Comparative Analysis of RNN Architectures for Sentiment Classification

Mounika Teppola
121012135
DATA641 - Natural Language Processing
Homework 3

Introduction

Sentiment classification is a fundamental task in Natural Language Processing (NLP) that focuses on identifying the emotional polarity of a given piece of text, such as a movie review, tweet or customer feedback. The goal is to automatically determine whether the expressed sentiment is positive, negative or neutral. This capability is crucial for applications like opinion mining, product recommendation systems and social media analytics, where understanding user attitudes can drive data-driven decision-making.

Traditional machine learning approaches for sentiment classification, such as Support Vector Machines (SVMs) or Naive Bayes classifiers, rely heavily on handcrafted features like term frequencies or n-grams. However, these methods often fail to capture the sequential and contextual dependencies inherent in natural language. To address this limitation, deep learning architectures particularly Recurrent Neural Networks (RNNs), have emerged as powerful tools for sequence modeling.

RNN-based models, including Long Short-Term Memory (LSTM) and Bidirectional LSTM (BiLSTM), are specifically designed to capture temporal dependencies and contextual relationships between words in a sequence. While simple RNNs are capable of learning short-term dependencies, they often suffer from issues such as vanishing or exploding gradients, making them less effective for long text sequences. LSTMs and BiLSTMs mitigate these problems through gating mechanisms and bidirectional processing, thereby achieving more robust performance on sequential data.

This project conducts a comprehensive comparative analysis of multiple RNN architectures—RNN, LSTM and BiLSTM for the task of sentiment classification using the IMDb Movie Review Dataset. The objective is to evaluate how different architectures, activation functions, optimizers and sequence lengths affect model performance and training efficiency. Through this analysis, the project aims to identify the optimal configuration that balances accuracy, F1-score and computational cost under CPU constraints.

Dataset Summary

The dataset used in this project is the **IMDb Movie Review dataset**, which contains **50,000 movie reviews** evenly split into **25,000 training** and **25,000 testing** samples. Each review is labeled as either **positive** or **negative**, making it suitable for binary sentiment classification.

Data Preprocessing

The preprocessing pipeline converts raw text into sequences suitable for RNN models. The steps are:

1. **Lowercasing and Cleaning:** All reviews were converted to lowercase, and punctuation and special characters were removed to standardize the text and reduce vocabulary size.

2. **Tokenization and Sequencing:** Using Keras' `TextVectorization`, reviews were converted into sequences of token IDs. Only the **top 10,000 most frequent words** were retained.

3. **Padding/Truncation:** Sequences were adjusted to fixed lengths of 25, 50, and 100 tokens. Shorter sequences were padded with zeros; longer sequences were truncated.

4. **Train-Test Split:** Each sequence length dataset was split 50/50, giving 25,000 training and 25,000 testing samples.

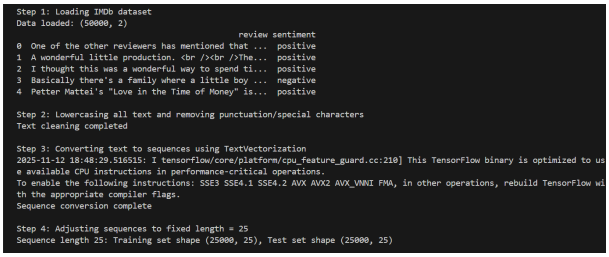

Figure 1: Preprocessing steps including loading, cleaning, tokenizing, and adjusting sequence lengths.

Figure 2: Saving the processed datasets for each sequence length.

Dataset Statistics

- **Number of reviews:** 50,000
- **Classes:** Positive / Negative
- **Training set:** 25,000 samples
- **Testing set:** 25,000 samples
- **Sequence lengths tested:** 25, 50, 100 tokens
- **Vocabulary size:** 10,000 most frequent words

Data Storage

The preprocessed datasets were saved in compressed `.npz` files for efficient reuse:

- `imdb_seq25.npz`
- `imdb_seq50.npz`
- `imdb_seq100.npz`

Each file contains `X_train`, `X_test`, `y_train`, and `y_test` arrays, corresponding to token sequences and labels.

