Comparative Analysis of RNN Architectures for Sentiment Classification

1. Introduction

Sentiment Classification is a core Natural Language Processing (NLP) task that involves categorizing the emotional tone of a piece of text, such as a movie review, tweet, or product comment into classes like positive or negative.

This project implements and evaluates multiple Recurrent Neural Network (RNN) architectures for this task, treating it as a sequence classification problem. We systematically compare the performance of simple **RNNs**, **LSTMs**, and **Bidirectional LSTMs** on the IMDb movie review dataset. The goal is to identify how architecture, activation functions (Tanh, ReLU, Sigmoid), optimizers (Adam, SGD, RMSProp), sequence length, and the use of gradient clipping affect model accuracy, F1-score, and training efficiency on a CPU-constrained system.

2. Dataset and Preprocessing

Dataset

The dataset used for this project is the **IMDb Movie Review Dataset**, which contains 50,000 highly polarized movie reviews. The dataset is balanced, with 25,000 reviews designated for training and 25,000 for testing.

Text Preprocessing

As per the project guidelines, the following preprocessing pipeline was applied to the raw text data:

1. **Normalization:** All text was converted to lowercase.
2. **Cleaning:** Punctuation and special characters were removed.
3. **Tokenization:** Sentences were tokenized into individual words.
4. **Vocabulary Creation:** A vocabulary was built based on the training data, limited to the **top 10,000 most frequent words**.
5. **Integer Sequencing:** Each review was converted into a sequence of corresponding token IDs from the vocabulary.
6. **Padding & Truncation:** To ensure uniform input length for the models, sequences were either padded with zeros or truncated to three fixed lengths: **25**, **50**, and **100** words. These three lengths were tested as separate experimental variables.

3. Experimental Setup

Hardware

All 52 experiments were conducted on a **CPU-only** system with **8 cores**. This constraint places a heavy emphasis on model training time and computational efficiency.

Model Architecture

All models shared a similar base structure:

1. An **Embedding Layer** to convert the integer-encoded vocabulary (10,000-dimensional) into dense vector representations.
2. A **Recurrent Layer** which was the main variable:
    - Simple RNN
    - LSTM (Long Short-Term Memory)
    - Bidirectional LSTM
3. A final **Dense Layer** with a Sigmoid activation function to output a single probability score for binary (positive/negative) classification.

Experimental Variations

A total of **52 unique configurations** were trained and evaluated by varying the following parameters:

- **Architecture:** RNN, LSTM, Bidirectional LSTM
- **Activation Function:** Tanh, ReLU, Sigmoid
- **Optimizer:** Adam, SGD, RMSProp
- **Sequence Length:** 25, 50, 100
- **Stability Strategy:** Gradient Clipping (max norm 1.0) vs. No Gradient Clipping

Each model was trained for 5 epochs, and its performance was evaluated on the 25,000-sample test set using Accuracy, F1-Score, and average epoch training time.

4. Results and Comparative Analysis

The 52 experiments yielded a wide range of results, highlighting significant differences in performance based on the chosen hyperparameters.

Best and Worst Performing Models

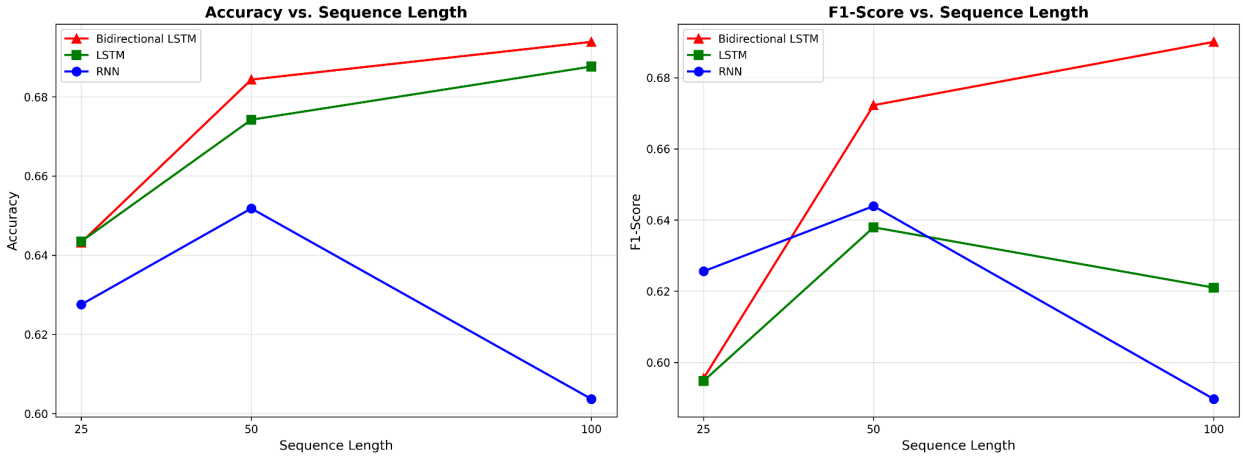
A summary of the best and worst-performing models identified from the experiments is presented below.

