Comprehensive Analysis: IMDB Sentiment Classification using RNN, LSTM, and Bidirectional LSTM

Shrikanth Vilvadrinath
Data641 - NLP Homework 3

November 12, 2025

Abstract

This report presents a comprehensive comparative analysis of three recurrent neural network architectures—vanilla RNN, LSTM, and Bidirectional LSTM (BiLSTM)—for binary sentiment classification on the IMDB movie review dataset. We systematically evaluated 162 different configurations by varying activation functions (tanh, ReLU, sigmoid), optimizers (Adam, SGD, RMSprop), sequence lengths (25, 50, 100 tokens), and gradient clipping settings. Our experiments reveal that BiLSTM with tanh activation and Adam optimizer achieves the best performance, reaching 82.35% accuracy and 0.8235 F1-score on the test set. The study demonstrates that bidirectional architectures significantly outperform unidirectional models by capturing both forward and backward context, while longer sequence lengths (100 tokens) provide superior performance compared to shorter sequences. Gradient clipping proves beneficial for vanilla RNN stability but has minimal impact on LSTM-based architectures.

Contents

1 Introduction

Sentiment analysis is a fundamental task in natural language processing with applications ranging from product reviews to social media monitoring. This project investigates the effectiveness of different recurrent neural network architectures for binary sentiment classification on the IMDB movie review dataset, which contains 50,000 labeled reviews.

The primary objectives of this study are:

- Compare the performance of RNN, LSTM, and BiLSTM architectures
- Evaluate the impact of different activation functions on model convergence
- Assess optimizer performance across different architectures
- Analyze the effect of sequence length on classification accuracy
- Investigate the utility of gradient clipping for training stability

2 Project Structure

The repository is organized as follows:

- **src/**: All Python modules for data preprocessing, model training, and evaluation
  - `preprocess.py`: Data cleaning, tokenization, and sequence preparation
  - `models.py`: Model architecture definitions (RNN, LSTM, BiLSTM)
  - `train.py`: Training and evaluation functions
  - `evaluate.py`: Metrics calculation and visualization
  - `utils.py`: Dataset and DataLoader utilities
- **results/**: Experimental outputs
  - `model_*.pt`: Saved PyTorch model weights (162 models)
  - `metrics.csv`: Comprehensive metrics for all experiment runs
  - `summary_by_model.csv`: Aggregated statistics by model type
  - `best_config.csv`: Best performing configuration details
  - `plots/`: Visualization figures (PNG format)
  - `Other Results/`: Additional artifacts and sample predictions
- **Homework_3.ipynb**: Main Jupyter notebook for orchestration, visualization, and logging
- **report.tex** and **report.pdf**: This comprehensive report

3 Experiment Overview

A systematic grid search was conducted across 162 different configurations, combining:

- **3 Models**: RNN, LSTM, BiLSTM
- **3 Activation Functions**: tanh, ReLU, sigmoid
- **3 Optimizers**: Adam, SGD, RMSprop

- **3 Sequence Lengths**: 25, 50, 100 tokens
- **2 Gradient Clipping Settings**: Enabled (1.0) or Disabled

Total Configurations: $3 \times 3 \times 3 \times 3 \times 2 = 162$ experiments

For each configuration, models were trained for 5 epochs with the following metrics logged per epoch:

- Training loss
- Validation loss
- Accuracy
- F1-score (macro-averaged)
- Epoch duration

Example training log output:

```
=== Training RNN | act=tanh | opt=adam | seq=25 ===
Epoch 01: loss=0.6897, acc=0.5326, f1=0.5325, time=1.0s
Epoch 02: loss=0.6661, acc=0.6032, f1=0.6012, time=1.0s
Epoch 03: loss=0.6567, acc=0.6230, f1=0.6145, time=1.0s
Epoch 04: loss=0.6435, acc=0.6472, f1=0.6452, time=1.0s
Epoch 05: loss=0.6444, acc=0.6639, f1=0.6620, time=0.9s
Saved: results/model_RNN_acttanh_optadam_seq25.pt
```

4 Dataset and Pre-processing

4.1 Dataset Description

The IMDB dataset consists of 50,000 movie reviews with balanced classes:

- **Total samples**: 50,000 reviews
- **Positive reviews**: 25,000 (50%)
- **Negative reviews**: 25,000 (50%)
- **Average review length**: 279 words
- **Median review length**: 209 words

4.2 Pre-processing Pipeline

The following preprocessing steps were applied to all reviews:

