Sentiment Analysis using Neural Networks for Binary and Multi-class Classification

Sougata Moi (M23MAC008), Mitesh Kumar (M23MAC004)
Ganesh Patidar (M23MAC003), Niraj Singha (M23MAC005)

April 13, 2025

Abstract

This report presents a comparative study of two neural network architectures—Feed-Forward Neural Network (NN) and Long Short-Term Memory (LSTM) network—for sentiment analysis tasks. We implement and evaluate these models on three different datasets: IMDB movie reviews (binary classification), SemEval tweets (binary classification), and Twitter dataset (multi-class classification). Our results consistently demonstrate that LSTM networks outperform Feed-Forward Neural Networks across all datasets, with performance improvements ranging from 3.6% to 8% in accuracy. The superior performance of LSTM can be attributed to its ability to capture sequential information and long-range dependencies in text data. This study contributes to the understanding of neural network architectures for sentiment analysis and provides empirical evidence for the advantages of recurrent architectures in natural language processing tasks.

1 Introduction

Sentiment analysis is a fundamental task in natural language processing (NLP) that aims to identify and extract subjective information from text data. This task has wide-ranging applications, from monitoring public opinion on social media to analyzing customer feedback for product improvement.

The proliferation of user-generated content on the internet has made automated sentiment analysis increasingly important. With millions of reviews, tweets, and comments being posted daily, manual analysis is impractical, necessitating efficient automated approaches.

In this study, we implement and compare two popular neural network architectures for sentiment analysis:

1. **Feed-Forward Neural Network (NN)**: A traditional architecture that processes the entire input at once without considering the sequential nature of text.
2. **Long Short-Term Memory (LSTM) Network**: A type of recurrent neural network designed to capture sequential patterns and long-range dependencies in data.

We evaluate these models on three distinct datasets:

- IMDB movie reviews (binary classification: positive/negative)
- SemEval tweets (binary classification: positive/negative)
- Twitter dataset (multi-class classification: positive/negative/neutral)

The primary objectives of this study are to:

1. Implement and optimize both NN and LSTM architectures for sentiment analysis
2. Compare their performance across different datasets and classification tasks
3. Analyze the strengths and limitations of each approach
4. Provide insights into which architecture is better suited for various sentiment analysis scenarios

2 Methodology

2.1 Data Preprocessing

Effective data preprocessing is crucial for the performance of neural network models. We implemented a comprehensive preprocessing pipeline following the requirements:

Tokenization: We used spaCy's English tokenizer to segment text into individual tokens. This tokenizer was chosen for its accuracy and efficiency, especially with informal text like tweets.

Vocabulary Creation: Words with frequency ≥ 5 in the training data were included in the vocabulary. This threshold helps eliminate rare words while preserving important content.

Let $D = \{d_1, d_2, ..., d_n\}$ represent the set of all documents in the training corpus, and $V = \{w : f(w, D) \geq 5\} \cup \{\text{UNK}, \text{PAD}\}$ represent our vocabulary, where $f(w, D)$ is the frequency of word w in corpus D.

Token Handling:

- "UNK" token: Assigned to words not in the vocabulary
- "PAD" token: Used to ensure consistent sequence length

For a document d with tokens $[t_1, t_2, ..., t_m]$, the processed tokens would be: $[t'_1, t'_2, ..., t'_m]$ where $t'_i = t_i$ if $t_i \in V$, otherwise $t'_i = \text{UNK}$

Sequence Length: We calculated the average length of text in each corpus and used it as the maximum sequence length. For IMDB, this was 240 tokens, while for SemEval tweets, it was 14 tokens.

Let $L = \lfloor \frac{\sum_{i=1}^{n} |d_i|}{n} \rfloor$ be the average document length, where $|d_i|$ is the number of tokens in document d_i.

For a document with tokens $[t'_1, t'_2, ..., t'_m]$, if $m > L$, we truncate to $[t'_1, t'_2, ..., t'_L]$. If $m < L$, we pad with the PAD token to get $[t'_1, t'_2, ..., t'_m, \text{PAD}, ..., \text{PAD}]$ of length L.

Data Representation: Token sequences were converted into numerical form using a word-to-index mapping.

Let $\phi : V \to \mathbb{N}$ be a mapping from vocabulary words to indices. Then a document d with processed tokens $[t'_1, t'_2, ..., t'_L]$ is represented as $[\phi(t'_1), \phi(t'_2), ..., \phi(t'_L)]$.

