Report on Sentiment Analysis with Custom Neural Networks

Prokhar Navitski, 818431

April 5, 2025

Abstract

This report presents the project developed in the "Build Your Own Neural Network" course at University Potsdam. The aim was to perform sentiment analysis on the IMDB dataset using different neural network architectures (LSTM, CNN, and FeedForward networks) implemented in PyTorch. The document describes the project task, software architecture (including a UML diagram), experiment results under various settings, and reflections on the work process, citing both the IMDB dataset credentials and the documentation of the packages used.

1 Introduction

The project focused on Natural Language Processing (NLP) for sentiment analysis by training neural network models to classify movie reviews from the IMDB dataset as positive or negative. The implementation, done in PyTorch[1], comprises data loading, vocabulary building, tokenization, and training routines for three model types. The IMDB dataset (Maas et al., 2011)[2] forms the backbone of the experiments.

2 Task and Program Components

The training script is modularly organized into data loading utilities, dataset classes, model definitions, and training/evaluation functions. The overall logic begins with loading the IMDB reviews from a defined directory structure, tokenizing text, and building a vocabulary. Three distinct classifiers are implemented:

- **LSTMClassifier:** Uses bidirectional LSTM layers for sequential data.
- **CNNClassifier:** Applies convolutional layers with various kernel sizes to extract features.
- **FFClassifier:** A simple feedforward network that averages embeddings.

The training and evaluation procedures are encapsulated in dedicated functions (`train_model` and `evaluate_model`) that leverage standard optimization and loss functions (BCEWithLogitsLoss and Adam optimizer).

[1] PyTorch Documentation: `https://pytorch.org/docs/stable/index.html`

[2] Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., & Potts, C. (2011). Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*.

2.1 UML Diagram of the System Architecture

Figure 1 presents a UML diagram summarizing the main classes and functions used in the project.

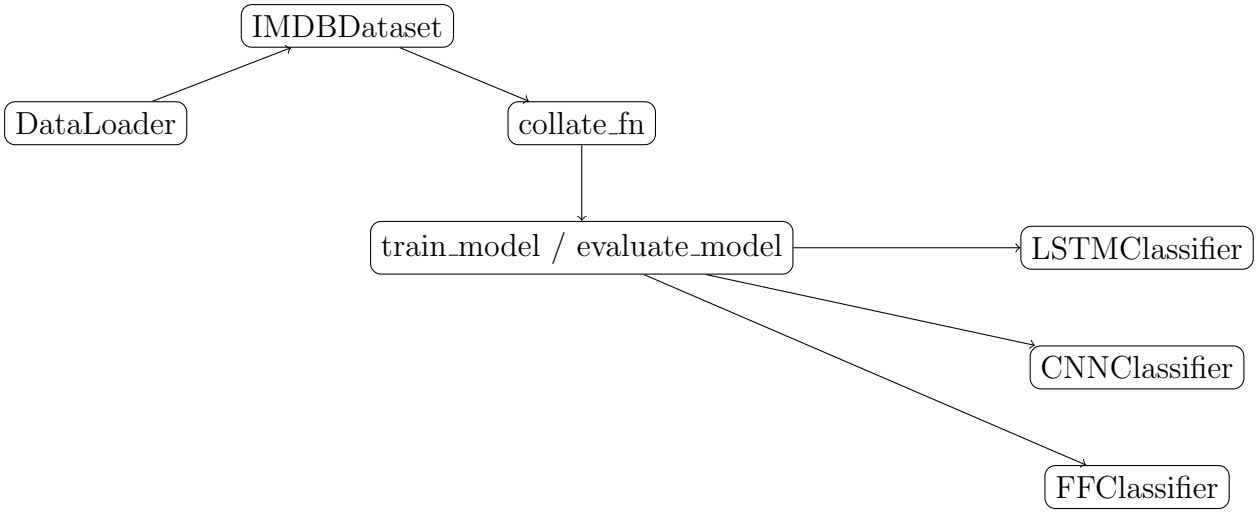

Figure 1: UML Diagram of the Project Architecture.

This diagram illustrates the flow from raw data to model training, highlighting the interaction between data preprocessing components and the three classifier models.

