50.040 Natural Language Processing - Final Project Report -

Group 21 - Sentence Senseis

Atul Parida (1006184)
Ansh Oswal (1006265)
Elvern Neylman Tanny (1006203)

December 13, 2024

Abstract

This project investigates sentiment classification on the Stanford IMDb movie review data set using a fine-tuned BERT-uncased model for binary sentiment analysis. Building upon the baseline RNN and CNN approaches, we developed a Transformer-based model to improve sentiment prediction performance. Our fine-tuned BERT model leverages pre-trained contextual embeddings to capture nuanced language representations, effectively distinguishing between positive and negative movie reviews. Through careful hyperparameter tuning and model optimization, we achieved competitive performance in the test set, demonstrating the effectiveness of transfer learning in natural language processing tasks. The research highlights the potential of pre-trained language models in sentiment analysis and provides insights into model selection and optimization strategies.

1 Introduction

1.1 Repository Location

This repository contains the necessary code and dataset files required for sentiment analysis, including the initial CNN and RNN architectures built according to the project's original questions, and the Hybrid CNN-RNN architecture with attention that was developed for the final design. The dataset repository for this project is hosted on GitHub and can be accessed via the following link:

Repository Link https://github.com/T2LIPthedeveloper/50.040-NLP-Final-Project

1.2 Project Overview

This project focuses on sentiment analysis of movie reviews, utilizing a novel approach that combines convolutional neural networks (CNNs) and recurrent neural networks (RNNs) with an attention mechanism. The model is designed to classify movie reviews as either positive or negative based on the text content, leveraging pre-trained GloVe embeddings and a hybrid architecture to capture both local patterns and long-term dependencies within the text. The dataset used for training and evaluation is the IMDB movie reviews dataset, which consists of 50,000 labeled reviews.

1.3 Purpose and Goals

The main objective of this project is to develop an effective sentiment analysis model that can accurately predict the sentiment of movie reviews based on textual data. The combination of CNN and RNN models with attention mechanisms allows the system to focus on the most relevant parts of the review while considering both local and sequential dependencies. This hybrid model approach aims to improve the classification accuracy by addressing various challenges in natural language processing (NLP), such as handling long-term dependencies and noisy data.

2 Model Description

2.1 Architecture Overview

The sentiment analysis model employs a **Hybrid CNN-RNN** architecture integrated with an **Attention Mechanism**. This combination allows the model to capture both local patterns through CNNs and long-term dependencies through RNNs, while the attention layer enables the model to focus on the most relevant parts of the input sequence for classification.

2.2 Components Detail

2.2.1 Pre-trained Embeddings

- **GloVe Embeddings:** Utilizes pre-trained GloVe embeddings (200-dimensional) to initialize the embedding layers, providing the model with rich semantic representations.
- **Handling OOV Words:** Words not present in the GloVe vocabulary are initialized with random vectors drawn from a normal distribution.

2.2.2 Embedding Layers

- **Trainable Embedding Layer:** A standard trainable embedding layer initialized with pre-trained GloVe vectors.
- **Constant Embedding Layer:** A non-trainable embedding layer also initialized with pre-trained GloVe vectors.

Rationale: Combining both trainable and constant embeddings offers a balance between preserving pre-trained semantic knowledge and allowing the model to adapt embeddings for task-specific optimization. This dual-embedding strategy enhances the model's capacity to generalize while maintaining the integrity of foundational linguistic representations.

2.2.3 Dropout Layer

- Applied after concatenating the embedding layers to prevent overfitting by randomly zeroing some of the elements during training.

Rationale: Applying dropout at this stage mitigates over-reliance on specific embedding features, encouraging the model to develop more generalized and resilient representations that enhance its ability to generalize to unseen data.

2.2.4 Convolutional Layers

- Multiple 1D convolutional layers with varying kernel sizes (3, 5, 7) to capture local n-gram features.
- Each convolutional layer uses padding to maintain the input sequence length.

Rationale: Employing multiple convolutional layers with varying kernel sizes allows the model to capture diverse n-gram features, enabling it to recognize different granularities of sentiment indicators within the text. This multi-scale feature extraction is crucial for identifying both short phrases (e.g., "not good") and longer expressions (e.g., "absolutely fantastic performance").

2.2.5 Activation Function

- **ReLU Activation:** Introduces non-linearity after each convolutional layer to enable the model to learn complex patterns.