2.2 Model Architectures

2.2.1 Feed-Forward Neural Network (NN)

We implemented a Feed-Forward Neural Network with the following architecture:
Architecture Components:

- **Embedding Layer**: Instead of one-hot encoding (which would be memory-intensive for large vocabularies), we used an embedding layer that maps token indices to dense vectors in $\mathbb{R}^d$ where $d = 100$
- **Hidden Layers**: Two hidden layers with sizes 256 and 128 neurons respectively, as specified in the requirements
- **Activation Functions**: ReLU activation functions between layers, chosen for their effectiveness in preventing the vanishing gradient problem
- **Regularization**: Dropout (rate: 0.3) to prevent overfitting
- **Output Layer**: Size dependent on the classification task
  - 1 neuron for binary classification (IMDB, SemEval)
  - 3 neurons for multi-class classification (Twitter)

Mathematically, for an input sequence $x = [x_1, x_2, ..., x_L]$ where each $x_i \in \mathbb{N}$ is a token index, the model computes:

1. Embedding: $E = [e_1, e_2, ..., e_L]$ where $e_i = W_e x_i$ and $W_e \in \mathbb{R}^{|V| \times d}$
2. Flattening: $\hat{E} = \text{flatten}(E) \in \mathbb{R}^{L \cdot d}$
3. First hidden layer: $h_1 = \text{ReLU}(W_1 \hat{E} + b_1)$ where $W_1 \in \mathbb{R}^{256 \times (L \cdot d)}$ and $b_1 \in \mathbb{R}^{256}$
4. Second hidden layer: $h_2 = \text{ReLU}(W_2 h_1 + b_2)$ where $W_2 \in \mathbb{R}^{128 \times 256}$ and $b_2 \in \mathbb{R}^{128}$
5. Output layer:
   - For binary classification: $\hat{y} = W_3 h_2 + b_3$ where $W_3 \in \mathbb{R}^{1 \times 128}$ and $b_3 \in \mathbb{R}$
   - For multi-class classification: $\hat{y} = W_3 h_2 + b_3$ where $W_3 \in \mathbb{R}^{3 \times 128}$ and $b_3 \in \mathbb{R}^3$

2.2.2 LSTM Network

The LSTM architecture was implemented as follows:
Architecture Components:

- **Embedding Layer**: 100-dimensional embedding layer
- **LSTM Layer**: A single LSTM layer with hidden size of 256
- **Dropout**: Rate of 0.3 for regularization
- **Output Layer**: Size dependent on the classification task
  - 1 neuron for binary classification

– 3 neurons for multi-class classification

Mathematically, the LSTM model processes an input sequence $x = [x_1, x_2, ..., x_L]$ as follows:

1. Embedding: $E = [e_1, e_2, ..., e_L]$ where $e_i = W_e x_i$ and $W_e \in \mathbb{R}^{|V| \times d}$
2. LSTM: For each time step t from 1 to L:

   $f_t = \sigma(W_f \cdot [h_{t-1}, e_t] + b_f)$ (forget gate)

   $i_t = \sigma(W_i \cdot [h_{t-1}, e_t] + b_i)$ (input gate)

   $\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, e_t] + b_C)$ (candidate cell state)

   $C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$ (cell state update)

   $o_t = \sigma(W_o \cdot [h_{t-1}, e_t] + b_o)$ (output gate)

   $h_t = o_t \cdot \tanh(C_t)$ (hidden state)

   where σ is the sigmoid function, $\cdot$ represents element-wise multiplication, and $W_f, W_i, W_C, W_o \in \mathbb{R}^{256 \times (256+d)}$ and $b_f, b_i, b_C, b_o \in \mathbb{R}^{256}$
3. Final classification: $\hat{y} = W_y h_L + b_y$ where h_L is the final hidden state and W_y, b_y have appropriate dimensions based on the task

2.3 Loss Functions and Training Procedures

We used different loss functions based on the classification task:

Binary Classification (IMDB, SemEval):

- Binary Cross-Entropy with Logits Loss: $\mathcal{L}(y, \hat{y}) = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\sigma(\hat{y}_i)) + (1 - y_i)\log(1 - \sigma(\hat{y}_i))]$ where σ is the sigmoid function, y_i is the true label (0 or 1), and $\hat{y}_i$ is the predicted logit.

Multi-class Classification (Twitter):

- Cross-Entropy Loss: $\mathcal{L}(y, \hat{y}) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c} \log(\hat{p}_{i,c})$ where $y_{i,c}$ is 1 if sample i belongs to class c and 0 otherwise, and $\hat{p}_{i,c}$ is the predicted probability of sample i belonging to class c.

