Classifying IMDB Reviews with Transformers and LSTMs

Joseph August DeLeonardis
CSCE 633 - Machine Learning
Texas A&M University
College Station, Texas
delon741@tamu.edu

***Abstract*—This paper compares Long Short-Term Memory (LSTM) and Transformer architectures for binary sentiment classification on an IMDB movie review dataset. Both models were implemented from a skeleton file provided from the class resources and trained with 10 epochs on 25,000 reviews using Google Colab's GPU Infrastructure.**

I. Introduction

Classifying text by emotional tone is a fundamental natural language processing task. In the domain of movie reviews, automatically determining whether a review is positive or negative enables automated analysis of audience reception and provides valuable insights for recommendation systems. The IMDB movie review dataset contains approximately 25,000 training reviews that can be binary classified as positive or negative, making it a standard benchmark for evaluating text classification models. Two prominent deep learning approaches exist for this classification problem: Long Short-Term Memory (LSTM) networks and Transformer architectures. LSTMs process sequences one token at a time through recurrent connections, while Transformers process entire sequences in parallel using self-attention mechanisms. The prevalence of Transformer-based models in modern NLP applications suggests they should outperform traditional recurrent architectures, though their relative performance on moderate-length classification tasks remains an open question. This paper implements and compares bidirectional LSTM and Transformer encoder architectures for classifying IMDB movie reviews. The following sections detail the implementation of both architectures, hardware considerations for training, and a comparative analysis of their performance.

II. Methods

A. Dataset and Preprocessing

The IMDB movie review dataset contains 25,000 training samples with balanced positive and negative sentiment labels. The dataset was split into training (80Text preprocessing was performed using a multi-step pipeline. First, all text was converted to lowercase, and punctuation and numeric characters were removed. The NLTK word tokenizer was then applied to split the cleaned text into individual tokens. A vocabulary was constructed from the 10,000 most frequent tokens in the training data, with three special tokens reserved: ¡pad¿ (index 0) for sequence padding, ¡unk¿ (index 1) for out-of-vocabulary words, and ¡cls¿ (index 2) for the Transformer's classification token. All sequences were truncated or padded to a uniform length of 512 tokens to enable efficient batch processing. For the LSTM model, sequences began directly with vocabulary tokens, while Transformer sequences were provided the ¡cls¿ token at the beginning to provide a dedicated representation for classification.

B. LSTM Architecture

The LSTM (Long Short-Term Memory) model employed a bidirectional architecture to capture contextual information from both directions in the input sequence. The architecture consisted of the following components: The embedding layer converted token indices into dense 100-dimensional vector representations, providing a continuous representation of the vocabulary. These embeddings were processed by a 2-layer bidirectional LSTM with 256 hidden units per direction. The stacked architecture allowed the model to learn hierarchical representations, with each layer capturing increasingly abstract patterns in the text. The bidirectional processing enabled the model to access both past context (forward direction) and future context (backward direction) for each token, which is particularly valuable for sentiment analysis where the meaning often depends on surrounding words. The final hidden states from both directions of the top LSTM layer were concatenated, producing a 512-dimensional representation that combined forward and backward context. Dropout regularization (rate 0.5) was applied to prevent overfitting. This concatenated representation was passed through a fully connected layer to produce a single output value for binary sentiment classification, which was trained using Binary Cross-Entropy with Logits loss. Shown in figure 1, is block diagram of how this architecture works.

C. Transformer Architecture

The Transformer encoder architecture processed the entire input sequence simultaneously, unlike the LSTM which processes tokens one at a time. A block diagram of the architecture is shown in Figure 2. The model began with an embedding layer that converted token indices into 128-dimensional vectors. To help the model understand word order, sinusoidal positional encodings were added to these embeddings. The architecture consisted of 3 stacked encoder