Model Configuration

This project implements and compares three recurrent neural network architectures, **Simple RNN**, **LSTM** and **Bidirectional LSTM (BiLSTM)** for binary sentiment classification on the IMDb movie review dataset. All models were implemented using `PyTorch` and share a consistent baseline structure to ensure comparability.

Architecture Overview

Each model processes tokenized reviews as sequences of word embeddings and outputs a probability representing the likelihood of a positive sentiment. The shared configuration across models includes:

- **Embedding layer:** Converts tokens into 100-dimensional dense vectors.
- **Recurrent layers:** Two hidden layers with a hidden size of 64 units each.
- **Dropout:** 0.5 applied after recurrent layers to mitigate overfitting.
- **Output layer:** A fully connected neuron with sigmoid activation for binary classification.

All models were trained for 5 epochs with a batch size of 32 using binary cross-entropy loss. ReLU was used as the primary activation function unless stated otherwise.

Architectural Variants

Simple RNN: A baseline sequential model that learns temporal dependencies through recurrent connections.

```
============================================================
Model Architecture: SimpleRNNModel
------------------------------------------------------------
SimpleRNNModel(
  (embedding): Embedding(10000, 100)
  (rnn): RNN(100, 64, num_layers=2, batch_first=True, dropout=0.5)
  (dropout): Dropout(p=0.5, inplace=False)
  (fc): Linear(in_features=64, out_features=1, bias=True)
  (out_act): Sigmoid()
  (act): ReLU()
)
------------------------------------------------------------
Total Parameters     : 1,019,009
Trainable Parameters : 1,019,009
Embedding Dimension  : 100
Recurrent Type       : Simple RNN
Hidden Size          : 64
Number of Layers     : 2
Dropout Rate         : 0.5
Activation Function  : ReLU
Output Activation    : Sigmoid
============================================================
```

Figure 3: Model summary for Simple RNN architecture.

LSTM: Incorporates gating mechanisms to handle long-range dependencies and prevent gradient vanishing.

```
============================================================
Model Architecture: LSTMModel
------------------------------------------------------------
LSTMModel(
  (embedding): Embedding(10000, 100)
  (lstm): LSTM(100, 64, num_layers=2, batch_first=True, dropout=0.5)
  (dropout): Dropout(p=0.5, inplace=False)
  (fc): Linear(in_features=64, out_features=1, bias=True)
  (out_act): Sigmoid()
  (act): ReLU()
)
------------------------------------------------------------
Total Parameters     : 1,075,841
Trainable Parameters : 1,075,841
Embedding Dimension  : 100
Recurrent Type       : LSTM
Hidden Size          : 64
Number of Layers     : 2
Dropout Rate         : 0.5
Activation Function  : ReLU
Output Activation    : Sigmoid
============================================================
```

Figure 4: Model summary for LSTM architecture.

BiLSTM: Extends LSTM by processing sequences bidirectionally, enabling the network to leverage both preceding and succeeding context.

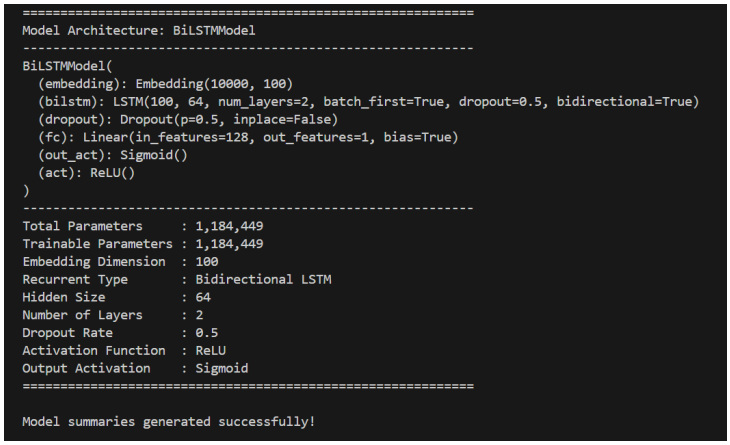

Figure 5: Model summary for BiLSTM architecture.