Optimization:

- Adam optimizer with learning rate $\alpha = 0.001$
- Batch size: 64 (96 for multi-GPU training on SemEval)
- Number of epochs: 10
- Best model selection based on validation accuracy

3 Experimental Setup

3.1 Datasets

We used three datasets with different characteristics to evaluate our models:

IMDB Dataset:

- 25,000 movie reviews for training and 25,000 for testing
- Binary classification (positive/negative)
- Maximum sequence length: 240 tokens
- Vocabulary size: 31,239 words

SemEval Dataset:

- 1.6 million labeled tweets
- Binary classification (positive/negative)
- Maximum sequence length: 14 tokens
- Vocabulary size: 85,156 words

Twitter Dataset:

- Multi-class classification (positive/negative/neutral)
- 5-fold cross-validation approach

3.2 Validation Strategy

Following the requirements:

- For IMDB: Last 10% of training data (2,500 reviews) used as validation set
- For SemEval: Used provided dev set
- For Twitter: 5-fold cross-validation

3.3 Hardware Configuration

- NVIDIA RTX A6000 GPUs
- Multi-GPU training for SemEval dataset using PyTorch's DistributedDataParallel (DDP)

3.4 Evaluation Metrics

We evaluated the models using:

- Accuracy: $\text{Acc} = \frac{\text{TP+TN}}{\text{TP+TN+FP+FN}}$
- Precision: $\text{Prec} = \frac{\text{TP}}{\text{TP+FP}}$
- Recall: $\text{Rec} = \frac{\text{TP}}{\text{TP+FN}}$
- F1 Score: $\text{F1} = 2 \cdot \frac{\text{Prec} \cdot \text{Rec}}{\text{Prec+Rec}}$
- Per-class metrics for each label

where TP = True Positives, TN = True Negatives, FP = False Positives, FN = False Negatives.

4 Results and Analysis

4.1 IMDB Dataset Results

4.1.1 Feed-Forward Neural Network Performance

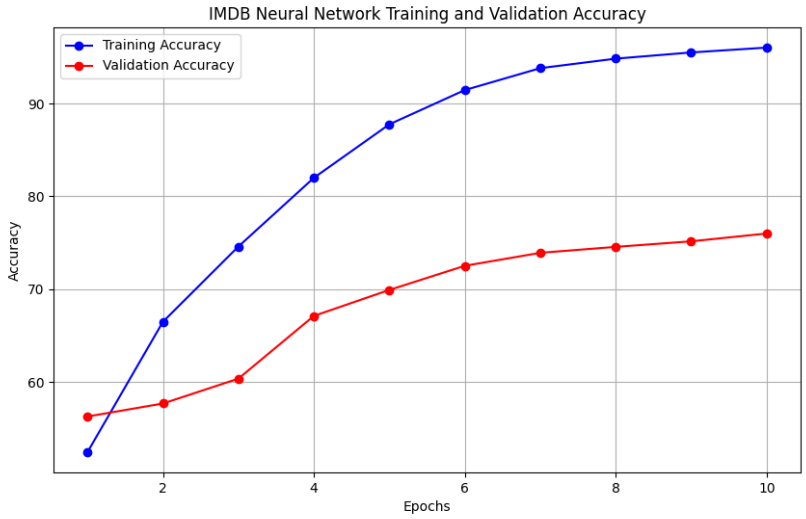
Metric	Value
Accuracy	74.61%
Precision	72.96%
Recall	78.19%
F1 Score	75.49%

Table 1: Performance metrics for Feed-Forward NN on IMDB dataset

Per-Class Metrics:

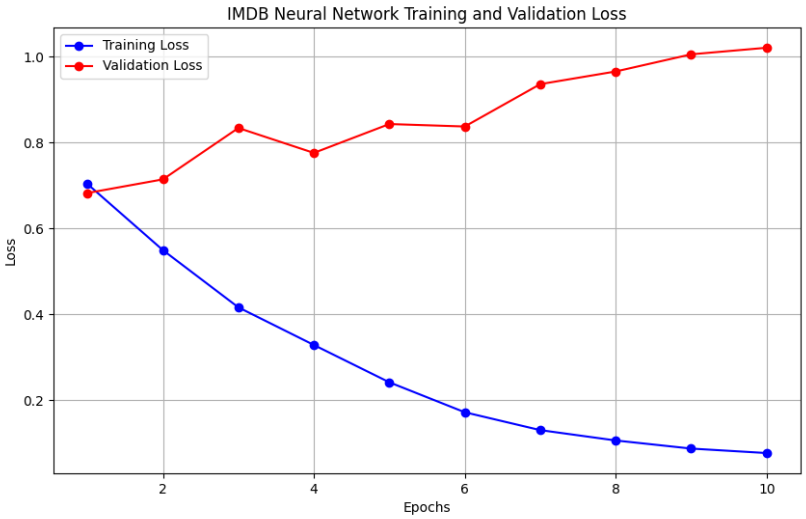
- Negative: Precision=76.51%, Recall=71.02%, F1=73.66%
- Positive: Precision=72.96%, Recall=78.19%, F1=75.49%