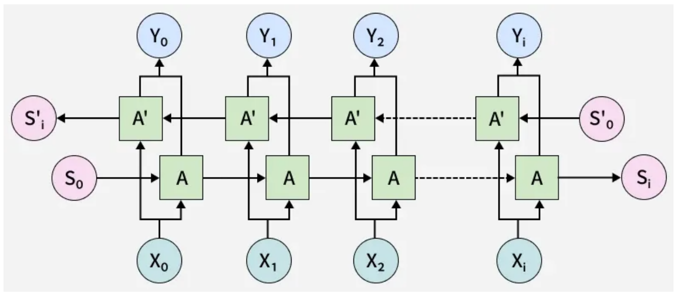

Fig. 1. Bidirectional LSTM architecture for sentiment classification. The forward LSTM (A) processes the sequence from start to end, while the backward LSTM (A') processes from end to start. The final output combines representations from both directions. Adapted from [4].

layers. Each layer used a multi-head attention mechanism with 8 heads. The attention mechanism allows each word to look at all other words in a review to determine which words are most important for understanding the review. The 8 heads allow the model to focus on different relationships between words simultaneously. After the attention layer, each encoder layer applied a feedforward network with 512 dimensions. This feedforward network further processes and refines the representations produced by the attention mechanism, applying nonlinear transformations to capture complex patterns in the data. Attention masking prevented the model from looking at padding tokens, ensuring it only processed actual review content. For classification, the model used the hidden state of the special ¡cls¿ token placed at the beginning of each sequence. This token accumulated information from the entire review through the attention layers. The ¡cls¿ representation passed through dropout (rate 0.1) and a final linear layer to predict sentiment. The model was trained using Binary Cross-Entropy with Logits loss.

D. Training Procedure

Both models were trained for 10 epochs using the Adam optimizer with a learning rate of 0.001 and Binary Cross-Entropy with Logits loss. Initial experiments were conducted with 5 epochs and a maximum sequence length of 256 tokens, but this configuration failed to achieve the target accuracy of 85The batch size was set to 32, allowing the models to process 32 reviews simultaneously during each training iteration, which improved computational efficiency on GPU hardware. Dropout regularization was applied differently for each architecture: the LSTM used a dropout rate of 0.5, while the Transformer used 0.1. This difference reflects the varying regularization needs of the architectures—LSTMs are more prone to overfitting on sequential data and benefit from stronger dropout, whereas Transformers incorporate inherent regularization through their attention mechanisms and require less aggressive dropout to maintain performance. Training dynamics exhibited some variability, with validation accuracy not increasing linearly with each epoch. To address this, a check was implemented in the code to save the best epoch as the model. This approach ensured that the final models represented peak performance rather than the last epoch's state.