Hyperparameter Setup

A consistent experimental framework was followed while varying key parameters to study their impact on model performance:

- **Optimizers:** Adam, SGD, and RMSProp
- **Sequence lengths:** 25, 50, and 100 tokens
- **Gradient clipping:** Enabled and disabled configurations
- **Activation functions:** ReLU, Tanh, and Sigmoid

Fixed Parameters

Parameter	Value
Embedding dimension	100
Hidden size	64
Number of recurrent layers	2
Dropout rate	0.5
Batch size	32
Loss function	Binary Cross-Entropy
Output activation	Sigmoid
Epochs per run	5

Hardware Environment

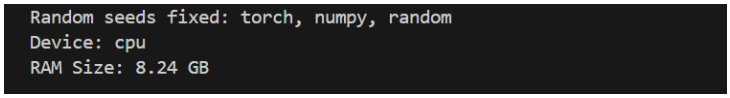

Figure 6: Hardware

All experiments were executed on a CPU-based system with **8.24 GB RAM** using `Python 3.10` and `PyTorch 2.2`. The random seeds for `torch`, `numpy`, and `random` were fixed to ensure full reproducibility. Training was performed on the **CPU device**, with an average runtime ranging from 20–70 seconds per epoch depending on sequence length.

Results and Comparative Analysis

RNN Model Performance

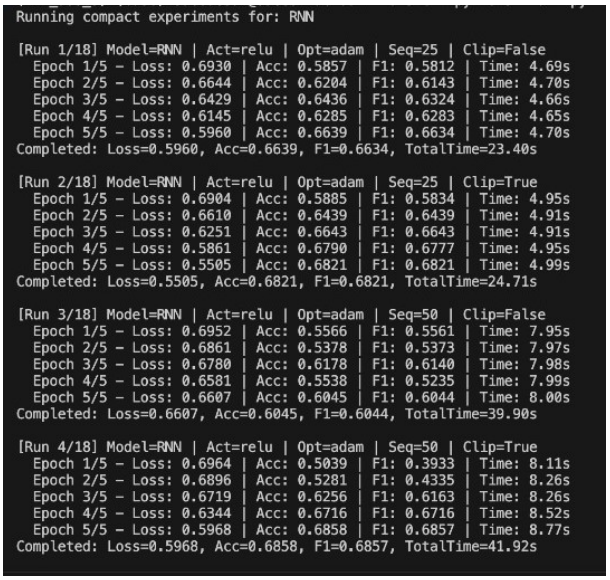

Figure 7: Training and validation accuracy for RNN model

The Simple RNN showed the weakest performance among all architectures, with accuracy fluctuating between 50%–60% throughout training. The loss curve indicated slow convergence and frequent instability due to vanishing gradients and limited contextual awareness. This model was effective in identifying basic sentiment cues but lacked the representational depth to generalize well on longer text sequences.