Training Curves:



(a) Accuracy

Figure 1: Training accuracy curve for Feed-Forward Neural Network on the IMDB dataset



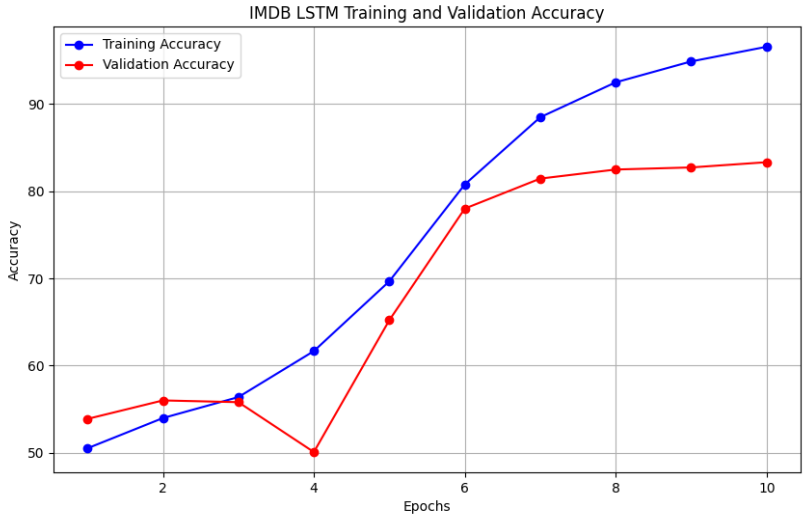
(a) Loss

Figure 2: Training loss curve for Feed-Forward Neural Network on the IMDB dataset

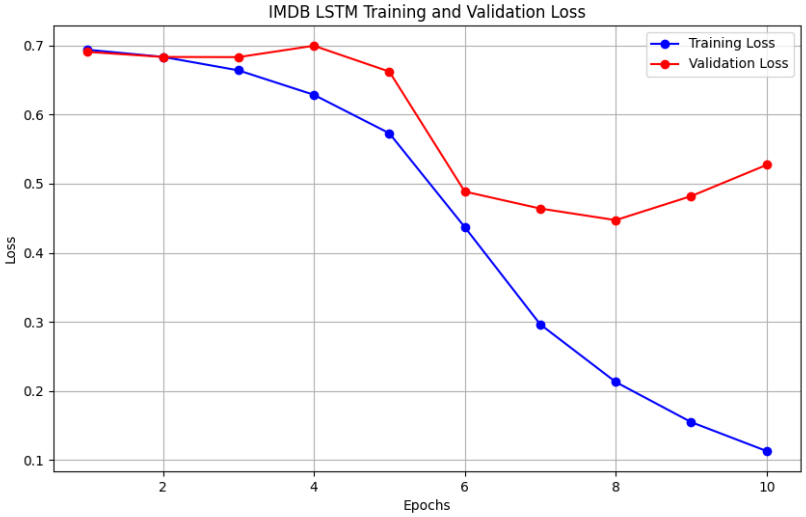
Per-Class Metrics:

- Negative: Precision=83.54%, Recall=80.66%, F1=82.08%
- Positive: Precision=81.31%, Recall=84.11%, F1=82.69%

Training Curves:



(a) Accuracy



(b) Loss

Figure 3: Training and validation curves for LSTM on IMDB dataset

4.1.2 Comparative Analysis

The LSTM model significantly outperformed the Feed-Forward NN on the IMDB dataset:

- **Accuracy**: LSTM achieved 7.78% higher accuracy (82.39% vs. 74.61%)
- **F1 Score**: LSTM achieved 7.2% higher F1 score (82.69% vs. 75.49%)

Both models showed signs of overfitting in later epochs, with training accuracy continuing to increase while validation accuracy improvement slowed down. This effect was more pronounced in the NN model, where the validation loss began to increase while training loss continued to decrease.