3 Process and Experimental Results

The experimental process was designed to evaluate the impact of different hyperparameter settings and data sizes on the performance of each model. For each experiment, the training logs were recorded and summarized. Before every change, the settings were reverted to a baseline to ensure comparability.

3.1 Experiment 1: Basic Settings with Full Dataset

The initial experiment used the original code settings and the full dataset. Table 1 summarizes the test performance metrics.

Model	Test Loss	Test Acc	Precision	Recall	F1 Score
LSTM	0.4699	0.8135	0.9162	0.6901	0.7872
CNN	0.3092	0.8687	0.8757	0.8594	0.8674
FeedForward	0.4065	0.8611	0.8844	0.8308	0.8567

Table 1: Performance with Basic Settings and Full Dataset.

Commentary: The CNN model outperforms the others with lower loss and higher accuracy, likely due to its ability to capture local features efficiently.

3.2 Experiment 2: Basic Settings with 5% of the Dataset

Due to long training times, the dataset was reduced to 5% of its original size. Table 2 shows the results.

Model	Test Loss	Test Acc	Precision	Recall	F1 Score
LSTM	0.9235	0.6112	0.6118	0.6520	0.6313
CNN	0.6015	0.7104	0.7277	0.6912	0.7090
FeedForward	0.6400	0.6592	0.6501	0.7194	0.6830

Table 2: Performance with 5% of the Dataset.

Commentary: With a reduced dataset, the models show degraded performance. However, CNN still performs relatively better, indicating its robustness even with less training data.

3.3 Experiment 3: Changed Embedding Dimension to 300

Increasing the embedding dimension to 300 was expected to capture more semantic information. The performance is summarized in Table 3.

Model	Test Loss	Test Acc	Precision	Recall	F1 Score
LSTM	1.0323	0.6224	0.6203	0.6708	0.6446
CNN	0.5072	0.7608	0.7858	0.7304	0.7571
FeedForward	0.6053	0.6952	0.7660	0.5799	0.6601

Table 3: Performance with Embedding Dimension set to 300.

Commentary: While increasing the embedding dimension should, in theory, improve feature representation, the improvement is marginal. In some cases, overparameterization may cause longer training times without significant gains.

3.4 Experiment 4: Changed Minimum Frequency to 5

By increasing the minimum word frequency for the vocabulary to 5, rare words were filtered out. Table 4 shows the performance.

Model	Test Loss	Test Acc	Precision	Recall	F1 Score
LSTM	0.8931	0.6048	0.6343	0.5329	0.5792
CNN	0.5950	0.7064	0.7978	0.5690	0.6642
FeedForward	0.6386	0.6664	0.6647	0.6991	0.6814

Table 4: Performance with Minimum Frequency set to 5.

Commentary: Removing infrequent words reduces noise but might also lose some useful nuance. This experiment reflects a trade-off between vocabulary size and model generalization.

3.5 Experiment 5: Changed Minimum Frequency to 1

Reducing the minimum frequency threshold to 1 allowed all words in the dataset to be considered. The results are in Table 5.

Model	Test Loss	Test Acc	Precision	Recall	F1 Score
LSTM	1.0521	0.6080	0.6034	0.6771	0.6381
CNN	0.6155	0.7008	0.6575	0.8636	0.7466
FeedForward	0.6318	0.6352	0.6112	0.7837	0.6868

Table 5: Performance with Minimum Frequency set to 1.

Commentary: Allowing every word increases vocabulary size and computational complexity. The results indicate a slight performance drop in LSTM while CNN maintains good recall.

3.6 Experiment 6: Added Weight Decay (1e-5)

Introducing weight decay as a regularization method is summarized in Table 6.

Model	Test Loss	Test Acc	Precision	Recall	F1 Score
LSTM	0.9027	0.6096	0.6036	0.6850	0.6417
CNN	0.5994	0.7056	0.7265	0.6787	0.7018
FeedForward	0.6580	0.6240	0.6120	0.7194	0.6614

Table 6: Performance with Weight Decay = 1e-5.

Commentary: Weight decay helps prevent overfitting by penalizing large weights. The improvements are subtle, suggesting that the original models were not overfitting severely.