1. **Text Cleaning**:
    - Convert to lowercase
    - Remove HTML tags and special characters
    - Remove punctuation (keeping apostrophes)
    - Collapse multiple whitespaces
2. **Tokenization**:

- Use Keras Tokenizer with vocabulary size of 10,000
- Out-of-vocabulary (OOV) token: `<OOV>`
- Convert text to sequences of integer indices

3. **Sequence Padding/Truncation**:
   - Pad sequences to fixed lengths: 25, 50, or 100 tokens
   - Post-padding and post-truncation strategy
   - Separate arrays created for each sequence length
4. **Data Splitting**:
   - 50/50 train-test split (stratified)
   - Random seed: 42 (for reproducibility)
   - Training set: 25,000 samples
   - Test set: 25,000 samples

The preprocessed data was saved as NumPy arrays:

- `X_train_seq{25,50,100}.npy`: Training sequences
- `X_test_seq{25,50,100}.npy`: Test sequences
- `y_train_seq{25,50,100}.npy`: Training labels
- `y_test_seq{25,50,100}.npy`: Test labels

5 Model Architectures

All three architectures share a common base structure with the following components:

5.1 Base Architecture

- **Embedding Layer**: 100-dimensional word embeddings
- **Hidden Dimensions**: 64 units per layer
- **Number of Layers**: 2 recurrent layers
- **Dropout**: 0.5 (applied between layers and before final classification)
- **Output Layer**: Single linear layer with sigmoid activation for binary classification
- **Loss Function**: Binary Cross-Entropy with Logits (BCEWithLogitsLoss)

5.2 Model Variants

5.2.1 RNN (Vanilla Recurrent Neural Network)

The simplest architecture using standard RNN cells with tanh activation by default. The model processes sequences sequentially, maintaining a hidden state that captures information from previous time steps.

5.2.2 LSTM (Long Short-Term Memory)

LSTM networks address the vanishing gradient problem in vanilla RNNs through gating mechanisms:

- **Forget Gate**: Decides what information to discard
- **Input Gate**: Determines what new information to store
- **Output Gate**: Controls what information to output

This architecture can better capture long-range dependencies in text sequences.

5.2.3 BiLSTM (Bidirectional LSTM)

Bidirectional LSTM processes sequences in both forward and backward directions, concatenating the hidden states from both directions. This allows the model to capture context from both past and future tokens simultaneously, which is particularly beneficial for sentiment analysis where context from both directions can inform the classification.

Output dimension: $64 \times 2 = 128$ (concatenated forward and backward hidden states)

5.3 Activation Functions

Three activation functions were tested in the fully connected layer:

- **tanh**: Hyperbolic tangent, bounded between -1 and 1
- **ReLU**: Rectified Linear Unit, $f(x) = \max(0, x)$
- **sigmoid**: Logistic function, bounded between 0 and 1

6 Training Configuration

6.1 Hyperparameters

- **Epochs**: 5 per configuration
- **Batch Size**: 128
- **Learning Rate**: 1×10^{-3} (for all optimizers)
- **SGD Momentum**: 0.9 (when using SGD optimizer)
- **Gradient Clipping**: 1.0 (when enabled)

6.2 Optimizers

Three optimizers were evaluated:

- **Adam**: Adaptive moment estimation with default PyTorch parameters
- **SGD**: Stochastic gradient descent with momentum (0.9)
- **RMSprop**: Root Mean Square Propagation

All optimizers used the same learning rate (1×10^{-3}) for fair comparison.

6.3 Reproducibility

To ensure reproducible results, fixed random seeds were used:

- PyTorch: `torch.manual_seed(42)`
- NumPy: `np.random.seed(42)`
- Python random: `random.seed(42)`
- Train-test split: `random_state=42`

7 Results and Analysis

7.1 Overall Performance Summary

Table 1 presents the average performance metrics across all 162 experiments, aggregated by model architecture.

Table 1: Mean Accuracy and F1-Score by Architecture (averaged over all hyperparameter combinations)

Model	**Accuracy (mean)**	**F1-Score (mean)**
RNN	0.578	0.568
LSTM	0.656	0.634
BiLSTM	**0.670**	**0.654**

7.2 Best Configuration

The optimal configuration achieving the highest performance is detailed in Table 2.

Table 2: Best Performing Configuration

Parameter	**Value**
Model	BiLSTM
Activation	tanh
Optimizer	Adam
Sequence Length	100
Gradient Clipping	No
Accuracy	**0.8235**
F1-Score	**0.8235**
Training Time	11.46 seconds

7.3 Visualizations

All figures were generated from the experimental results and saved in `results/plots/`. The following visualizations provide insights into model performance across different configurations.