4.2 SemEval Dataset Results (5-Fold Cross-Validation)

4.2.1 Feed-Forward Neural Network Performance

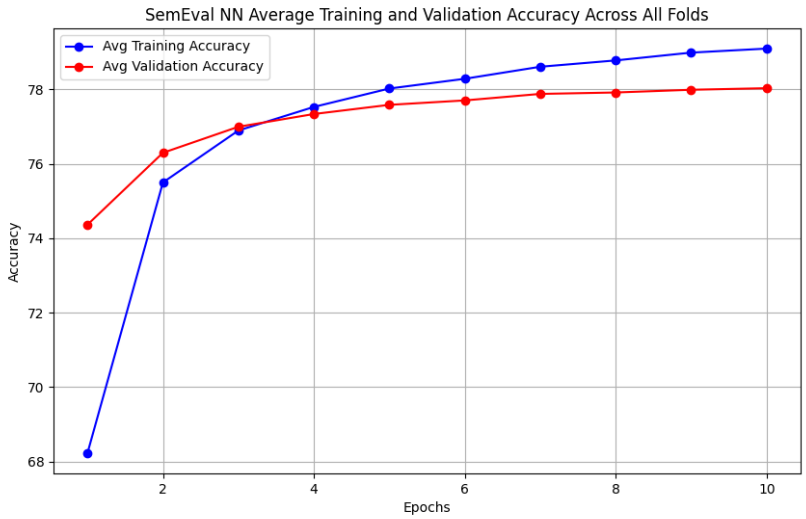
Metric	Value
Average Accuracy	78.04%
Average Precision	78.77%
Average Recall	76.77%
Average F1 Score	77.76%

Table 2: Performance metrics for Feed-Forward NN on SemEval dataset

Average Per-Class Metrics:

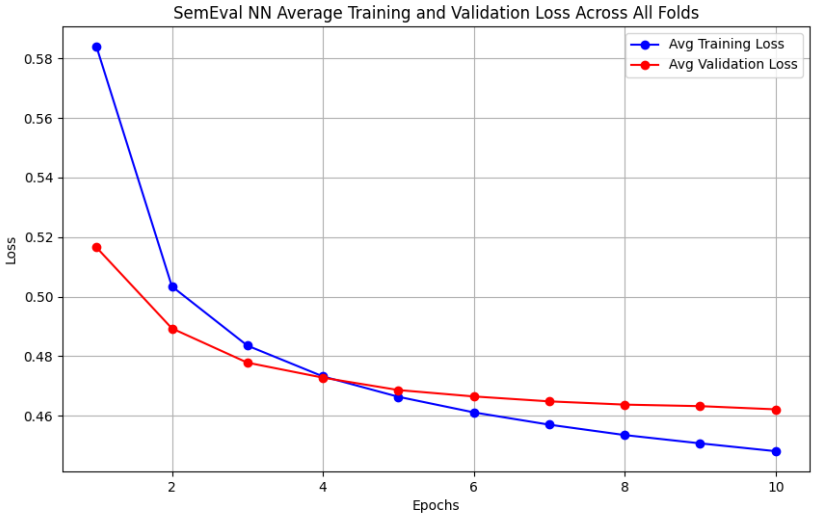
- Negative: Precision=77.35%, Recall=79.30%, F1=78.31%
- Positive: Precision=78.77%, Recall=76.77%, F1=77.76%

Training Curves:



(a) Accuracy

Figure 4: Training accuracy curve for Feed-Forward NN on SemEval dataset



(a) Loss

Figure 5: Training loss curve for Feed-Forward NN on SemEval dataset

4.2.2 LSTM Network Performance

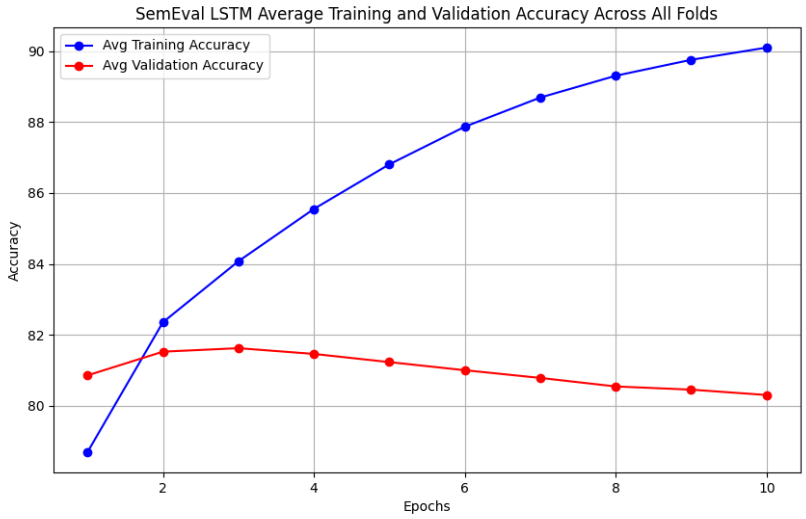
Metric	Value
Average Accuracy	81.64%
Average Precision	81.77%
Average Recall	81.44%
Average F1 Score	81.60%

