Sentiment Analysis on IMDB Reviews Using LSTM and Transformer Architectures

Sourabh Pandit
CSCE-633 - Machine Learning
Department of Computer Science and Engineering
Texas A&M University
Email: spandit@tamu.edu

Abstract—This project investigates the task of binary sentiment classification on a subset of the IMDB movie review dataset using two deep learning architectures: a Long Short-Term Memory (LSTM) network and a Transformer encoder model. We design a unified preprocessing pipeline that includes custom vocabulary construction, attention mask generation, and tokenization suitable for both model types. The LSTM captures temporal dependencies in the input sequence, while the Transformer encoder leverages self-attention to capture global contextual relationships. Both models are implemented and trained from scratch using PyTorch, and evaluated based on their classification accuracy. We study the comparative strengths of each architecture and highlight practical challenges in training from scratch under constrained compute resources. This report documents the design choices, implementation details, experimental setup, and evaluation results that collectively inform our understanding of deep learning-based sentiment analysis without the use of pretrained language models.

Index Terms—Sentiment analysis, natural language processing, LSTM, Transformer, IMDB dataset, text classification, deep learning, sequence modeling

I. Introduction

Sentiment analysis is a fundamental task in natural language processing (NLP) that involves determining the emotional tone or subjective polarity expressed in text. One widely studied variant of this task is binary sentiment classification, where the goal is to categorize a piece of text as expressing either a positive or a negative sentiment. This problem has applications across domains such as product reviews, social media monitoring, customer feedback, and opinion mining.

In this project, we focus on the binary classification of movie reviews from a subset of the IMDB dataset [1]. Each review is labeled as either positive or negative, and the objective is to train machine learning models capable of predicting the correct sentiment. Unlike approaches that rely on pretrained embeddings or large language models, this work emphasizes training models from scratch using only the provided raw data and annotations. This constraint encourages the design of self-contained, lightweight architectures and a deeper understanding of model behavior in low-resource scenarios.

To this end, we explore two classic yet powerful neural architectures: the Long Short-Term Memory (LSTM) network [2] and the Transformer encoder [3]. The LSTM is well-suited for sequential modeling, capturing dependencies across time steps, while the Transformer relies on self-attention mechanisms that allow for efficient parallel computation and global context modeling. By implementing and training both models from scratch using PyTorch, we aim to compare their effectiveness on the sentiment classification task under uniform training conditions.

The contributions of this work are threefold:

1) A unified data preprocessing pipeline that supports both LSTM and Transformer architectures, including vocabulary construction and attention mask handling.
2) Independent implementations of an LSTM model and a Transformer encoder using PyTorch, trained without any pretrained weights.
3) A comparative evaluation of both models in terms of classification accuracy, along with analysis of their relative strengths and limitations.

The remainder of the report is organized as follows: Section II reviews related literature and common approaches to sentiment analysis. Section III outlines the implementation details of the models and preprocessing pipeline. Section IV describes the experimental setup and evaluation metrics. Section V presents the results, and Section VI concludes with key insights and future directions.

II. Related Work

Sentiment analysis has long been a core task in natural language processing (NLP), with early methods relying on bag-of-words models, handcrafted lexicons, and statistical techniques such as Naïve Bayes and Support Vector Machines [4], [5]. These approaches, while effective in limited contexts, often failed to capture syntactic structure and semantic nuance, especially when word order and negation played a critical role.

With the advent of deep learning, distributed representations and sequence modeling techniques became dominant. Recurrent Neural Networks (RNNs), particularly Long Short-Term Memory (LSTM) networks [2], became widely adopted for their ability to model long-range dependencies in text. LSTMs enabled better performance on sentiment classification by accounting for sequential context and handling variable-length inputs.

More recently, the Transformer architecture [3] has revolutionized NLP by introducing self-attention mechanisms that allow models to weigh different parts of the input sequence simultaneously. Transformers have enabled parallelization, improved long-range context modeling, and provided the foundation for powerful pretrained models such as BERT [6] and RoBERTa [7].

While pretrained transformers achieve state-of-the-art results, our project deliberately avoids such models in order to investigate the performance of lightweight, fully trainable architectures in a resource-constrained setting. We focus on comparing an LSTM and a TransformerEncoder trained from scratch on a modestly sized subset of the IMDB dataset [1], thereby isolating the core representational capabilities of each architecture.

