Deep Learning for Natural Language Processing

Samyak Jain (2022MT11658)

April 7, 2025

1 Introduction

This report presents the implementation and analysis of two deep learning architectures for a multi-category classification problem: (a) a hybrid CNN-LSTM model and (b) a Transformer-based text encoding model. The study examines the impact of using pre-trained word embeddings and custom-trained embeddings, self-attention (C-LSTM model), positional embeddings and number of encoder blocks (Transformer) on classification performance.

2 Dataset Description

The dataset consists of labeled text samples for a multi-category classification task. The dataset is divided into training and test sets, with the following statistics:

- Number of training samples: `1491`
- Number of test samples: `735`
- Number of categories: `5 (Sports, Tech, Entertainment, Politics and Business)`

3 Pre-Training Pipeline

3.1 Tokenization and Vocabulary creation

The dataset is loaded from a CSV file using `pd.read_csv`, and the text is extracted from the "Text" column with all entries converted to lowercase. A `RegexpTokenizer` with the pattern `"\d|\w+"` is used to tokenize the text, capturing both digits and words. English stopwords are removed using NLTK's stopwords list. Tokens from all entries are aggregated, and their frequencies are computed via `Counter`; the most frequent tokens up to the specified vocabulary size are retained. Finally, a mapping from words to unique indices (with a reserved index for `<'UNK'>`) is created, along with a reverse mapping. This process efficiently converts raw text into a numerical format for further analysis.

3.2 Word Embeddings

Pre-trained word embeddings from 300 M words Google news dataset(A) are compared against use of custom-trained word embeddings using CBOW model(B). Pre-trained word embeddings (A) have dimension 300 and CBOW model trained embeddings have dimension *20*.
Hyperparameters for CBOW model:

- Vocabulary Size: 10000
- Context Window Size: 2

3.3 Dataset and Embedding Conversion Methodology

The dataset is loaded from a CSV file using `pd.read_csv`, where each record contains a news article and its corresponding category. For each article, the text is processed by the `clean_text` function to tokenize and clean the input. The resulting tokens are converted to embeddings using a provided embedding dictionary, with a maximum sequence length of `max_len=100` applied by truncating longer texts and padding shorter ones with zero vectors. Categories are mapped to integer labels based on a predefined dictionary. Finally, the `__getitem__` method returns the text embeddings and the associated label as PyTorch tensors, ensuring the data is formatted appropriately for model training.

4 Model Description

4.1 Hybrid CNN-LSTM Model

Two Models with the following architecture and hyperparameters were compared:

4.1.1 C-LSTM A

- Architecture:
  - A convolutional layer to extract local features from text sequences.
  - A Long Short-Term Memory (LSTM) for sequential processing.
  - A self-attention mechanism (optional) to enhance feature representation.
  - A fully connected layer with softmax activation for classification.
- Hyperparameters:
  - Number of CNN filters: `50`
  - Kernel size: `10`
  - LSTM hidden units: `64`

– Dropout rate: `0.2`

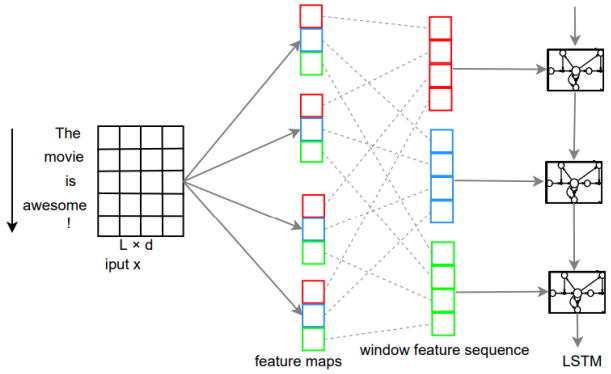

Figure 1: Architecture of C-LSTM A

4.1.2 C-LSTM B

- Architecture:
  - Three parallel convolutional layers with different kernel sizes to extract local features from the input text.
  - A Long Short-Term Memory (LSTM) layer to capture sequential dependencies from the input embeddings.
  - Feature concatenation: The mean-pooled features from the CNN layers are concatenated with the final hidden state of the LSTM.
  - Two fully connected layers that maps the concatenated features to the desired number of classes for classification.
- Hyperparameters:
  - Kernel Sizes= `[3,5,7]`
  - LSTM hidden dimension: `64`
  - Number of CNN filters per layer: `50`