3.7 Experiment 7: Set Dropout to 3

Increasing the dropout value (interpreted as a change in dropout rate or layer configuration) had the following effect (see Table 7):

Model	Test Loss	Test Acc	Precision	Recall	F1 Score
LSTM	0.9927	0.6088	0.5995	0.7038	0.6474
CNN	0.5805	0.7176	0.6976	0.7884	0.7403
FeedForward	0.6251	0.6720	0.6717	0.6991	0.6851

Table 7: Performance with Modified Dropout.

Commentary: Increasing dropout typically enhances regularization. In this case, CNN and FeedForward models showed modest improvements in test accuracy and F1 score, whereas LSTM results remained similar.

3.8 Justification of Experimental Changes

Each change was introduced to test the sensitivity of the training process to different hyperparameters. Modifying dataset size, embedding dimensions, vocabulary thresholds, regularization (weight decay), and dropout all influence the balance between model capacity and overfitting. This systematic exploration helps identify configurations that yield optimal generalization performance given time and computational constraints.

4 Reflection

The workflow started with extensive background research on neural network design and training, including studying open-source projects in sentiment analysis. PyTorch was chosen for its dynamic graph capabilities and robust community support[3]. The IMDB dataset, a classical benchmark for sentiment analysis, provided an ideal testbed due to its balanced and well-documented structure.

4.1 Difficulties and Successes

- **Easiest Part:** The idea generation was straightforward because the IMDB sentiment analysis project is a well-known pet project in data analysis.
- **Challenging Part:** Designing a training process that would yield the best results was challenging. Many approaches exist—each with pros and cons—and due to technical and time limitations, it was not feasible to try them all.
- **Challenges Resolved:** The biggest challenge was determining the optimal training configuration (e.g., choosing markers, setting learning rates, dropout, etc.). This was addressed by reviewing reports and articles on similar projects[4].
- **Unresolved Issues:** Some hyperparameter choices remain open to further tuning, and more exhaustive experiments could provide additional insights.

4.2 Future Directions

Next time, it would be valuable to:

- Experiment with alternative frameworks such as **TensorFlow** and **Keras** to compare performance and ease of use.
- Investigate more advanced architectures and techniques like attention mechanisms or transformer-based models.

Further, the project could be expanded by applying the trained model to other domains (e.g., online marketplace reviews) to assess its generalizability. An extension to multiclass sentiment analysis (using a scale of positivity and negativity) would require a more complex model design and longer training times, offering a rich area for future research.

[3]PyTorch Documentation: `https://pytorch.org/docs/stable/index.html`

[4]Hussain Wali, *PyTorch for Text Classification: A Step-by-Step Guide with Code Blocks*, `https://hussainwali.medium.com/pytorch-for-text-classification-a-step-by-step-guide-with-code-blocks-3bb25a40e936`

5 Conclusion

This project successfully implemented three neural network architectures for sentiment analysis using the IMDB dataset. The systematic experimental process demonstrated the impact of different hyperparameter settings on model performance. Through detailed analysis, adjustments in data volume, embedding dimensions, vocabulary filters, regularization methods, and dropout configurations were shown to affect accuracy, precision, recall, and F1 score. The reflections highlight both the straightforward and challenging aspects of the project and provide guidance for future work. Overall, the report underscores the importance of systematic experimentation in neural network training and the value of using robust frameworks such as PyTorch.

References

[1] PyTorch Documentation.
`https://pytorch.org/docs/stable/index.html`
Accessed: 2025-04-05.

[2] Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., & Potts, C. (2011).
Learning word vectors for sentiment analysis.
In Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies.

[3] Hussain Wali.
PyTorch for Text Classification: A Step-by-Step Guide with Code Blocks.
https://hussainwali.medium.com/pytorch-for-text-classification-a-step-by-step-guide-with-code-blocks-3bb25a40e936
Accessed: 2025-04-05.

[4] Analytics Vidhya. *Classification in Machine Learning.* `https://medium.com/analytics-vidhya/classification-in-machine-learning-ed30753d9461` (Accessed: 2025-04-05).