Table 3: Performance metrics for LSTM on SemEval dataset

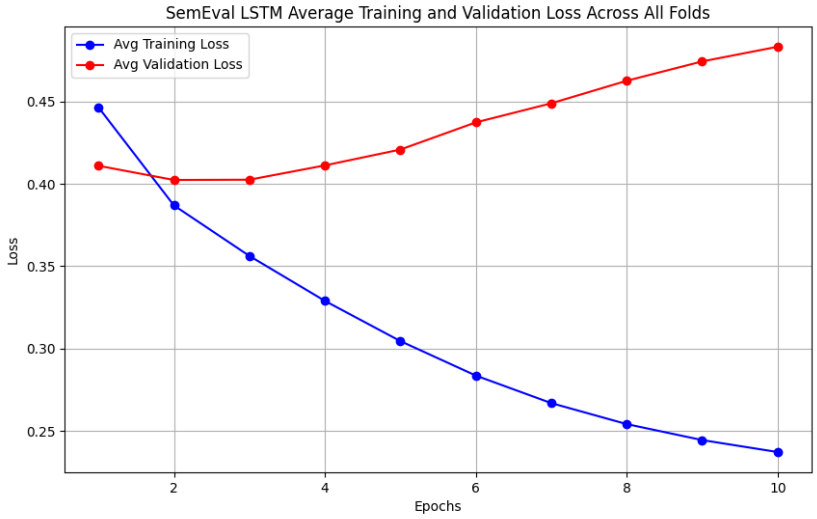
Average Per-Class Metrics:

- Negative: Precision=81.52%, Recall=81.83%, F1=81.67%
- Positive: Precision=81.77%, Recall=81.44%, F1=81.60%

Training Curves:



(a) Accuracy



(b) Loss

Figure 6: Average training and validation curves for LSTM on SemEval dataset

4.2.3 Comparative Analysis

For the SemEval dataset, the LSTM model again outperformed the NN model:

- **Accuracy**: LSTM achieved 3.6% higher accuracy (81.64% vs. 78.04%)
- **F1 Score**: LSTM achieved 3.84% higher F1 score (81.60% vs. 77.76%)

The consistent performance across all five folds indicates the robustness of both models, with the LSTM model showing more balanced metrics between precision and recall.

4.3 Twitter Dataset Results (Multi-class)

4.3.1 Feed-Forward Neural Network Performance

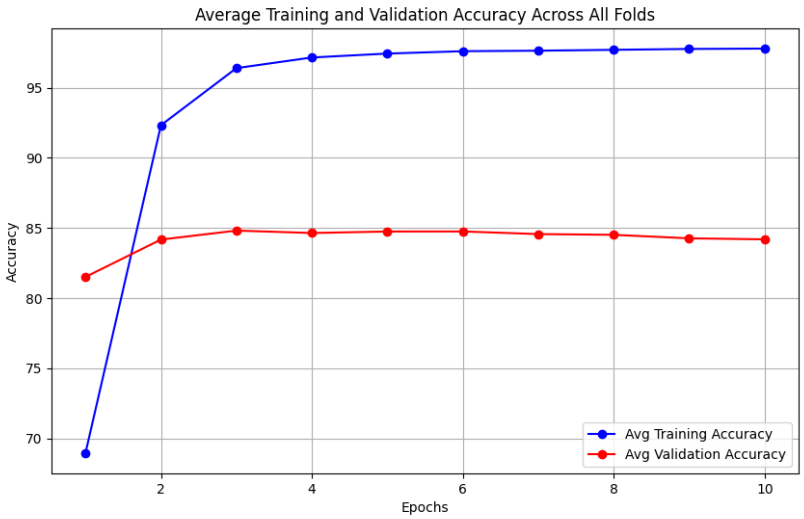
Metric	Value
Average Accuracy	84.87%
Average Precision	84.89%
Average Recall	84.87%
Average F1 Score	84.87%

Table 4: Performance metrics for Feed-Forward NN on Twitter dataset

Per-Class Metrics:

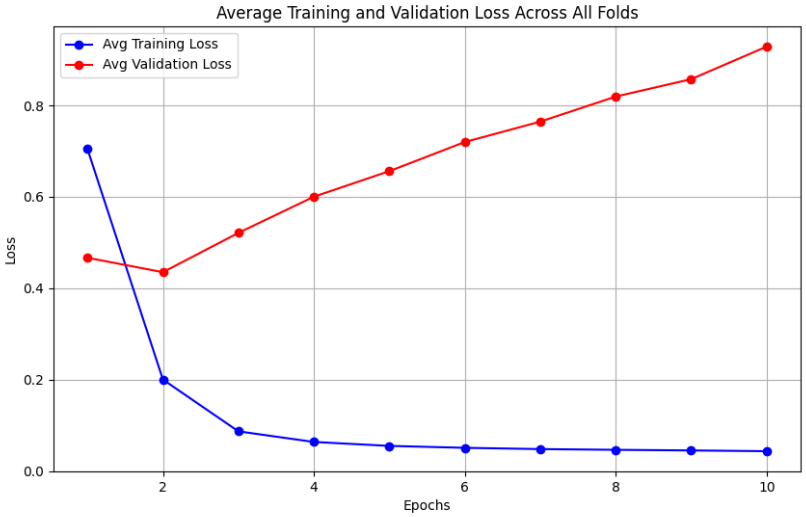
- Negative: Precision=86.51%, Recall=86.67%, F1=86.57%
- Neutral: Precision=83.36%, Recall=82.40%, F1=82.87%
- Positive: Precision=84.45%, Recall=85.10%, F1=84.76%