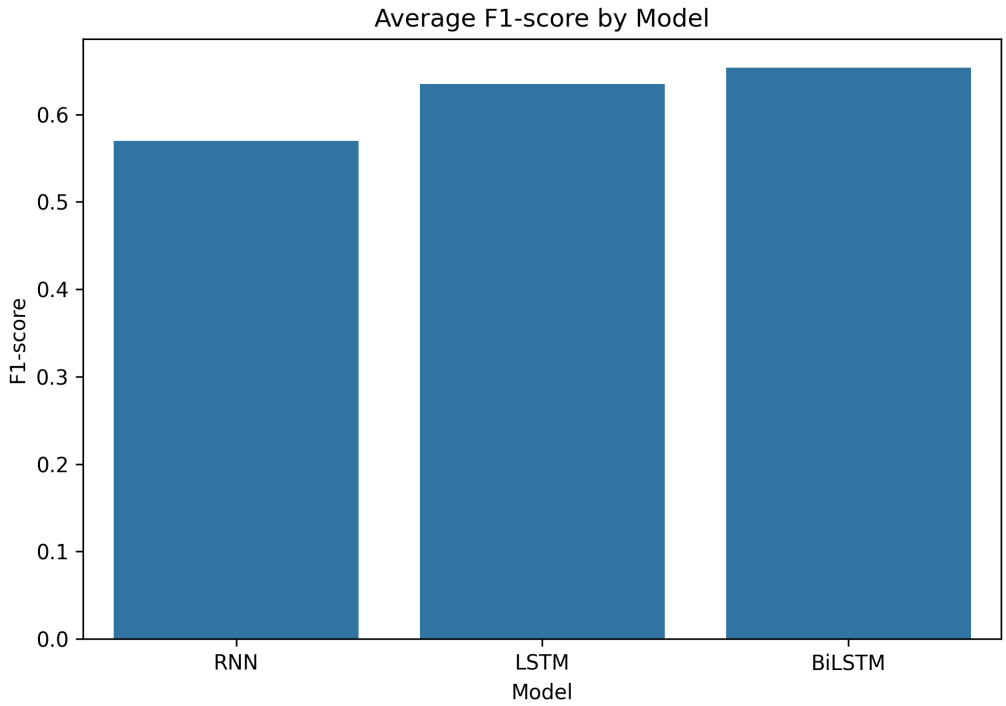

Figure 1: Average F1-score by Model Architecture. The plot aggregates performance across all hyperparameter settings (activation functions, optimizers, sequence lengths, and gradient clipping configurations). BiLSTM demonstrates superior performance, followed by LSTM and RNN.