Rationale: ReLU is chosen for its simplicity and effectiveness in mitigating the vanishing gradient problem, facilitating faster and more efficient training of deep networks by allowing gradients to flow through the network without significant attenuation.

2.2.6 Bidirectional LSTM

- Processes the concatenated convolutional outputs to capture sequential dependencies in both forward and backward directions.
- Comprises two layers with a hidden size of 150 units each.

Rationale: A bidirectional LSTM is instrumental in understanding the context surrounding each token by considering both preceding and succeeding words in the sequence. This dual perspective enhances the model's ability to disambiguate sentiment indicators that may depend on contextual clues from either side of a token (e.g., "not good" vs. "good enough").

2.2.7 Attention Mechanism

- Computes attention weights for each timestep in the LSTM output.
- Generates a context vector as a weighted sum of LSTM outputs, emphasizing the most relevant parts of the sequence.

Rationale: The attention mechanism enhances the model's interpretability and performance by enabling it to prioritize certain parts of the input sequence that are more indicative of the sentiment. This dynamic weighting allows the model to effectively ignore irrelevant or noisy parts of the text, focusing computational resources on the most informative segments.

2.2.8 Fully Connected Layer

- Maps the context vector to the final output classes (positive and negative sentiments).

Rationale: The fully connected layer serves as the final decision-making component of the model, translating the rich, context-aware representations into actionable predictions for sentiment classification.

3 Model Components and Mathematical Representation

3.1 Input Representation

Let the input be a sequence of tokens (words) represented as:

$$\mathbf{x} = (x_1, x_2, \ldots, x_T)$$

where T is the sequence length.

3.2 Embedding Layers

The model uses two embedding layers:

Standard Embedding: Maps each token to a dense vector:

$$\mathbf{E} \in \mathbb{R}^{V \times d}$$

where V is the vocabulary size and d is the embedding dimension. The embedding for token x_i is:

$$\mathbf{e}(x_i) = \mathbf{E}[x_i]$$

Constant Embedding: Another embedding layer with the same dimensions:

$$\mathbf{E}_c \in \mathbb{R}^{V \times d}$$

The constant embedding for token x_i is:

$$\mathbf{e}_c(x_i) = \mathbf{E}_c[x_i]$$

Concatenated Embedding: The combined embedding is:

$$\mathbf{h}_i = \begin{bmatrix} \mathbf{e}(x_i) \\ \mathbf{e}_c(x_i) \end{bmatrix} \in \mathbb{R}^{2d}$$

Dropout: Dropout is applied to the embeddings to prevent overfitting:

$$\tilde{\mathbf{h}}_i = \text{Dropout}(\mathbf{h}_i)$$

Embedding Matrix for CNN: The embeddings are arranged for convolution as:

$$\tilde{\mathbf{H}} = [\tilde{\mathbf{h}}_1, \tilde{\mathbf{h}}_2, \ldots, \tilde{\mathbf{h}}_T]^\top \in \mathbb{R}^{2d \times T}$$

3.3 Convolutional Layers

Multiple 1D convolutional layers capture various n-gram features:

Convolution Parameters: For each convolutional layer m:

$$k_m = \text{kernel size}, \quad c_m = \text{number of output channels}$$

Padding p_m ensures output has the same temporal dimension as input:

$$p_m = \left\lfloor \frac{k_m - 1}{2} \right\rfloor$$

Convolution Operation: The convolution operation followed by ReLU activation is:

$$\mathbf{y}_m = \mathrm{ReLU}(\mathbf{W}_m * \tilde{\mathbf{H}} + \mathbf{b}_m) \in \mathbb{R}^{c_m \times T}$$

Concatenation of Convolution Outputs: Combine outputs from all convolutional layers along the channel dimension:

$$\mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_M] \in \mathbb{R}^{\left(\sum_{m=1}^{M} c_m\right) \times T}$$

Transpose for RNN: Prepare the tensor for the RNN by transposing:

$$\mathbf{Y}' = \mathbf{Y}^\top \in \mathbb{R}^{T \times \sum_{m=1}^{M} c_m}$$

3.4 Recurrent Neural Network (LSTM)

A bidirectional LSTM processes the concatenated convolutional features:

LSTM Output: At each time step t:

$$\mathbf{h}_t = \begin{bmatrix} \overrightarrow{\mathbf{h}}_t \\ \overleftarrow{\mathbf{h}}_t \end{bmatrix} \in \mathbb{R}^{2H}$$

where H is the hidden size of each LSTM direction. The sequence of outputs is:

$$\mathbf{H}_{\mathrm{LSTM}} = (\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_T) \in \mathbb{R}^{T \times 2H}$$

3.5 Attention Mechanism

The attention mechanism assigns weights to each LSTM output:

Attention Scores: Compute unnormalized scores for each time step:

$$a_t = \mathbf{w}^\top \mathbf{h}_t + b_a \in \mathbb{R}$$

where $\mathbf{w} \in \mathbb{R}^{2H}$ and $b_a \in \mathbb{R}$ are parameters.

