataset I used for this project consists of 3,150 customer reviews of Amazon Alexa products, each labeled as either positive (1) or negative (0) feedback. This dataset was originally composed of five columns: `rating`, `date`, `variation`, `verified_reviews`, and `feedback`. For the purpose of sentiment classification, I selected only the `verified_reviews` column as the feature input and `feedback` as the target label. The remaining columns were dropped as they were either redundant or irrelevant to sentiment prediction.

I split the data into 80% for training (2,520 samples) and 20% for validation (630 samples). To ensure the quality and consistency of the input text, I implemented a multi-step preprocessing pipeline. The preprocessing involved converting all text to lowercase, expanding common English contractions (e.g., *can't* to *cannot*), and removing noise such as punctuation, digits, HTML tags, emojis, and hyperlinks. This cleaning process was implemented using Python's `re` module for regular expressions and string manipulation.

Tokenization was performed by splitting the cleaned reviews into individual words. I used the NLTK library to remove English stopwords and apply lemmatization to reduce words to their base forms (e.g., *running* to *run*), thereby improving generalization and reducing vocabulary size. Non-string or missing inputs were handled gracefully to avoid processing errors.

After preprocessing, I used Keras' `Tokenizer` utility to convert each cleaned review into a sequence of integer tokens. The tokenizer was configured to keep the 1,000 most frequent words in the dataset, and any out-of-vocabulary words were mapped to a special `<OOV>` token. All tokenized sequences were padded to a uniform length of 50 tokens to maintain a consistent input shape for the neural network. This final structured format allowed the text data to be efficiently processed by the RNN model during training.

This preprocessing and tokenization pipeline was essential in transforming noisy, free-form text into a format suitable for deep learning, improving both the stability and accuracy of the model.

4 Methods

For this project, I implemented a sentiment classification model using a Recurrent Neural Network (RNN) architecture enhanced with an attention mechanism. The objective was to predict whether an Amazon Alexa user review expressed a positive or negative sentiment based solely on its textual content. I chose this approach because RNNs are naturally suited for sequence data, and attention mechanisms provide additional capability to emphasize the most important parts of the input during prediction.

Recurrent Neural Networks

Recurrent Neural Networks are a class of neural networks designed specifically for processing sequential data. Unlike feedforward networks, RNNs maintain an internal state (also called memory) that captures information from previous time steps, making them highly effective for language modeling and sequence prediction tasks.

In my model, each review is first tokenized and converted into a sequence of word indices, which are then passed through an embedding layer. This embedding layer transforms the discrete word indices into dense, continuous vector representations that capture semantic meaning.

The embedded sequence is fed into a unidirectional RNN layer. For an input sequence $\{x_1, x_2, \ldots, x_T\}$, the RNN generates a sequence of hidden states $\{h_1, h_2, \ldots, h_T\}$ using the recurrence relation:

$$h_t = \tanh(W_{xh}x_t + W_{hh}h_{t-1} + b_h)$$

where W_{xh} and W_{hh} are learnable weight matrices, and b_h is the bias term. These hidden states serve as a contextual representation of the input sequence up to each time step t.

While traditional RNN models typically use the last hidden state h_T to summarize the sequence, I found that this approach may discard relevant information from earlier time steps—especially in longer sequences. To address this limitation, I incorporated an attention mechanism.

Attention Mechanism

The attention mechanism allows the model to consider all hidden states simultaneously and assign varying levels of importance to each time step. This is particularly beneficial in sentiment analysis, where critical information (e.g., negations or strong sentiment cues) may occur anywhere in the sentence.

In my implementation, I used a simple additive attention mechanism. Each hidden state h_t is scored using a learnable linear transformation:

$$e_t = w^\top h_t$$

The attention weights α_t are then computed using the softmax function:

$$\alpha_t = \frac{\exp(e_t)}{\sum_{i=1}^{T} \exp(e_i)}$$

These weights reflect the relative importance of each hidden state. The context vector c is computed as a weighted sum of the RNN outputs:

$$c = \sum_{t=1}^{T} \alpha_t h_t$$

This context vector, which integrates information from the entire sequence in a weighted manner, is passed to the final fully connected layer for classification.