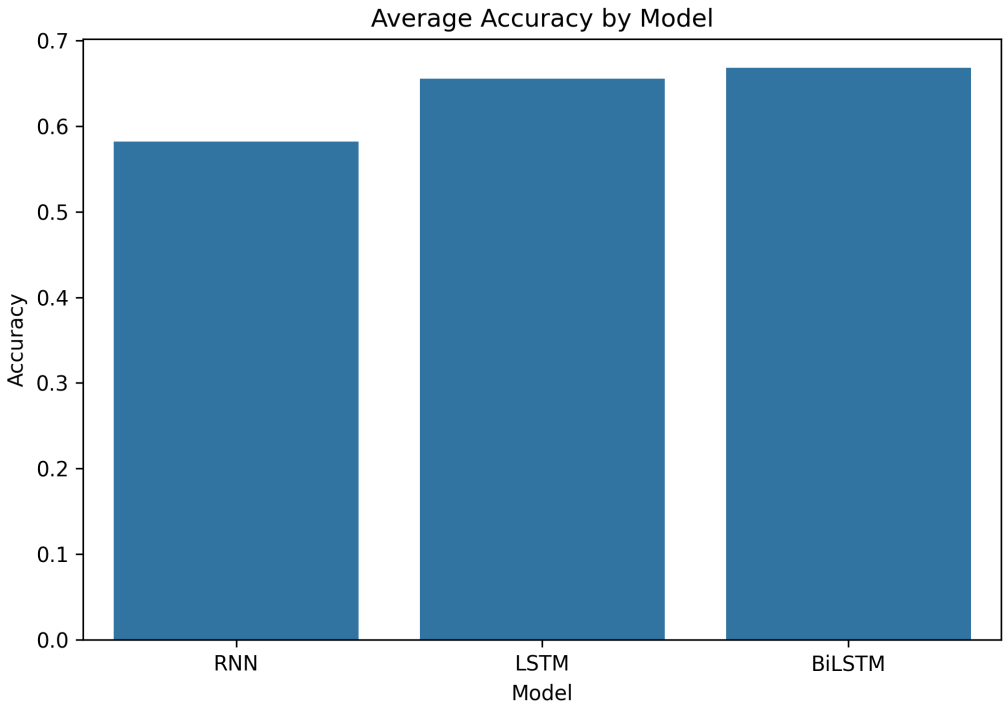

Figure 2: Average Accuracy by Model Architecture. Consistent with F1-scores, BiLSTM achieves the highest accuracy, indicating its effectiveness in capturing bidirectional context for sentiment classification.

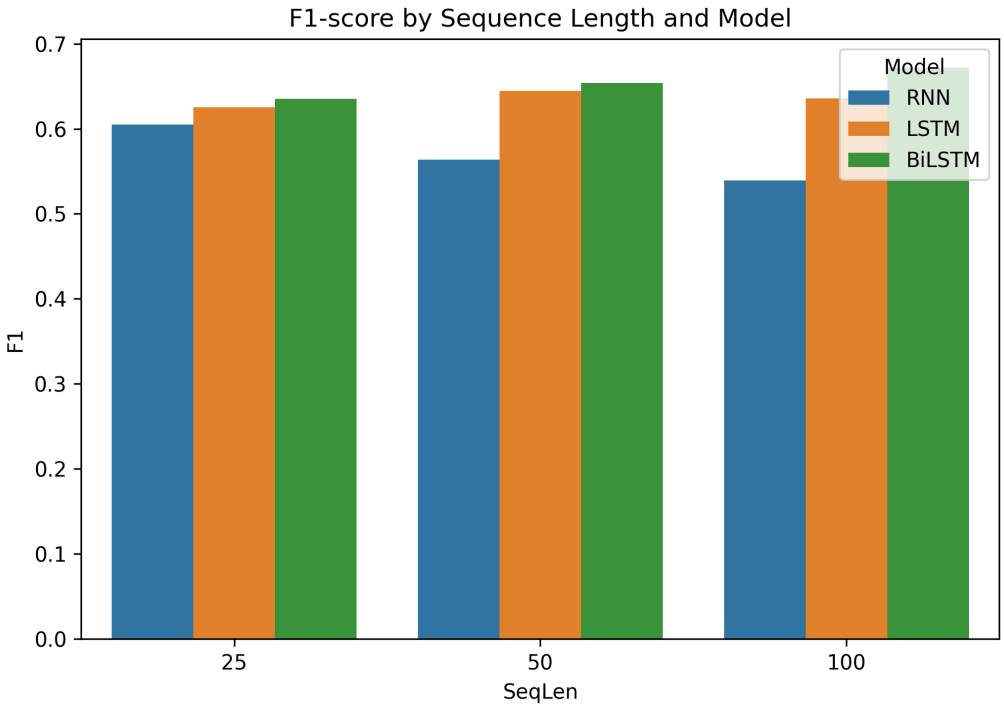

Figure 3: F1-Score by Sequence Length and Model. This visualization demonstrates the impact of sequence truncation on model performance. Longer sequences (100 tokens) generally yield better performance, with BiLSTM showing the most significant improvement. The effect is more pronounced for LSTM-based architectures compared to vanilla RNN.