Attention Weights: Normalize scores with softmax:

$$\alpha_t = \frac{\exp(a_t)}{\sum_{k=1}^{T} \exp(a_k)} \in \mathbb{R}$$

Context Vector: Compute the weighted sum of LSTM outputs:

$$\mathbf{c} = \sum_{t=1}^{T} \alpha_t \mathbf{h}_t \in \mathbb{R}^{2H}$$

3.6 Decoder (Output Layer)

The context vector is passed through a linear layer to produce the final output:

$$\mathbf{o} = \mathbf{W}_d \cdot \mathrm{Dropout}(\mathbf{c}) + \mathbf{b}_d \in \mathbb{R}^{C}$$

where C is the number of output classes.

3.7 XLNet Testing and Decision

Initially, the XLNet model was considered for its ability to capture bidirectional contexts and dependencies. The XLNet representation:

$$P(\mathbf{x}) = \prod_{i=1}^{T} P(x_i|\mathbf{x}_{\setminus i})$$

was evaluated using permutation-based attention mechanisms. However, preliminary experiments revealed excessive computational overhead and marginal improvements in performance. Thus, we opted for the CNN-LSTM-Attention architecture, which provided a more efficient trade-off between complexity and accuracy.

Components The model consists of two embedding layers, convolutional layers for feature extraction, a bidirectional LSTM for capturing sequential dependencies, an attention mechanism for weighing the importance of time steps, and a decoder layer for the final output. These components work together to classify sentiment from text data efficiently.

Iterations The model was developed iteratively, starting with a basic CNN-RNN architecture, followed by the addition of attention mechanisms to improve performance. Various hyperparameters such as the number of convolutional layers, LSTM hidden size, and dropout rate were tuned to optimize the model's accuracy. The final version integrates both CNN for feature extraction and LSTM for sequence modeling, with an attention mechanism to focus on the most relevant parts of the sequence.

4 Training Settings

4.1 Dataset

- **Dataset Used:** IMDB Movie Reviews Dataset
- **Data Source:** `https://ai.stanford.edu/~amaas/data/sentiment/`
- **Description:** The dataset consists of 50,000 movie reviews labeled as positive or negative, with an even distribution of classes.

4.2 Preprocessing

1. **Tokenization:**
    - Utilizes spaCy's tokenizer to split text into tokens.
    - Converts all tokens to lowercase.
    - Filters out punctuation and whitespace.
2. **Vocabulary Building:**
    - Constructs a vocabulary from the training data with a minimum frequency threshold of 5.
    - Includes a reserved `<pad>` token for padding sequences.

3. **Sequence Preparation:**
    - Truncates or pads each tokenized review to a fixed length of 500 tokens.
    - Ensures uniform input sizes for efficient batch processing.
4. **Embedding Initialization:**
    - Loads pre-trained GloVe embeddings (200-dimensional).
    - Initializes an embedding matrix where known words are assigned GloVe vectors, and OOV words receive random vectors.

4.3 Hyperparameters

- **Batch Size:** 64
- **Embedding Dimension:** 200
- **Convolutional Kernel Sizes:** [3, 5, 7]
- **Number of Convolutional Channels:** [100, 100, 100] for each kernel size
- **LSTM Hidden Size:** 150
- **Number of LSTM Layers:** 2
- **Dropout Rate:** 0.5
- **Optimizer:** Adam
    - **Learning Rate:** 0.0005
    - **Weight Decay:** 1e-5
- **Loss Function:** Cross-Entropy Loss
- **Number of Epochs:** 10

5 Model Performance

The models were evaluated using various metrics like accuracy, precision, recall, and F1 score.

5.1 Bi-directional RNN Performance on IMDB Test Data

The Bi-directional RNN (BiRNN) achieved the following performance metrics on the IMDB test dataset. See Table 1 for detailed results.