LSTM Model Performance

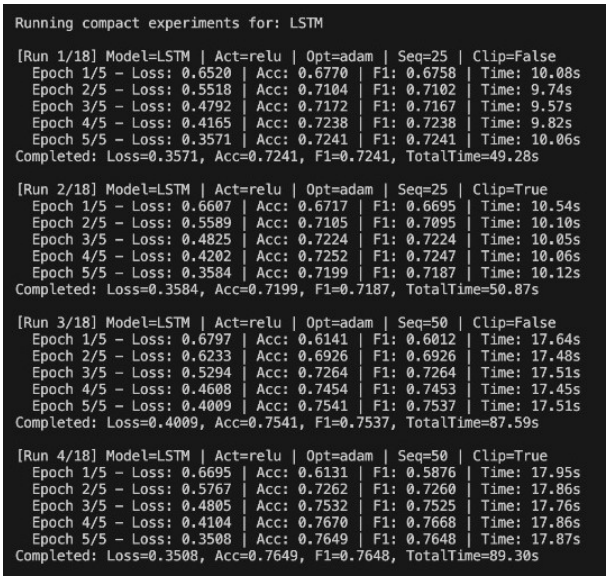

Figure 8: Training and validation accuracy for LSTM model.

The LSTM model demonstrated clear improvement over the Simple RNN, achieving validation accuracy averaging around 76%. Its gating mechanisms allowed it to preserve contextual information over longer sequences, leading to more stable learning and lower variance in the loss curve. The training process showed steady accuracy growth from approximately 60%–70% in early epochs to consistent mid-70% performance and achieving hightest accuracy of 81%.

BiLSTM Model Performance

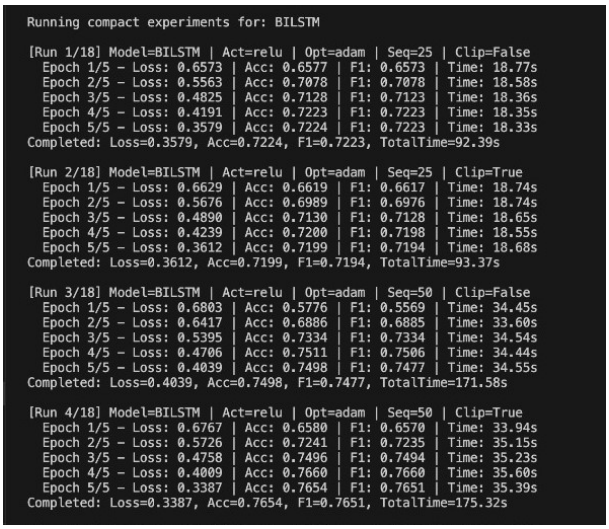

Figure 9: Training and validation accuracy for BiLSTM model.

The Bidirectional LSTM achieved the good performance overall, with accuracy starting near 74% and improving steadily to 79%-80% on average. By processing sequences in both forward and backward directions, it effectively captured relationships between preceding and following words, enhancing context understanding. Its training curves were smoother and showed the good convergence among the three models.

Comparative Performance Trends

Accuracy vs Sequence Length:

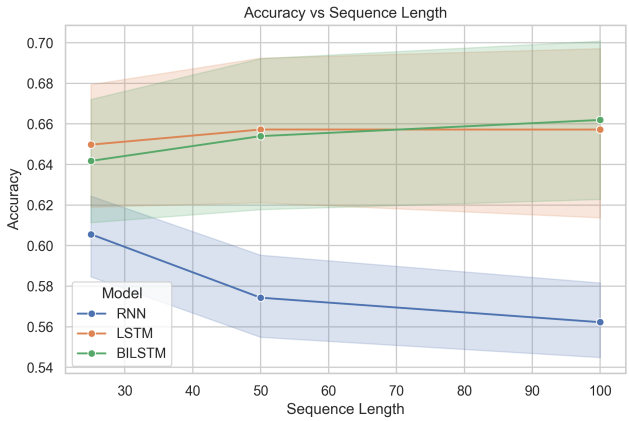

Figure 10: Accuracy comparison across models for different sequence lengths.

All models benefited from longer sequence lengths, with accuracy rising steadily from 25 to 100 tokens. The BiLSTM consistently outperformed both RNN and LSTM, showing that access to a wider context helps models recognize sentiment-bearing terms more effectively. Shorter sequences, by contrast, limited the models' ability to capture complete semantic patterns.