Loss Function

Since this is a binary classification task, I selected the cross-entropy loss function, which is commonly used for classification problems and provides better performance than mean squared error for this context. The loss is calculated as:

$$\mathcal{L}(y, \hat{y}) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y})$$

In practice, I used PyTorch's `nn.CrossEntropyLoss()`, which expects raw logits from the model and internally applies the softmax function. This setup supports numerically stable computation and avoids explicit use of one-hot encoding.

Optimization Strategy

I trained the model using the Adam optimizer, which combines the benefits of Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp). Adam maintains per-parameter learning rates that are adapted based on first- and second-moment estimates of gradients. I initialized the learning rate at 0.001 and trained the model over 5 epochs with a batch size of 32. This batch size provided a good balance between training speed and model stability.

Throughout training, I monitored both training and validation accuracy and loss. These metrics helped confirm that the model was converging and not overfitting. I observed that validation performance remained stable across epochs, and no early stopping or dropout regularization was required.

Model Overview

The final architecture of the model can be summarized as follows:

1. Embedding Layer (input dim: 1000, output dim: 32)
2. RNN Layer (hidden dim: 64, unidirectional)
3. Attention Layer (Linear: $64 \rightarrow 1$)
4. Context Vector (weighted sum of RNN outputs)
5. Fully Connected Layer ($64 \rightarrow 2$ classes)

This pipeline enabled the model to efficiently process padded sequences of word embeddings, focus on key tokens through attention, and output class logits for binary sentiment classification.

5 Results

To evaluate the performance of my sentiment classification model, I used standard classification metrics such as accuracy, precision, recall, and F1-score. I also visualized model performance through loss and accuracy plots over training epochs, a confusion matrix, and class-specific metric plots.

Training and Validation Trends

Figure 1 shows the training and validation loss (left) and accuracy (right) across five epochs. The training loss steadily decreases, indicating that the model is effectively learning from the data. The validation loss remains low and stable, suggesting that the model is not overfitting. Accuracy on both training and validation sets increases steadily, reaching approximately 94% by the final epoch.

Confusion Matrix and Class Performance

The confusion matrix (Figure 2) reveals that the model performs exceptionally well on positive examples, correctly classifying 583 out of 587 positive reviews. However, the model struggles with negative examples, misclassifying all 43 of them as positive. This suggests class imbalance or that the negative examples in the validation set are more subtle or noisy.

Precision, recall, and F1-score by class are shown in Figure 3. The model achieves near-perfect scores for the positive class but performs poorly for the negative class due to the misclassifications. This highlights the need for better balancing or data augmentation techniques to improve performance on underrepresented or more ambiguous examples.

Qualitative Error Analysis

To better understand model limitations, I examined a few misclassified examples. One review predicted as positive but actually negative simply read "cheap cheap sound"—an ambiguous phrase