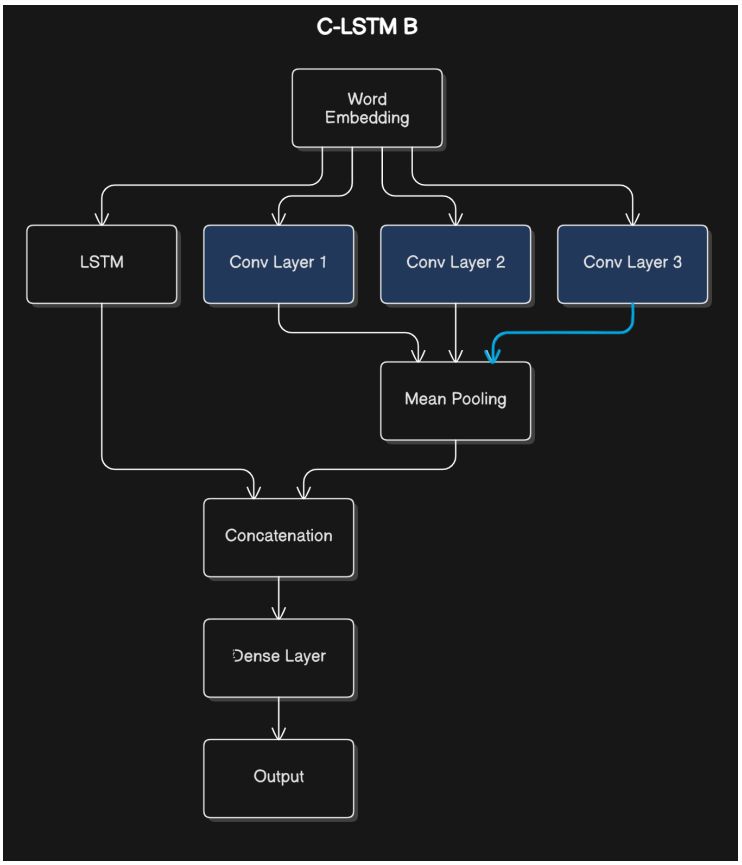

Figure 2: Architecture of C-LSTM B

4.2 Transformer-Based Model

- Architecture:
  - Optionally applies positional encoding to the input embeddings using fixed sine and cosine functions computed for a maximum sequence length (here, 100).
  - A stack of Transformer encoder blocks, each composed of:
    - Multi-head self-attention with dropout.
    - A two-layer feedforward network with ReLU activation.
    - Residual connections and layer normalization applied after both the self-attention and feedforward sub-layers.
  - Global average pooling is performed over the sequence (averaging across tokens) after the Transformer layers.

  - A fully connected layer that maps the pooled features to the desired number of classes for classification.

- Hyperparameters:
  - Number of attention heads: `5`
  - Feed-forward network hidden dimension: `2*embedding_dim`
  - Dropout rate: `0.2`
  - Maximum sequence length for positional encoding: `=max_len(100)`

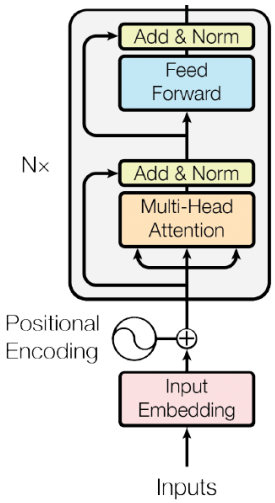

Figure 3: Architecture of Transformer Encoder

5 Justification of Model Design and Hyperparameter Choices

5.1 Hybrid CNN-LSTM Models

5.1.1 Justification for Architecture Choices

CNN Layers: Convolutional layers are effective at capturing local patterns in text, such as n-gram features, which are crucial for understanding the semantics of natural language. In **C-LSTM A**, a single convolutional layer is used to extract these local features before feeding them into the LSTM, ensuring that short-term dependencies are effectively modeled. **C-LSTM B** enhances this approach by using *three parallel convolutional layers* with different kernel sizes (`[3,5,7]`), capturing a richer and more diverse set of n-gram features (e.g., trigrams, 5-grams, 7-grams) from the input text.