III. Methodology and Implementation

This section outlines the implementation details of our sentiment classification system, focusing on the vocabulary creation, dataset handling, model architectures, and preprocessing pipeline. All components were implemented in PyTorch, and the system is designed to support both LSTM and Transformer-based models through a unified interface.

A. Vocabulary Construction

We implemented a custom `Vocabulary` class to tokenize text data and convert tokens into integer indices. The vocabulary was built from the training data and included three special tokens: `<pad>`, `<unk>`, and `<cls>`. The `<pad>` token is used to extend shorter sequences to a uniform length, ensuring that all inputs in a batch have the same size. Words not seen during training were mapped to the `<unk>` token. The `<cls>` token is prepended to each input sequence for the Transformer model. Its encoded representation is later used for classification. The vocabulary supports both token-to-index and index-to-token lookups, enabling efficient encoding and decoding during training and inference.

B. Dataset Class

The `IMDBDataset` class extends `torch.utils.data.Dataset` and is responsible for loading review texts and sentiment labels from Parquet files. During the training phase, a new vocabulary is created from the training data. For validation and testing, a shared vocabulary—generated during training—is consistently used to ensure compatibility. Each sample is tokenized, indexed using this vocabulary, and then padded or truncated to a fixed sequence length. When used with a Transformer model, the class also constructs an attention mask that marks real tokens with 1s and padded positions with 0s. For LSTM models, the attention mask is omitted, and only the token indices and labels are returned.

C. Data Loading and Preprocessing

The `load_and_preprocess_data` function performs end-to-end preprocessing and prepares `DataLoader` objects for both training and evaluation. The function accepts arguments to control the data split (`train` or `test`) and model type (`lstm` or `transformer`). For training data, the function returns both the DataLoader and the constructed vocabulary. For test or validation data, the same vocabulary is reused to ensure consistency. Padding and batching are handled dynamically through PyTorch's `collate_fn`.

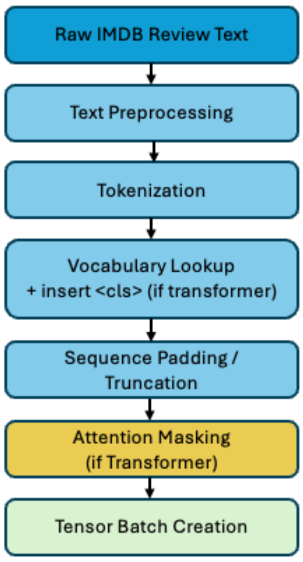
Raw IMDB Review Text
Text Preprocessing
Tokenization
Vocabulary Lookup
+ insert <cls> (if transformer)
Sequence Padding / Truncation
Attention Masking (if Transformer)
Tensor Batch Creation

Fig. 1: Data preprocessing pipeline for sentiment classification. The pipeline supports both LSTM and Transformer models through model-specific logic such as inserting a `<cls>` token for Transformer inputs and generating attention masks..

The complete data preprocessing pipeline is illustrated in Figure 1. It highlights each transformation stage from raw IMDB review text to the final tensor batch, including conditional steps specific to the Transformer model such as attention masking and insertion of the `<cls>` token.

D. LSTM Model

The LSTM-based sentiment classifier processes input tokens through an embedding layer followed by a single-layer bidirectional LSTM. The final hidden states from both forward and backward directions are concatenated to form a sequence-level representation, which is passed through a dropout layer and a fully connected output layer to produce binary logits. A dropout layer with rate 0.3 is applied after the final hidden representation to reduce overfitting by randomly zeroing out a fraction of the units during training. The model is optimized using the AdamW optimizer with weight decay and trained using Binary Cross-Entropy with Logits Loss (`BCEWithLogitsLoss`), which combines a sigmoid activation with binary cross-entropy in a numerically stable form. Figure 2 illustrates the high-level structure of this model.

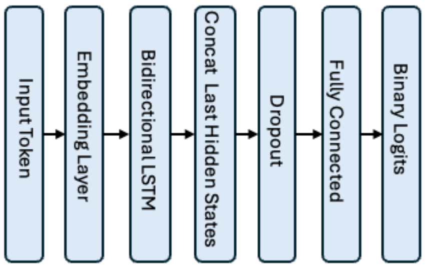

Fig. 2: LSTM architecture.