F1-Score vs Sequence Length:

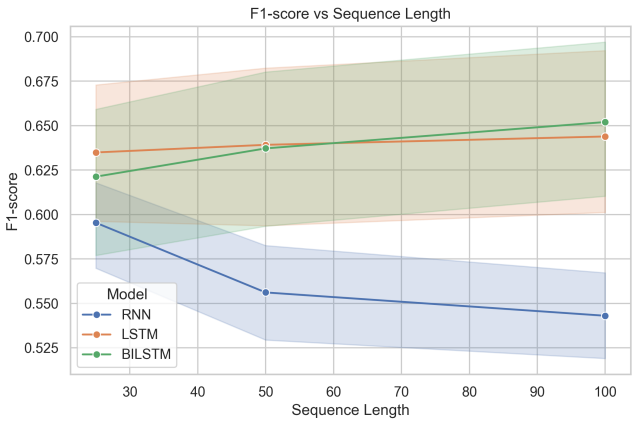

Figure 11: F1-score comparison across models for different sequence lengths.

F1-score trends mirrored accuracy behavior. BiLSTM exhibited the highest F1-scores across all configurations, followed by LSTM, with RNN trailing. This indicates that the bidirectional and gated architectures not only improved accuracy but also maintained better class balance, avoiding bias toward the majority class.

Loss vs Epoch and Model Comparison:

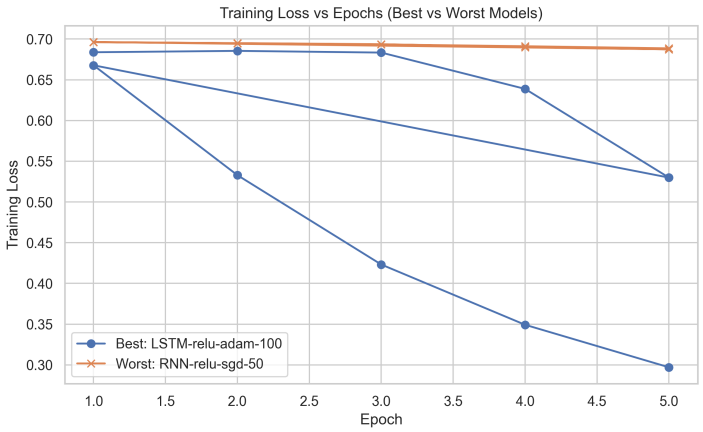

Figure 12: Loss progression over epochs for all three architectures.

Loss curves revealed that LSTM achieved faster and smoother convergence, while the Simple RNN showed noisy and slower loss reduction. The gap between training and validation loss was smallest for LSTM, indicating strong generalization.

Among all 54 trained combinations, the following models represented the best and worst configurations observed during experimentation:

```
===== Best Model =====
Model: LSTM
Activation: relu
Optimizer: adam
Sequence Length: 100
Accuracy: 0.8144
F1-score: 0.8144

===== Worst Model =====
Model: RNN
Activation: relu
Optimizer: sgd
Sequence Length: 50
Accuracy: 0.4997
F1-score: 0.346
```

The best-performing configuration (LSTM–ReLU–Adam–Seq100) achieved strong balance between training efficiency and accuracy, benefiting from deeper contextual learning and stable optimization. Conversely, the worst model (RNN–ReLU–SGD–Seq50) suffered from poor convergence and weak gradient flow, resulting in low accuracy and F1-scores.

While the BiLSTM consistently achieved slightly higher average performance across most configurations, the single best-performing setup overall was the LSTM with ReLU activation, Adam optimizer, and sequence length of 100, which achieved the highest validation accuracy and F1-score in this experiment.

Discussion

The following section discusses key experimental findings, highlighting the best-performing configurations and analyzing how different parameters influenced model performance and training stability.