LSTM Layer: LSTMs are chosen for their ability to capture long-range dependencies and temporal structure in sequential data. In both models, the LSTM follows the CNN layer(s), allowing it to model contextual relationships among features extracted by the CNN, which is a well-established architecture for text classification tasks.

Feature Concatenation (C-LSTM B): Combining mean-pooled CNN features with the final LSTM hidden state enables the model to simultaneously leverage both local and global sequence representations, enhancing classification performance, especially on tasks where both syntactic and semantic features are important.

Self-Attention (Optional in both C-LSTM A and C-LSTM B): The inclusion of an optional self-attention mechanism allows the model to weigh the importance of different parts of the sequence when making predictions, improving interpretability and robustness to irrelevant tokens.

5.1.2 Justification for Hyperparameter Choices

- **Number of CNN filters (50):** A moderate number of filters strikes a balance between representational capacity and computational efficiency. It is sufficient to capture diverse local patterns without overfitting.
- **Kernel Sizes ([10] in A, [3,5,7] in B):** A large kernel size like 10 in C-LSTM A captures broader context in a single filter, while multiple smaller kernels in C-LSTM B allow multi-scale pattern recognition.
- **LSTM Hidden Dimension (64):** A hidden size of 64 provides enough capacity to learn sequential dependencies without significantly increasing the number of parameters, which helps avoid overfitting on smaller datasets.
- **Dropout (0.2):** Dropout is used for regularization. A rate of 0.2 is chosen to prevent overfitting while retaining a large portion of the learned representations during training.

5.2 Transformer-Based Model

5.2.1 Justification for Architecture Choices

Positional Encoding: Transformers lack an inherent sense of sequence, so positional encodings are necessary to inject order information. Fixed sine-cosine encoding is computationally efficient and works well in practice for capturing relative token positions.

Transformer Encoder Layers: The encoder stack is well-suited for text classification, as it allows tokens to attend to each other regardless of distance, effectively capturing both local and global dependencies. Residual connections and layer normalization ensure stable training and deep representation learning.

Global Average Pooling: Averaging across all token embeddings ensures a smooth representation of the entire sequence and improves robustness.

Final Fully Connected Layer: This layer projects the pooled features to class logits. Using a simple fully connected layer reduces model complexity while being effective when preceded by powerful contextual encoding.

5.2.2 Justification for Hyperparameter Choices

- **Number of Attention Heads (5):** Using multiple attention heads enables the model to jointly attend to information from different representation subspaces. Five heads provide sufficient diversity while remaining computationally efficient.
- **Feedforward Network Size (2×embedding_dim):** Doubling the hidden size is a common practice in Transformer architectures to provide increased learning capacity in the feedforward component without excessive overhead.
- **Dropout (0.2):** A moderate dropout rate helps prevent overfitting in deeper Transformer models, especially when training on relatively smaller text datasets.
- **Maximum Sequence Length (100):** This value is chosen based on empirical observation of input length distribution. It captures the vast majority of sentence lengths without unnecessary padding, ensuring training efficiency.

6 Experimental Results

6.1 Hybrid C-LSTM model

The performance of the models are first compared on the basis of use of pretrained embeddings and custom-trained embeddings using CBOW

Embeddings	C-LSTM model A	C-LSTM model B	Transformer (4 Encoder Blocks)
Custom Trained	Accuracy:92.52% F1 score:0.9251 Precision:0.9256 Recall:0.9252	Accuracy:93.20% F1 score:0.9320 Precision:0.9321 Recall:0.9320	Accuracy:92.79% F1 score:0.9277 Precision:0.9280 Recall:0.9279
Pre Trained	Accuracy:93.88% F1 score:0.9390 Precision:0.9396 Recall:0.9388	Accuracy:95.78 % F1 score:0.9579 Precision:0.9582 Recall:0.9578	Accuracy:96.05% F1 score:0.9606 Precision:0.9608 Recall:0.9605

Table 1: Comparision of different models with and without pre-trained embeddings

Since the results are better for pre-trained embeddings, the pre-trained embeddings are used for the rest of the analysis.