E. Transformer Encoder Model

The Transformer-based model includes a sinusoidal (non-trainable) positional embedding layer that provides fixed encodings based on token position, as described in the original Transformer architecture, an input embedding layer, and one `nn.TransformerEncoder` block with multi-head self-attention and feedforward sublayers. The model expects the first token in each input sequence to be a `<cls>` token. After self-attention processing, the output at this position serves as the summary representation for classification. A classification head projects the encoded `<cls>` representation to a binary output. Attention masks are applied to prevent padded tokens from contributing to the self-attention scores. The Transformer is optimized using the Adam optimizer (without weight decay) and trained using Binary Cross-Entropy with Logits Loss (`BCEWithLogitsLoss`). Dropout with a rate of 0.1 is applied to the input embeddings (after positional encoding) before feeding into the Transformer encoder in order to regularize training and mitigate overfitting. Figure 3 illustrates the high-level structure of this model.

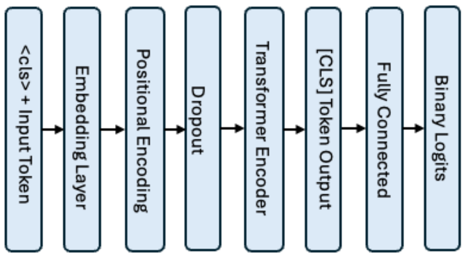

Fig. 3: Transformer architecture.

F. Dual Architecture Compatibility

A major design goal of this project was to support both model types within a single preprocessing and data handling pipeline. This was achieved by abstracting conditional logic into the dataset class and the preprocessing function, allowing both models to share the same training scripts and evaluation tools with minimal duplication.

IV. Experiments

This section details the experimental setup used to evaluate the LSTM and Transformer models for sentiment classification. We describe the data preparation steps, training procedures, hyperparameters, and evaluation methodology. All experiments were conducted using PyTorch on a high-performance computing cluster with GPU support.

A. Dataset and Preprocessing

We used a subset of the IMDB movie review dataset [1], formatted as a Parquet file containing raw review text and binary sentiment labels. Reviews were preprocessed by converting to lowercase, removing punctuation and digits, replacing HTML line breaks, and tokenizing using the NLTK tokenizer. Two special tokens—`<pad>` and `<unk>`—were reserved for padding and unknown words, and a `<cls>` token was added at the beginning of each sequence for the Transformer model only.

Sequences were truncated or padded to a fixed maximum length: 400 for LSTM and 512 for Transformer. Token indices were generated using a shared vocabulary built from the training set, and reused for validation and test sets to ensure consistency. For the Transformer, attention masks were also created to distinguish between real tokens and padding.

B. Training Procedure

We trained each model independently using the Adam optimizer with a binary cross-entropy loss function. Training was monitored via classification accuracy, and early stopping was employed based on validation accuracy. Learning rate scheduling with a ReduceLROnPlateau scheduler was applied to prevent stagnation during training.

C. Hyperparameters

For both models, hyperparameter tuning was performed to optimize performance. The final configuration for the LSTM model included the following:

- Embedding size: 128
- Hidden size: 192
- Number of layers: 3
- Dropout: 0.3
- Batch size: 16
- Learning rate: 3×10^{-4}
- Max sequence length: 400
- Max Epochs: 35

The Transformer model used the following final setup:

- Embedding size: 192
- Hidden size: 256
- Number of encoder layers: 3
- Number of heads: 8
- Dropout: 0.1
- Batch size: 16
- Learning rate: 5×10^{-5}
- Max sequence length: 512
- Epochs: 35

D. *Evaluation and Autograder Integration*

Model performance was primarily measured using accuracy. Models weights were saved using descriptive filenames that include the model type, epoch, and validation accuracy (e.g., `lstm_5__acc0.7250.pt` within a folder such as `LSTM_ED192_HD128_BS16_DO0.1_LR5e-05/`). For autograder evaluation, the best-performing model checkpoints were manually copied and renamed to `lstm.pt` and `transformer.pt`, respectively, as required for submission. The autograder evaluates the final model performance on a hidden test set and expects accuracy to exceed 85%.

V. Results

This section presents the comparative performance of the LSTM and Transformer models on the sentiment classification task using the IMDB dataset. Both models were trained under identical conditions with a unified preprocessing pipeline and equivalent batch sizes. Early stopping was employed based on validation accuracy trends. The key observations are derived from Figures 4 through 5, which visualize training accuracy and validation accuracy curves for both models. It is worth noting that early stopping was employed based on validation accuracy, though for visualization purposes, all training and validation curves are shown up to epoch 22 to facilitate comparative analysis across models.