Training Curves:



(a) Accuracy

Figure 7: Training accuracy curve for Feed-Forward NN on the Twitter dataset



(a) Loss

Figure 8: Training loss curve for Feed-Forward NN on the Twitter dataset

4.3.2 LSTM Network Performance

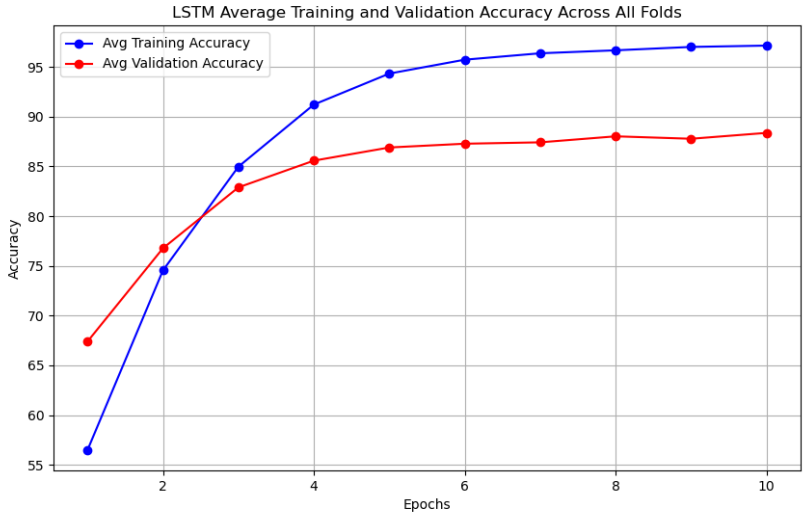
Metric	Value
Average Accuracy	88.46%
Average Precision	88.53%
Average Recall	88.46%
Average F1 Score	88.47%

Table 5: Performance metrics for LSTM on Twitter dataset

Per-Class Metrics:

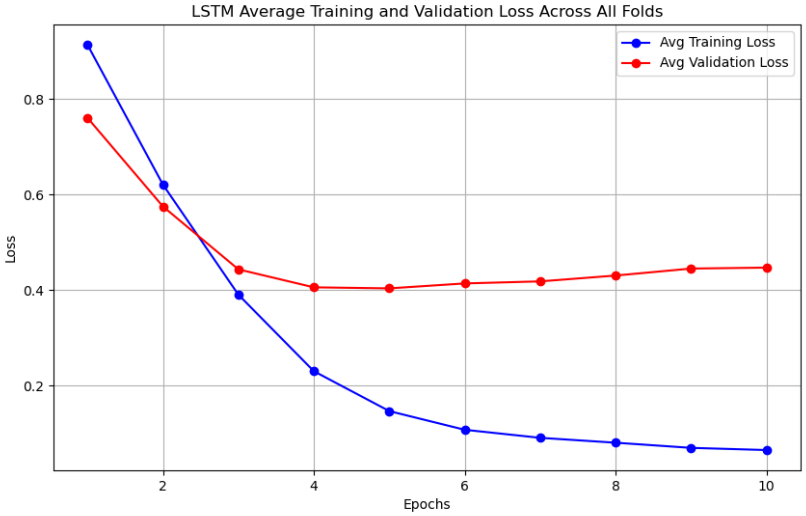
- Negative: Precision=91.11%, Recall=88.81%, F1=89.94%
- Neutral: Precision=85.73%, Recall=87.82%, F1=86.75%
- Positive: Precision=88.22%, Recall=88.63%, F1=88.40%

Training Curves:



(a) Accuracy

Figure 9: Training accuracy curve for LSTM on the Twitter dataset



(a) Loss

Figure 10: Training loss curve for LSTM on the Twitter dataset

4.3.3 Comparative Analysis

In the multi-class classification task, both models performed well, but the LSTM model still maintained a clear advantage:

- **Accuracy**: LSTM achieved 3.59% higher accuracy (88.46% vs. 84.87%)
- **F1 Score**: LSTM achieved 3.6% higher F1 score (88.47% vs. 84.87%)

It's notable that both models performed well on the "neutral" class, although with slightly lower metrics compared to the positive and negative classes.