Self Attention Used	C-LSTM model A	C-LSTM model B
Not Used	Accuracy:93.88% F1 score:0.9390 Precision:0.9396 Recall:0.9388	Accuracy:95.78 % F1 score:0.9579 Precision:0.9582 Recall:0.9578
Used	Accuracy:93.33% F1 score:0.9338 Precision:0.9359 Recall:0.9333	Accuracy:93.61 % F1 score:0.9359 Precision:0.9417 Recall:0.9361

Table 2: Comparision of CNN-LSTM models with and without Self Attention

6.2 Transformer-based Encoder

The performance of the model is compared with various encoder blocks and the use of positional embeddings.

Model	Without Positional Encoding	With Positional Encoding
Transformer Model (2 Encoder Blocks)	Accuracy: 94.97% F1 score: 0.9499 Precision: 0.9514 Recall: 0.9497	Accuracy: 91.02% F1 score: 0.9102 Precision: 0.9126 Recall: 0.9102
Transformer Model (4 Encoder Blocks)	Accuracy: 95.78% F1 score: 0.9578 Precision: 0.9588 Recall: 0.9578	Accuracy: 92.52% F1 score: 0.9252 Precision: 0.9298 Recall: 0.9252
Transformer Model (6 Encoder Blocks)	Accuracy: 96.05% F1 score: 0.9605 Precision: 0.9609 Recall: 0.9605	Accuracy: 95.10% F1 score: 0.9510 Precision: 0.9513 Recall: 0.9510

Table 3: Comparision of Transformer models with and without positional encoding

7 Discussion

From the experimental results, several important observations can be made regarding the effectiveness of different architectures and model configurations:

7.1 Impact of Pre-trained vs Custom-trained Embeddings

As seen in Table 1, the use of pre-trained embeddings consistently improved model performance across all architectures. For instance, C-LSTM B improved from an F1 score of 0.9320 (custom-trained) to 0.9579 (pre-trained), while the Transformer model with 4 encoder blocks improved from 0.9277 to 0.9606. This indicates that leveraging semantic knowledge captured in large pre-trained embedding models significantly enhances classification performance compared to embeddings trained from scratch using CBOW.

7.2 Effect of Self-Attention in Hybrid C-LSTM Models

Incorporating self-attention in the hybrid C-LSTM models led to a decline in performance. For example, C-LSTM B saw a reduction in F1 score from 0.9579 to 0.9359 when self-attention was added. This suggests that the added complexity of the self-attention layer might not be necessary or optimal for this dataset and task when already using LSTM and convolutional components.

7.3 Transformer Model and Positional Encoding

Interestingly, the use of positional encoding resulted in worse performance across all Transformer configurations. For instance, the Transformer with 6 encoder blocks achieved an F1 score of 0.9605 without positional encoding, but this dropped to 0.9510 with positional encoding. This is counterintuitive as positional encoding is typically crucial in Transformers to introduce sequential information. One possible explanation could be that the input sequence structure was already well captured by the model through other means, or that the fixed sinusoidal encoding did not align well with the task-specific patterns.

7.4 Effect of Encoder Depth in Transformers

Increasing the number of encoder blocks in the Transformer consistently improved performance. The model with 6 encoder blocks achieved the highest performance (F1 score: 0.9605), followed by 4 and 2 blocks respectively. This trend indicates that deeper Transformer models were able to capture more complex hierarchical representations of the input text, contributing to better classification accuracy.

8 Conclusion

In this study, we compared various deep learning architectures for text classification, including hybrid CNN-LSTM models and Transformer-based encoders. The key findings are as follows:

- Pre-trained embeddings significantly boost model performance and should be preferred over training embeddings from scratch.
- The C-LSTM B model outperformed C-LSTM A, highlighting the effectiveness of multi-kernel CNN layers and feature concatenation strategies.
- Self-attention did not provide performance benefits in the C-LSTM models and may introduce unnecessary complexity for certain tasks.
- Transformer-based models outperformed CNN-LSTM hybrids when sufficient encoder depth was used, with the 6-layer Transformer model achieving the best overall results.
- Surprisingly, positional encoding slightly hurt performance in this setting, suggesting that its effectiveness may vary depending on data characteristics and model depth.

These findings provide insights into architectural choices for text classification and emphasize the importance of empirical evaluation when designing deep learning models.