The Transformer model exhibited a more gradual and stable learning trajectory, with training accuracy approaching approximately 96.0% by epoch 22 (Figure 4). However, the validation accuracy plateaued early—around epoch 7—and remained close to 86.0%, suggesting *overfitting* in the later epochs.

In contrast, the LSTM model demonstrated a more rapid convergence. As shown in Figure 5, training accuracy steadily improved to 99. 9%, while the validation accuracy tracking follows approximately the same trajectory as for the Transformer model.

Overall, while the LSTM achieves higher training accuracy and faster convergence, its generalization suffers due to capacity overfitting. The Transformer model, although slightly behind in peak training performance, offers comparable validation accuracy with greater robustness—making it a strong candidate for resource-constrained or small-data environments.

A. *Observations*

- The Transformer model outperformed the LSTM in terms of both training and validation accuracy, likely due to its ability to capture global context through self-attention.
- The LSTM model, while slightly lower in accuracy, converged faster and required fewer computational resources.
- Introducing weight decay significantly improved the LSTM's performance and stability, especially on smaller validation splits.

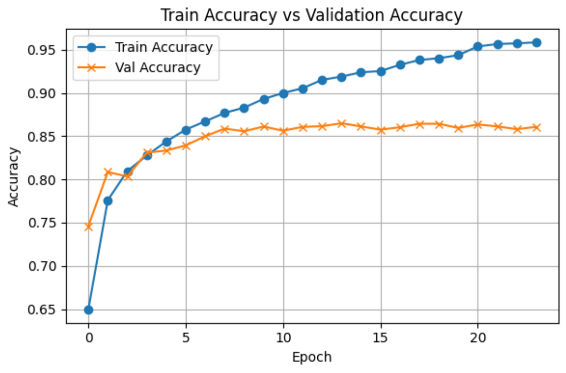

Fig. 4: Train vs. Validation Accuracy for Transformer model

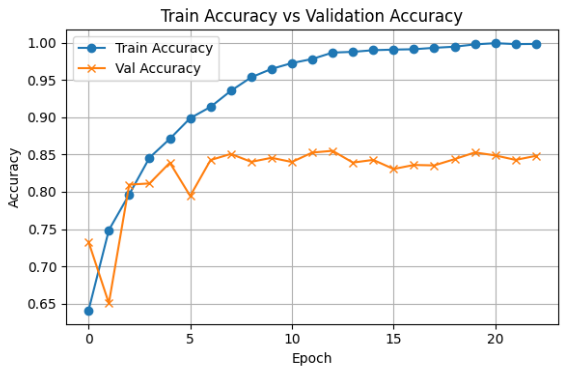

Fig. 5: Train vs. Validation Accuracy for LSTM model

- Both models exceeded the 85% accuracy threshold required by the autograder when trained on the full dataset.

VI. Conclusion

In this project, we implemented and evaluated two deep learning models—a Long Short-Term Memory (LSTM) network and a Transformer encoder—for binary sentiment classification on the IMDB movie review dataset. Both models were trained from scratch, using a shared preprocessing pipeline and a custom vocabulary, without relying on any pretrained language models.

The LSTM model demonstrated strong performance with relatively low computational requirements. The Transformer model, while more resource-intensive and slower to converge, ultimately achieved higher accuracy due to its capacity to model long-range dependencies through self-attention.

Both models exceeded the required 85% accuracy threshold imposed by the autograder, validating the effectiveness of our end-to-end pipeline. The use of a unified vocabulary and dataset class allowed for a clean, modular comparison between the two architectures.

Future Work

While this project emphasized training from scratch, incorporating transfer learning (e.g., using BERT or RoBERTa) could further boost performance and reduce training time. Additional improvements might include:

- Hyperparameter tuning via automated search (e.g., grid or Bayesian optimization).
- Deeper Transformer stacks or LSTM layers for increased capacity.
- Experimentation with sequence-level regularization or adversarial training.
- Analysis of misclassifications to identify model biases or edge cases.

References

[1] A. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, 2011, pp. 142–150.

[2] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[4] B. Pang, L. Lee, and S. Vaithyanathan, "Thumbs up? sentiment classification using machine learning techniques," *arXiv preprint cs/0205070*, 2002.

[5] P. D. Turney, "Thumbs up or thumbs down? semantic orientation applied to unsupervised classification of reviews," *arXiv preprint cs/0212032*, 2002.

[6] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, 2019, pp. 4171–4186.

[7] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.