Metric	Best Performing Model	Worst Performing Model
Model	LSTM	LSTM
Activation	Tanh	ReLU
Optimizer	Adam	SGD
Seq Length	100	100
Grad Clipping	No	No
Accuracy	**82.44%**	**49.99%**
F1-Score	**82.42%**	**33.33%**
Avg Epoch Time	21.32 s	21.62 s

The worst-performing model's metrics (F1-score of 0.333, Accuracy of ~50%) indicate a complete failure to learn, essentially performing no better than a random guess.

Performance vs. Sequence Length

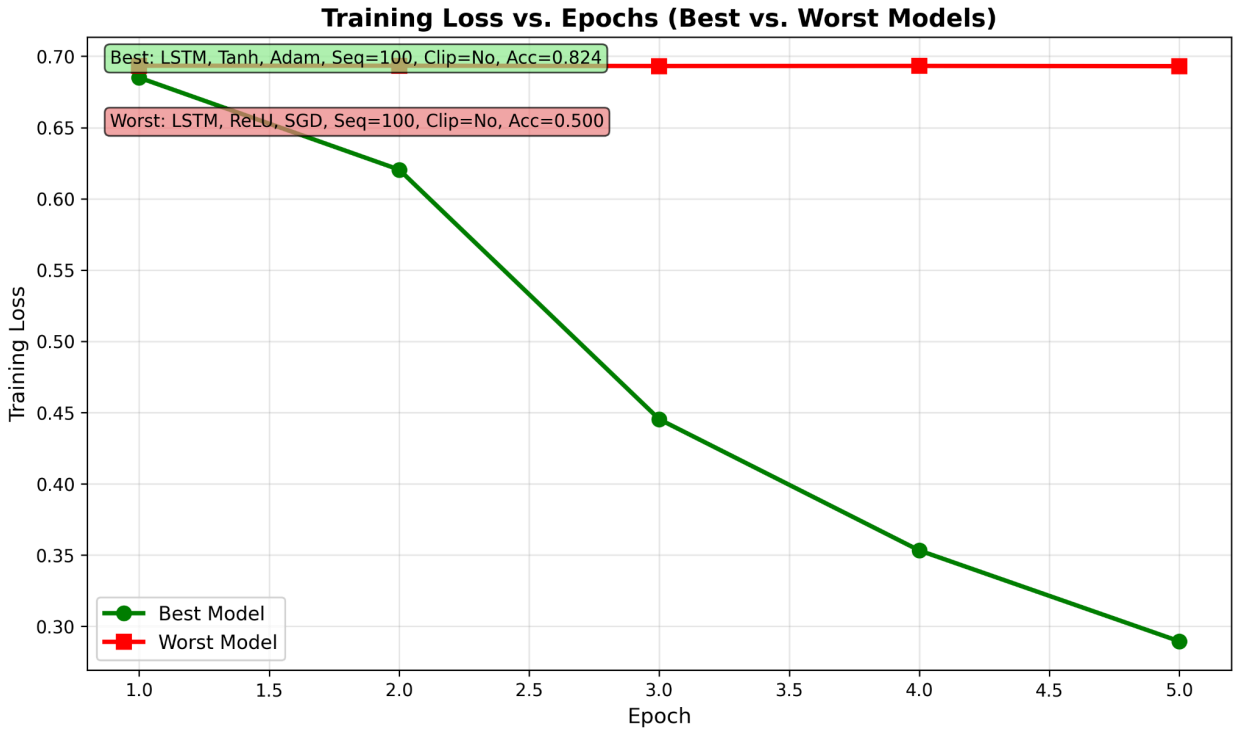
The chart below illustrates the impact of sequence length (25, 50, or 100) on model accuracy and F1-score, grouped by the optimizer used.



As shown in the plot, for effective optimizers like Adam and RMSProp, **increasing the sequence length consistently improved performance**. Models using a sequence length of 100 almost universally outperformed those using 50 or 25. Conversely, for the ineffective SGD optimizer, sequence length had no impact as the models failed to learn regardless.

Training Loss Analysis

The training loss curves for the best and worst-performing models show a stark contrast in convergence.