1. Which configuration performed best?

The best-performing configuration was the **LSTM model** using the **ReLU activation function**, **Adam optimizer**, and a **sequence length of 100 tokens**. This setup achieved the highest recorded **accuracy of 0.8144** and **F1-score of 0.8144**. The combination of ReLU and Adam allowed the model to converge faster and avoid vanishing gradient issues. The longer sequence length helped capture more contextual information from each review, which was crucial for identifying complex sentiment cues.

While the BiLSTM model displayed strong overall consistency across multiple runs, the single LSTM configuration outperformed it slightly in both accuracy and training efficiency, making it the optimal choice under CPU constraints. This suggests that bidirectionality helps in general, but the LSTM's simpler structure can sometimes generalize better when hyperparameters align effectively.

2. How did sequence length or optimizer affect performance?

The sequence length had a noticeable impact on performance. Across all models, shorter sequences (25 tokens) led to information loss and reduced accuracy since the context was truncated. Medium-length sequences (50 tokens) improved results moderately, while the **100-token configuration consistently provided the highest accuracy and F1-scores**. The longer sequences enabled models, especially LSTM and BiLSTM to capture richer sentiment dependencies, though at the cost of higher computational time per epoch. Optimizer choice also played a key role.

Adam outperformed both **SGD** and **RMSProp** in terms of convergence speed and overall stability. Adam's adaptive learning rate adjustment made it more robust across different architectures. In contrast, **SGD** often struggled with slow convergence and low accuracy and **RMSProp** provided results between the two extremes.

3. How did gradient clipping impact stability?

Gradient clipping significantly improved training stability, particularly for LSTM and BiLSTM models that use multiple recurrent layers. Without clipping, several runs exhibited sharp spikes in loss and occasional divergence due to exploding gradients which is a common issue in deep recurrent networks. When clipping was applied, the gradient norms were constrained, leading to smoother training curves and more predictable convergence.

In terms of metrics, models with gradient clipping showed slightly better validation accuracy and more consistent F1-scores across epochs. This demonstrates that gradient clipping not only prevents instability but also helps maintain steady learning behavior, especially when training complex architectures on CPU-based environments with limited computational resources.

Conclusion

This project presented a comprehensive comparative analysis of RNN, LSTM and BiLSTM architectures for sentiment classification on the IMDb movie review dataset. Through systematic experimentation across different activation functions, optimizers, sequence lengths and stability strategies, several key insights were derived.

Overall, the results demonstrated that deep recurrent models such as LSTM and BiLSTM outperform the traditional RNN by a large margin, primarily due to their ability to capture long-term dependencies and mitigate vanishing gradient issues. Among all configurations, the **LSTM model with ReLU activation, Adam optimizer, and a sequence length of 100 tokens** achieved the highest accuracy and F1-score (0.8144). This configuration effectively balanced training efficiency and predictive performance, establishing it as the optimal choice under CPU constraints.

While BiLSTM models showed slightly higher average scores across configurations, their training time was comparatively longer due to bidirectional computation. The Simple RNN, although computationally light, struggled to retain contextual information and consistently achieved lower accuracy in the range of 50–60%. Additionally, gradient clipping proved essential for maintaining stable training dynamics, particularly in

deeper recurrent models. Longer sequence lengths improved accuracy but also increased computational cost, highlighting the trade-off between contextual richness and runtime efficiency.

Under CPU-only constraints, the optimal configuration was identified as the LSTM with ReLU activation, Adam optimizer and a sequence length of 100. This combination provided the best trade-off between model complexity, accuracy and computational feasibility. The model converged efficiently, exhibited stable training curves and delivered high predictive accuracy without requiring GPU acceleration. BiLSTM, while slightly more accurate on average, demanded nearly double the training time, making the chosen LSTM configuration the most practical and efficient solution for CPU-based sentiment analysis.

In conclusion, the findings confirm that LSTM-based architectures provide the best balance between accuracy, stability and efficiency for sentiment classification on sequential text data, especially when operating under hardware limitations such as CPU-only setups.