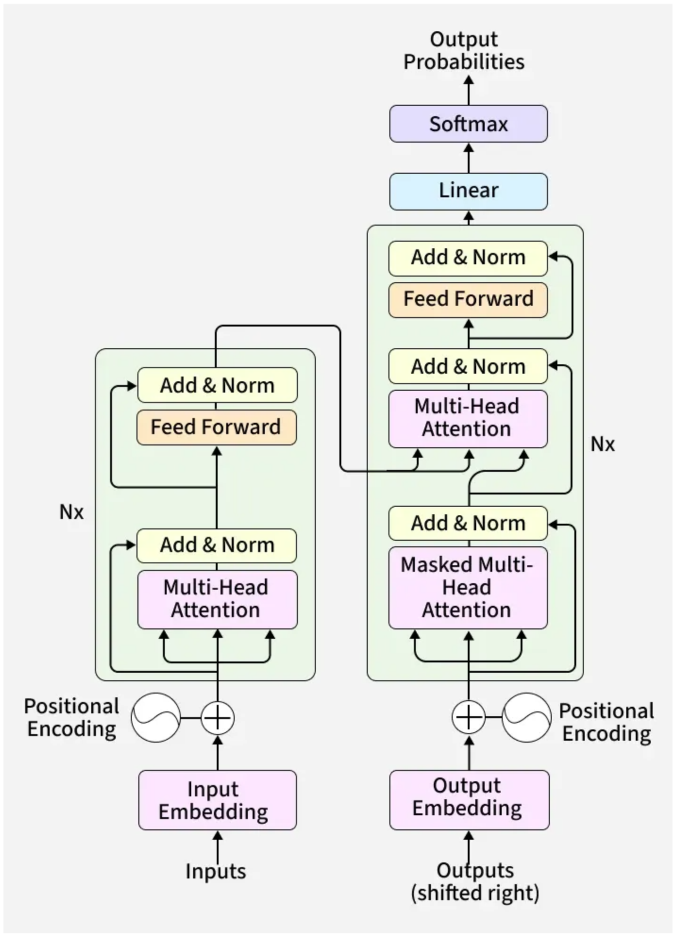

Fig. 2. Transformer encoder architecture showing multi-head self-attention and feed-forward layers with residual connections and layer normalization. Our implementation uses only the encoder portion for classification, extracting the [CLS] token representation for sentiment prediction. Adapted from [5].

TABLE I
MODEL HYPERPARAMETERS

Parameter	LSTM	Transformer
Vocab Size	10,000	10,000
Max Sequence Length	512	512
Embedding Dimension	100	128
Hidden Dimension	256	N/A
Number of Layers	2	3
Attention Heads	N/A	8
Feedforward Dimension	N/A	512
Dropout	0.5	0.1
Batch Size	32	32
Learning Rate	0.001	0.001
Optimizer	Adam	Adam

III. EXPERIMENTS AND RESULTS

Shown below in table II is a comparison of the results between both models.

TABLE II
MODEL PERFORMANCE

Model	Test Accuracy
LSTM (Bidirectional)	85.166%
Transformer Encoder	85.087%

A. Training Dynamics

Both models achieved the target performance after 10 epochs of training. The LSTM obtained a test accuracy of 85.166The hypothesis is that training could have continued for much higher results by increasing the maximum sequence length beyond 512 tokens and extending the number of training epochs. Validation accuracy had not fully plateaued by epoch 10, suggesting potential for further improvement. However, the current results demonstrate that both architectures achieve strong classification performance on the IMDB sentiment analysis task.

B. Model Comparison

Despite achieving nearly identical test accuracies, the two architectures offer distinct advantages for different deployment scenarios. The LSTM's sequential processing inherently captures temporal dependencies and requires fewer parameters for this task, making it suitable for resource-constrained environments. In contrast, the Transformer's parallel processing enables faster inference on modern GPU hardware and scales more effectively to longer sequences without the vanishing gradient issues that can affect recurrent architectures. The similar performance on this binary sentiment classification task suggests that the relatively short context window of 512 tokens and straightforward nature of the classification problem do not strongly favor either architectural approach. Both models successfully learned sentiment-bearing features from the IMDB reviews, indicating that architectural choice for this task can be guided by deployment constraints rather than expected accuracy differences.

IV. IMPLEMENTATION

A. Technical Stack

Given the local hardware constraints, alternative computational resources were evaluated for model training. The High Performance Computing (HPC) center at Texas A&M University was initially considered, but access was not granted in time for project completion. Consequently, the Google Colab platform was explored as a viable alternative. This migration was necessary because training on local CPU hardware was estimated to require 2-3 hours per epoch, which would total over 40 hours for both models to complete their full 10-epoch training cycles for both models. This would have been extremely inefficient for troubleshooting and iterative hyperparameter tuning, particularly if models failed to meet the target accuracy threshold.

TABLE III
GOOGLE COLAB HARDWARE SPECIFICATIONS

Component	Specification
GPU	NVIDIA T4
GPU Memory	16 GB GDDR6
System RAM	12-13 GB
CPU	Intel Xeon (2 cores)
Storage	78 GB (session)
Training Time/Epoch (LSTM)	2-3 minutes
Training Time/Epoch (Trans.)	2-3 minutes

TABLE IV
LOCAL CPU HARDWARE SPECIFICATIONS

Component	Specification
CPU	Intel Core i7-6600U
CPU Frequency	2.60 GHz (max 3.40 GHz)
Cores	2 (4 threads)
System RAM	16 GB
Architecture	x86_64 (6th gen Intel)
Cache L1d/L1i	64 KiB each
Cache L2	512 KiB
Cache L3	4 MiB
Training Time/Epoch (Est.)	3 hours

B. Development Challenges

The primary implementation challenge emerged during initial testing, where the model training code failed with cryptic autograder errors. Test 1.3 provided minimal diagnostic information, making it difficult to identify the root cause. The autograder alternated between reporting "tuple index out of range" and "too many values to unpack" errors, with no clear indication of the underlying issue.