The best model (LSTM, Tanh, Adam, 100) shows a rapid and smooth decrease in training loss over the 5 epochs, indicating successful learning. The worst model (LSTM, ReLU, SGD, 100) shows a completely flat loss at ~0.693, which is characteristic of a model that is not learning and is stuck at the initial random-guess baseline.

Note: A potential stability issue was observed during the experiments. Specifically, when using the LSTM and Bidirectional LSTM models with the RMSProp optimizer, disabling gradient clipping led to model failure. This is likely due to gradient explosion, which can destabilize the training process even with an adaptive optimizer like RMSProp.

Full Experimental Results

The complete summary table for all 52 experiments is provided in the **metrics.csv** file and for more information check **detailed_results.json**.

5. Discussion

Based on the analysis of the results, we can answer the key questions posed by the assignment.

Which configuration performed best?

The configuration that achieved the highest accuracy and F1-score was the **LSTM** model (Row 29) with the following parameters:

- **Activation:** Tanh
- **Optimizer:** Adam
- **Sequence Length:** 100
- **Gradient Clipping:** No

This model achieved an **Accuracy of 82.44%** and an **F1-Score of 82.42%**, with a respectable average epoch time of 21.32 seconds on the CPU.

How did sequence length or optimizer affect performance?

Optimizer Effect:

The choice of optimizer was the single most critical factor in model performance.

- **SGD:** The Stochastic Gradient Descent optimizer, even with a standard learning rate of 0.01, **failed to train any model architecture** (RNN, LSTM, or BiLSTM). As seen in the results table, all 18 experiments using SGD resulted in accuracy scores of ~50-51% and F1-scores near or below 0.5, which is consistent with random guessing. The training loss logs confirm these models never converged.
- **Adam and RMSProp:** In stark contrast, both adaptive optimizers performed exceptionally well. They were able to successfully train all three architectures, leading to significant drops in training loss and test accuracies ranging from the high 60s to the low

80s. The best overall model used Adam, but RMSProp also produced highly competitive results, particularly with the simple RNN.

Sequence Length Effect:

As illustrated in the Performance vs. Sequence Length chart, for models that successfully trained (i.e., those using Adam or RMSProp), performance scaled directly with sequence length.

- Using a sequence length of **100** consistently yielded the best results for all three architectures.
- A length of **50** was a clear improvement over 25.
- A length of **25** was too short, likely truncating too much of the context needed to determine the sentiment of a review, leading to the lowest (but still non-random) scores.

How did gradient clipping impact stability?

The impact of gradient clipping was **not definitive** and appeared to be secondary to optimizer and sequence length choice.

- The **best-performing model (Row 29) had gradient clipping disabled**, and its training loss curve shows it was stable without it.
- However, in other direct comparisons, clipping provided a minor benefit. For example, the Bidirectional LSTM (ReLU, RMSProp, 50) achieved **77.34% accuracy *with* clipping** (Row 48) versus **76.34% *without* it** (Row 51).
- For the SGD models, clipping made no practical difference, as the models were not learning, and their gradients were likely already small (vanishing).

This suggests that for this dataset and model size, the adaptive optimizers (Adam, RMSProp) already provided sufficient training stability, making gradient clipping a non-essential, though sometimes slightly helpful, addition.

6. Conclusion

This project successfully demonstrated the significant impact of hyperparameter choices on the performance of recurrent models for sentiment analysis.

The **optimal configuration** for **maximum accuracy** was an **LSTM** model with **Tanh** activation, the **Adam** optimizer, and a sequence length of **100**, achieving **82.44% accuracy**.

However, when considering the **CPU constraint**, a strong trade-off must be considered.

- The best LSTM model took **21.32 seconds** per epoch.
- The best Bidirectional-LSTM model (Row 46) was slightly less accurate (81.76%) but took more than twice as long (**47.00 seconds** per epoch), making it an inefficient choice.
- Interestingly, a simple **RNN** (Row 14: ReLU, RMSProp, 100, Yes Clip) achieved a respectable **76.68% accuracy** in only **9.72 seconds** per epoch.

Final Conclusion:

The LSTM (Tanh, Adam, 100) model is the optimal choice, as it provides the highest performance without the extreme computational overhead of the Bidirectional models. While the simple RNN is faster, its ~6% drop in accuracy is a significant trade-off. The 21-second epoch time for the best LSTM model is a reasonable compromise for achieving over 82% accuracy on a CPU. The clear failure of SGD highlights that using an adaptive optimizer like Adam or RMSProp is non-negotiable for this task.

7. Reproducibility

The code for this project is submitted alongside this report. To reproduce these results, please refer to the README.md file for instructions on setting up the environment (e.g., from requirements.txt) and running the main training script.