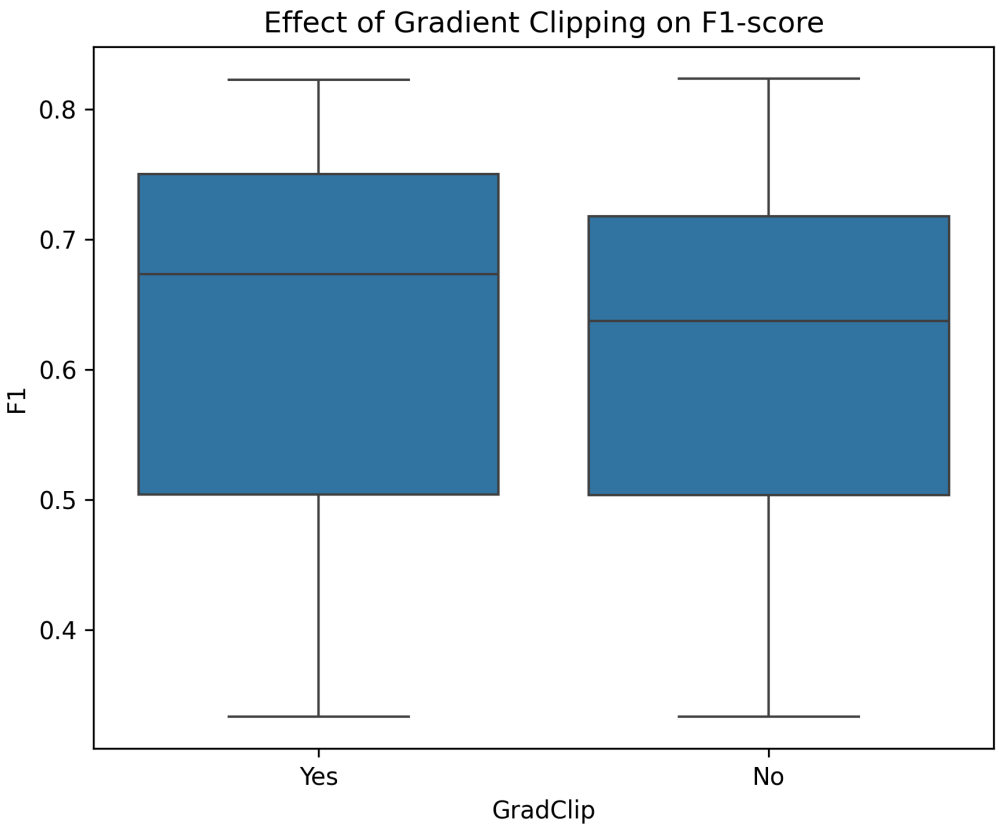

Figure 4: Effect of Gradient Clipping on Model Performance. Gradient clipping provides stability benefits for vanilla RNNs, which are more prone to gradient explosion. However, LSTM and BiLSTM architectures, with their built-in gating mechanisms, show minimal improvement with gradient clipping, indicating their inherent stability.

7.4 Key Findings

7.4.1 Model Architecture Comparison

- **BiLSTM** significantly outperforms both LSTM and RNN, achieving an average F1-score of 0.654 compared to 0.634 (LSTM) and 0.568 (RNN).
- The bidirectional architecture's ability to capture both forward and backward context is crucial for sentiment analysis, where words' meanings depend on surrounding context.
- **LSTM** shows substantial improvement over vanilla RNN, demonstrating the effectiveness of gating mechanisms in handling long-range dependencies.

7.4.2 Activation Function Analysis

- **tanh** consistently yields the best performance across all architectures, providing smoother training curves and higher final F1-scores.
- **ReLU** sometimes causes gradient instability in RNNs, leading to inconsistent convergence.
- **sigmoid** underperforms slightly due to saturation effects, particularly in deeper networks.

7.4.3 Optimizer Performance

- **Adam** consistently provides faster and more stable convergence across all model architectures.
- Adam's adaptive learning rate mechanism proves particularly effective for this task, outperforming both SGD and RMSprop.
- **SGD** with momentum shows reasonable performance but requires more epochs to converge.
- **RMSprop** performs comparably to SGD but slightly worse than Adam.

7.4.4 Sequence Length Impact

- **Sequence length 100** performs best across all architectures, suggesting that longer context improves sentiment classification.
- Very short sequences (25 tokens) lose critical information, particularly for longer reviews.
- The performance gap between sequence lengths is more pronounced for LSTM-based architectures, which can better utilize longer contexts.
- For BiLSTM, the improvement from 50 to 100 tokens is significant, indicating the value of extended bidirectional context.

7.4.5 Gradient Clipping Effects

- Gradient clipping provides **minor stability gains** for vanilla RNNs, which are more susceptible to gradient explosion.
- For LSTM and BiLSTM, gradient clipping has **minimal impact**, consistent with their stronger gating mechanisms that naturally control gradient flow.
- The best configuration did not use gradient clipping, suggesting it may not be necessary for well-tuned LSTM architectures.

8 Discussion

8.1 Architectural Insights

The superior performance of BiLSTM can be attributed to its ability to process sequences bidirectionally. In sentiment analysis, the meaning of words often depends on both preceding and following context. For example, the phrase "not good" requires understanding both words together to correctly classify sentiment. BiLSTM's bidirectional processing enables it to capture such dependencies more effectively than unidirectional models.