The root cause was eventually identified as a dimensional mismatch between the model output and loss function. This breakthrough came only after extensive print statements were added throughout the code pipeline and intentionally incorrectly trained models were submitted with each iteration to trigger more detailed autograder feedback, particularly from Test 2.2 which provided more comprehensive error diagnostics. Prior to this approach, the error messages provided insufficient context to diagnose the problem. The additional diagnostic information proved critical to identifying and resolving the root cause, enabling the models to be properly trained.

The initial implementation used `output_dim=2` with `CrossEntropyLoss`, which expects integer class labels. However, the dataset returned float tensors of shape `(batch_size, 1)` appropriate for binary classification. The solution required changing to `output_dim=1` with `BCEWithLogitsLoss` to match the binary classification format. The dataset's `__getitem__` method was also modified to ensure consistent dimensionality by wrapping labels in brackets: `torch.tensor([label], dtype=torch.float)`. This seemingly minor change was critical—it ensured each label maintained its `(1,)` shape rather than becoming a scalar, preventing dimension mismatches during batch processing.

C. Use of Generative AI

Generative AI (Claude) was utilized during the debugging and implementation phases of this project.

Prompts Provided:

- "Why am I getting 'tuple index out of range' when training my model?"
- "How should I structure my dataset to return correct label dimensions for binary classification?"
- "What's the difference between BCEWithLogitsLoss and CrossEntropyLoss?"
- "The model is training extremely slowly, what could be the issue?"
- "How do I set up Google Colab to use GPU acceleration?"

AI Mistakes Identified:

- Initially suggested complex tensor reshaping workarounds when the solution was simpler: changing the loss function and output dimension
- Focused on code optimization for slow training instead of identifying the root cause: CPU vs GPU execution

Validation Methods:

- PyTorch documentation review for loss function requirements
- Extensive print statements to verify tensor shapes at each pipeline stage
- Training metric monitoring to ensure proper convergence

V. Conclusion

This work compared LSTM and Transformer architectures for sentiment analysis on the IMDB movie review dataset. Both models achieved approximately 85Data preprocessing and cleaning proved critical throughout this project. Correctly structuring the data pipeline to feed information into the models was essential for achieving the desired results. The dimensional mismatch issues encountered during development underscored the importance of understanding how data flows through neural network architectures. Given the prevalence of Transformers in modern NLP applications, the Transformer was expected to outperform the LSTM. However, the results showed nearly identical performance between the architectures. This similarity likely reflects the fact that neither model was trained to its maximum potential accuracy. With optimal hyperparameters, extended training, and longer sequence lengths, the Transformer would likely surpass the LSTM, as its parallel attention mechanism scales more effectively than sequential recurrent processing. This project also demonstrated the critical importance of GPU hardware for machine learning training. CPU-based training proved unsustainable for models of this complexity, with estimated training times exceeding 40 hours. GPU acceleration reduced this to approximately 30 minutes total, enabling practical iterative development and hyperparameter tuning. Modern deep learning workflows require appropriate computational resources to be feasible.

Future work could explore extend sequence lengths beyond 512 tokens to capture more context, and more epochs so the models could train past the point of convergence so that it can be observed what is the true maximum for accuracy that can be achieved.

References

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

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

[3] A. L. 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.

[4] "Bidirectional LSTM in NLP," GeeksforGeeks. [Online]. Available: https://www.geeksforgeeks.org/nlp/bidirectional-lstm-in-nlp/

[5] "Transformers in Machine Learning," GeeksforGeeks. [Online]. Available: https://www.geeksforgeeks.org/machine-learning/getting-started-with-transformers/

[6] G. James, D. Witten, T. Hastie, R. Tibshirani, and J. Taylor, *An Introduction to Statistical Learning with Applications in Python*. Cham, Switzerland: Springer Nature, 2023.