Metric	**Value**
Accuracy	0.8320
Precision	0.9116
Recall	0.7352
F1 Score	0.8140

Table 1: BiRNN Performance on IMDB Test Data

5.2 Text-CNN Performance on IMDB Test Data

The Text-CNN model achieved the following performance metrics on the IMDB test dataset. See Table 2 for detailed results.

Metric	**Value**
Accuracy	0.8476
Precision	0.8132
Recall	0.9026
F1 Score	0.8556

Table 2: Text-CNN Performance on IMDB Test Data

5.3 Design Project Performance on IMDB Test Data

The hybrid CNN-RNN architecture with attention, developed for the design project, achieved the following performance metrics on the IMDB test dataset. See Table 3 for detailed results.

Metric	**Value**
Accuracy	0.9035
Precision	0.9021
Recall	0.9053
F1 Score	0.9037

Table 3: Hybrid CNN-RNN Performance on IMDB Test Data

5.4 Design Project Performance on Evaluation Data

The hybrid CNN-RNN model was further evaluated on an additional dataset, achieving the following performance metrics. See Table 4 for detailed results.

Metric	**Value**
Accuracy	0.9472
Precision	0.9464
Recall	0.9480
F1 Score	0.9472

Table 4: Hybrid CNN-RNN Performance on Evaluation Data

6 Code

```python
class HybridCNNRNN(nn.Module):
    def __init__(self, vocab_size, embed_size, kernel_sizes, num_channels,
                 lstm_hidden_size, num_lstm_layers, dropout=0.5, **kwargs):
        super(HybridCNNRNN, self).__init__(**kwargs)
        self.embedding = nn.Embedding(vocab_size, embed_size)
        self.constant_embedding = nn.Embedding(vocab_size, embed_size)
        self.dropout = nn.Dropout(dropout)
        self.convs = nn.ModuleList()
        for c, k in zip(num_channels, kernel_sizes):
            padding = (k - 1) // 2
            self.convs.append(
                nn.Conv1d(
                    in_channels=2 * embed_size,
                    out_channels=c,
                    kernel_size=k,
                    padding=padding
                )
            )
        self.relu = nn.ReLU()
        self.lstm = nn.LSTM(
            input_size=sum(num_channels),
            hidden_size=lstm_hidden_size,
            num_layers=num_lstm_layers,
            bidirectional=True,
            batch_first=True
        )
        self.attention = nn.Linear(2 * lstm_hidden_size, 1)
        self.decoder = nn.Linear(2 * lstm_hidden_size, 2)

    def forward(self, inputs):
        embeddings = torch.cat((self.embedding(inputs), self.constant_embedding(inputs)), dim=2)
        embeddings = self.dropout(embeddings)
        embeddings = embeddings.permute(0, 2, 1)
        conv_outputs = []
        for conv in self.convs:
            conv_out = self.relu(conv(embeddings))
            conv_outputs.append(conv_out)
        conv_outputs = torch.cat(conv_outputs, dim=1)
        conv_outputs = conv_outputs.permute(0, 2, 1)
        lstm_out, _ = self.lstm(conv_outputs)
        attention_weights = torch.softmax(
            self.attention(lstm_out).squeeze(-1), dim=1)
        context_vector = torch.sum(lstm_out * attention_weights.unsqueeze(-1), dim=1)
        outputs = self.decoder(self.dropout(context_vector))
        return outputs
```

Listing 1: Fine-Tuned Model Code

7 Work Division Amongst Team Members

Atul Parida: Report writing, RNN implementation and XLNet experimentation.
Ansh Oswal: Data preprocessing, hyperparameter tuning and report development.
Elvern Neylman Tanny: Design project development, BERT experimentation and Hybrid model development.

Acknowledgments

We thank the Stanford AI Group for the IMDb dataset, which was fundamental to our sentiment analysis project. We also appreciate the Singapore University of Technology and Design (SUTD), particularly the 50.040 Natural Language Processing course, for their academic support and guidance.

Supplementary Files

- **Supplementary Dataset:** IMDb movie reviews dataset used for model training and evaluation.
- **Model Script:** Python script implementing the final model (`sentencesenseis_script.py`).
- **Exploratory Notebook:** Jupyter notebook showcasing testing, experimentation, and results (`finalproject_sentencesenseis.ipynb`).
- **Dependencies:** List of project dependencies included in `requirements.txt`.
- **Results:** Condensed results packaged into a single ZIP file (`results.zip`), containing multiple `results.csv` files for each model.