4.4 Overall Comparative Analysis

Dataset	Model	Accuracy	Precision	Recall	F1 Score
IMDB	NN	74.61%	72.96%	78.19%	75.49%
IMDB	LSTM	82.39%	81.31%	84.11%	82.69%
SemEval	NN	78.04%	78.77%	76.77%	77.76%
SemEval	LSTM	81.64%	81.77%	81.44%	81.60%
Twitter	NN	84.87%	84.89%	84.87%	84.87%
Twitter	LSTM	88.46%	88.53%	88.46%	88.47%

Table 6: Overall performance comparison across all datasets and models

The LSTM model consistently outperformed the Feed-Forward Neural Network across all datasets and metrics. The performance improvement ranged from 3.6% to 8% in accuracy.

5 Discussion

5.1 Model Comparison

The LSTM network consistently outperformed the Feed-Forward Neural Network across all datasets. This superiority can be attributed to several factors:

1. **Sequential Information**: LSTM's ability to capture and utilize the sequential nature of language is crucial for sentiment analysis, where word order and context significantly impact meaning. For example, phrases like "not good" and "good" have opposite meanings despite sharing the word "good".

2. **Long-range Dependencies**: In longer texts like IMDB reviews, LSTM's capacity to maintain information over extended sequences provides a substantial advantage over the NN model, which loses sequential information.

3. **Context-awareness**: The LSTM model can better disambiguate words with different sentiments based on their context, which is particularly important for detecting sentiment shifts and nuanced expressions.

5.2 Dataset Characteristics

The performance gap between models varies across datasets, which can be explained by their characteristics:

1. **Text Length**: The gap is largest for IMDB (7.78% in accuracy) and smaller for SemEval and Twitter (around 3.6% in accuracy). This suggests that LSTM's advantages are more pronounced for longer texts.

2. **Linguistic Complexity**: Movie reviews often contain more complex sentiment expressions, including irony and mixed opinions, which benefit from LSTM's sequential processing.

3. **Class Balance**: Both models performed well on the balanced datasets, with metrics being relatively consistent across classes.

5.3 Error Analysis

Common error patterns observed:

1. **Negation Handling**: Both models, but particularly the NN model, struggled with negation (e.g., "not bad" being incorrectly classified as negative).

2. **Neutral Classification**: The neutral class in the Twitter dataset was the most challenging for both models, likely due to the more ambiguous nature of neutral sentiments.

3. **Overfitting**: The NN model showed stronger signs of overfitting than the LSTM model, as evidenced by the increasing gap between training and validation accuracy in later epochs.

6 Conclusion

This study implemented and compared Feed-Forward Neural Network and LSTM architectures for sentiment analysis across three different datasets. The results consistently demonstrated the superiority of LSTM networks for sentiment classification tasks, with performance improvements ranging from 3.6% to 8% in accuracy compared to Feed-Forward Neural Networks.

The advantages of LSTM were particularly pronounced for longer texts, highlighting the importance of capturing sequential information in natural language processing tasks. The implementation of efficient preprocessing techniques, including spaCy tokenization and embedding layers, contributed to the overall effectiveness of both models.

6.1 Future Work

Several directions for future work include:

1. **Pre-trained Embeddings**: Incorporating GloVe or Word2Vec embeddings might further improve performance.

2. **Bidirectional LSTM**: Implementing bidirectional LSTM could capture context from both directions.

3. **Attention Mechanisms**: Adding attention layers might help the models focus on the most sentiment-relevant parts of the text.

4. **Transformer-Based Models**: Comparing these traditional neural approaches with transformer-based models like BERT or RoBERTa.

5. **Ensemble Methods**: Combining predictions from multiple models could potentially improve overall performance.

This research demonstrates the effectiveness of neural approaches for sentiment analysis and highlights the particular advantages of recurrent architectures for capturing the sequential nature of language in sentiment classification tasks.

A Model Hyperparameters

Common Hyperparameters:

- Learning rate: $\alpha = 0.001$
- Number of epochs: 10
- Batch size: 64 (96 for multi-GPU training)
- Optimizer: Adam
- Dropout rate: $p = 0.3$

Feed-Forward Neural Network:

- Embedding dimension: $d = 100$
- Hidden layer 1 size: $h_1 = 256$
- Hidden layer 2 size: $h_2 = 128$
- Activation function: ReLU

LSTM Network:

- Embedding dimension: $d = 100$
- Hidden layer size: $h = 256$
- Number of layers: 1
- Batch-first: True

Loss Functions:

- Binary classification: $\mathcal{L}_{\text{binary}} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\sigma(\hat{y}_i)) + (1 - y_i)\log(1 - \sigma(\hat{y}_i))]$
- Multi-class classification: $\mathcal{L}_{\text{multi}} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c}\log(\hat{p}_{i,c})$