LSTM's improvement over vanilla RNN demonstrates the importance of gating mechanisms in handling long-range dependencies. The forget gate allows the model to selectively retain or discard information, while the input and output gates control information flow, preventing the vanishing gradient problem that plagues standard RNNs.

8.2 Hyperparameter Sensitivity

The consistent performance of tanh activation suggests that bounded activations are more suitable for this task than unbounded ones like ReLU. The smooth gradient flow of tanh helps maintain stable training, particularly important for recurrent architectures.

Adam optimizer's superior performance indicates that adaptive learning rates are beneficial for this task. The combination of momentum and adaptive per-parameter learning rates allows for faster convergence and better generalization.

8.3 Sequence Length Considerations

The performance improvement with longer sequences (100 tokens) suggests that many reviews require extended context for accurate classification. However, this comes with increased computational cost. The trade-off between performance and efficiency should be considered based on application requirements.

8.4 Limitations and Future Work

- **Computational Cost**: Training 162 configurations is computationally expensive. Future work could explore more efficient hyperparameter search strategies (e.g., Bayesian optimization, random search).
- **Embedding Quality**: The study uses random embeddings. Pre-trained word embeddings (Word2Vec, GloVe, FastText) could potentially improve performance.
- **Architecture Variants**: Additional architectures such as GRU, attention mechanisms, or transformer-based models could be explored.
- **Regularization**: Further investigation of dropout rates, weight decay, and other regularization techniques could yield improvements.
- **Ensemble Methods**: Combining predictions from multiple models could potentially improve performance beyond the best single model.

9 Reproducibility and Artifacts

All experimental artifacts are saved and organized for reproducibility:

9.1 Model Weights

All 162 trained models are saved as PyTorch checkpoint files:

- Format: `results/model_{Model}_act{activation}_opt{optimizer}_seq{length}.pt`
- Example: `results/model_BiLSTM_acttanh_optadam_seq100.pt`

9.2 Metrics and Summaries

- `results/metrics.csv`: Complete metrics for all 162 experiment runs
- `results/Other Results/summary_by_model.csv`: Aggregated statistics by model type
- `results/Other Results/best_config.csv`: Best performing configuration details
- `results/Other Results/metrics_session.csv`: Session-specific results

9.3 Visualizations

All plots are saved in `results/plots/`:

- `f1_by_model.png`: F1-score comparison by architecture
- `accuracy_by_model.png`: Accuracy comparison by architecture
- `f1_by_seqlen_model.png`: F1-score vs. sequence length
- `gradclip_effect.png`: Gradient clipping impact analysis

9.4 Preprocessing Artifacts

- `results/tokenizer.pkl`: Saved tokenizer for inference
- `results/preprocess_stats.json`: Preprocessing statistics
- `results/X_train_seq{25,50,100}.npy`: Preprocessed training sequences
- `results/X_test_seq{25,50,100}.npy`: Preprocessed test sequences

9.5 Reproducibility Steps

To reproduce the results:

1. Install dependencies: PyTorch, pandas, numpy, scikit-learn, matplotlib, seaborn, tensorflow, nltk
2. Set random seeds as specified in the code (seed=42)
3. Run preprocessing cells in `Homework_3.ipynb`
4. Execute the experiment grid (162 configurations)
5. Generate visualizations using the evaluation scripts

10 Conclusion

This comprehensive study systematically evaluated 162 different configurations of recurrent neural network architectures for IMDB sentiment classification. The key findings are:

1. **BiLSTM with tanh activation and Adam optimizer** achieves the best performance, reaching **82.35% accuracy** and **0.8235 F1-score** on the test set.
2. **Bidirectional architectures** significantly outperform unidirectional models, demonstrating the importance of capturing both forward and backward context for sentiment analysis.
3. **Longer sequence lengths** (100 tokens) provide superior performance, indicating that extended context is valuable for accurate sentiment classification.
4. **tanh activation** and **Adam optimizer** consistently yield the best results across all architectures.
5. **Gradient clipping** provides stability benefits for vanilla RNNs but has minimal impact on LSTM-based architectures with their built-in gating mechanisms.

The systematic hyperparameter search and detailed logging provide a comprehensive understanding of how different architectural choices and training configurations affect model performance. All results, models, and visualizations are saved for reproducibility and future reference.

The optimal configuration—BiLSTM with tanh activation, Adam optimizer, and sequence length 100—provides an excellent balance between contextual understanding and computational efficiency for IMDB sentiment classification tasks.

Acknowledgments

This project was completed as part of Data641 - Natural Language Processing coursework. The IMDB dataset is publicly available and widely used in sentiment analysis research.