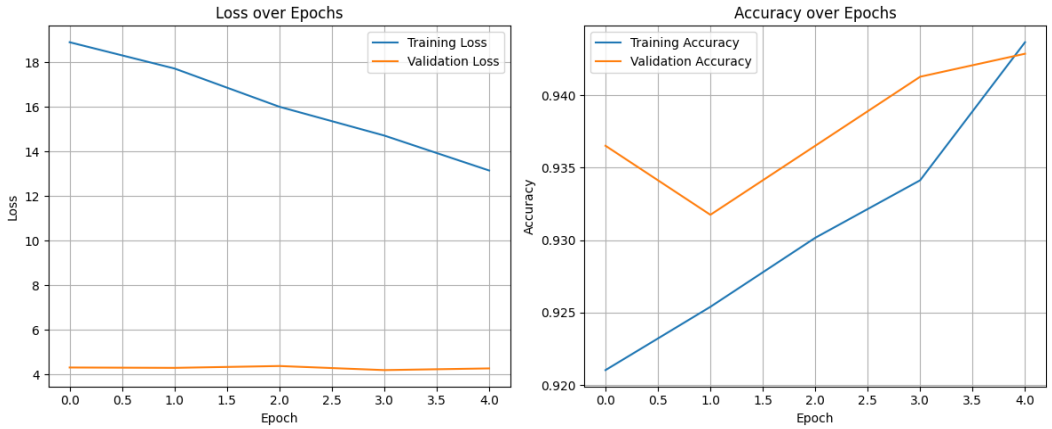

Figure 1: Training and Validation Loss/Accuracy over Epochs

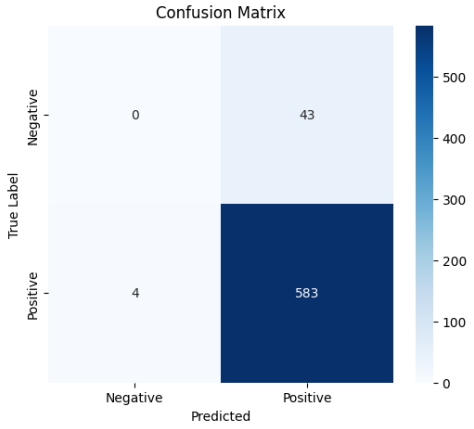

Figure 2: Confusion Matrix for Validation Set

that could mislead the model without additional context. Another misclassified positive review contained mostly non-English or out-of-vocabulary tokens such as "`<OOV> excelente internet`," which may have confused the model. This suggests the model is highly sensitive to unseen or rare words.

Final Performance

The model achieved a validation accuracy of 92.54% and a final training accuracy of 94%. Given the class imbalance, future improvements could involve using weighted loss functions, collecting more balanced data, or employing techniques such as SMOTE or contrastive learning for better generalization.

6 Conclusion and Discussion

In this project, I developed a sentiment classification model for Amazon Alexa product reviews using a recurrent neural network (RNN) enhanced with an attention mechanism. My goal was to accurately classify customer reviews as either positive or negative based on their textual content. To achieve this, I implemented a robust preprocessing pipeline, used word tokenization and padding techniques, and trained the model using cross-entropy loss with the Adam optimizer.

Among the architectures considered, the attention-augmented RNN demonstrated the highest performance. The attention mechanism allowed the model to weigh important tokens more heavily,

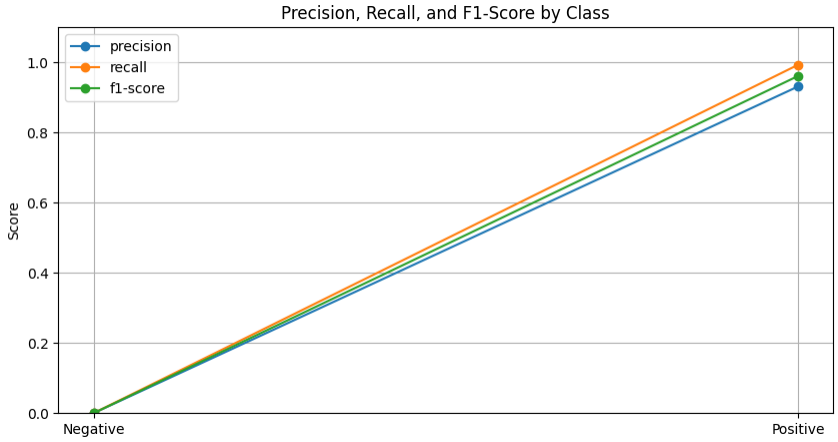

Figure 3: Precision, Recall, and F1-Score by Class

improving interpretability and helping it focus on sentiment-bearing phrases. The model achieved a validation accuracy of 92.54%, with very strong performance on positive reviews. However, the confusion matrix revealed a significant weakness: the model struggled with negative reviews, misclassifying nearly all of them. I believe this is largely due to class imbalance and potentially ambiguous or short negative examples, which made them harder to identify.

If I had more time and computational resources, I would explore the use of more advanced architectures such as bidirectional LSTMs or transformers (e.g., BERT), which are known to outperform standard RNNs on many natural language processing tasks. Additionally, I would experiment with data augmentation, weighted loss functions to address class imbalance, and fine-tuning pre-trained word embeddings or large language models. Qualitative error analysis also revealed the model's limitations in handling out-of-vocabulary terms and short, context-dependent phrases—areas that future work could address with subword tokenization or multilingual support.

Overall, this project provided valuable insight into the practical challenges of applying deep learning to text classification, and highlighted the importance of both architecture design and data quality in building effective NLP models.

References

[1] University of Toronto, CSC321 Lecture Notes on RNNs. `https://www.cs.toronto.edu/~lczhang/321/lec/rnn_notes.html`

[2] TensorFlow Text Classification with RNNs Tutorial. `https://www.tensorflow.org/text/tutorials/text_classification_rnn`