HW1 experiment Report

Name:李崇楷 Student ID: 313834006

Code: https://github.com/kailee0422/RNN-Transformer/tree/main/HW1

Introduction

With the rapid advancement of AI, distinguishing between AI-generated and human-written text has become an important challenge in various fields. This assignment aims to develop an LSTM-based deep learning model to classify text from the AI_Human.csv dataset as either AI-generated or human-written. The task involves preprocessing the text data, including tokenization and embedding, followed by training an LSTM model to optimize classification accuracy. The dataset will be split into training, validation, and test sets, and techniques such as hyperparameter tuning, dropout regularization, and model combination may be explored to enhance performance. Finally, all program files (.ipynb) and the report (.pdf) will be uploaded to GitHub, with the repository URL serving as the assignment submission.

Method

In this study, several word embedding methods were evaluated to investigate their effectiveness in capturing semantic representations for text classification tasks. The explored embedding techniques include BERT, AutoTokenizer with AutoEmbedding, basic custom embedding, Word2Vec, and FastText.

1. BERT(Tokenizer and Embedding): BERT utilizes a transformer-based architecture and performs contextualized word embedding. I specifically employed the BERT base model (cased), which is case-sensitive, distinguishing between words such as "english" and "English." The dimensionality of the embeddings provided by this model is 768. BERT embeddings leverage positional encoding and attention mechanisms, enabling the capture of contextual nuances. Precomputing embeddings prior to training enhanced computational efficiency and improved model convergence during the training phase.
2. AutoTokenizer and AutoEmbedding: We also adopted the AutoTokenizer combined with AutoEmbedding from Hugging Face transformers, specifically the BERT base uncased variant. This method automatically selects the appropriate tokenizer and embedding based on the provided pre-trained model. Unlike the cased model, this uncased version normalizes all input text to lowercase, thereby ignoring distinctions such as "english" versus "English." The embedding dimension remains consistent at 768. Precomputing embeddings prior to training enhanced computational efficiency and improved model convergence during the training phase.
3. Basic: A basic embedding was implemented by constructing a custom vocabulary file (voc.txt). Padding tokens were designated an embedding index of 0, and unknown tokens were assigned an index of 1. Due to computational constraints, input sequences were limited to a maximum length of 512 tokens. The embedding matrix was initialized and trained using standard neural network embedding layers (nn.Embedding). While simple to implement and computationally lightweight, this approach does not inherently capture semantic relationships as effectively as pre-trained embeddings.
4. Word2Vec: Word2Vec, which Google developed, generates static word embeddings by predicting context words from given target words (skip-gram) or predicting target words from a set of context words (Continuous Bag-of-Words, CBOW). The embeddings derived from Word2Vec effectively capture semantic

relationships based on local context, though they lack contextual flexibility found in transformer-based embeddings. The dimension chosen for Word2Vec embeddings in this study was 300.

5. FastText: FastText embeddings, proposed by Facebook, extend the Word2Vec methodology by modeling words as compositions of character n-grams, thereby capturing morphological information and improving embeddings for rare or unseen words. Similar to Word2Vec, FastText embeddings were also fixed at a dimension of 300 in this research.

And three primary neural network architectures were employed to evaluate the effectiveness of the different embedding techniques:

1. LSTM: The first architecture consisted of Long Short-Term Memory (LSTM) layers designed to effectively model temporal sequences and capture long-term dependencies within textual data. The outputs from the LSTM layer were fed into a fully connected (FC) layer to produce classification outcomes.
2. CNN-LSTM: A hybrid model combining Convolutional Neural Networks (CNNs) and LSTMs was also tested. CNN layers first extracted local and hierarchical features from embeddings, after which the processed features were passed to the LSTM layers for capturing temporal dependencies. Finally, predictions were obtained through an FC layer. The CNN-LSTM architecture effectively harnesses both local feature extraction and sequential modeling capabilities.
3. Bidirectional LSTM (BiLSTM): The BiLSTM model utilized two separate LSTM networks operating in forward and backward directions. By integrating information from both past and future contexts, the BiLSTM enhances contextual understanding. This property typically performs better than unidirectional LSTM models but with increased computational complexity.

Each embedding technique was evaluated using all three model architectures to systematically assess their strengths and limitations, providing insights into optimal configurations for text classification tasks.

Result

	Bert	AutoTokenizer AutoEmbedding	Basic	Word2Vec	FastText
LSTM	99.82%	98.48%	**99.96%**	99.78%	99.74%
CNN-LSTM	99.73%	98.58%	**99.93%**	99.35%	99.82%
BiLSTM	99.81%	98.68%	**99.95%**	99.85%	99.59%

Note: All the experiments are trained by 20 epochs with Adam as an optimizer and 1e-4 as lr. Bold text indicates the word embedding method with the highest accuracy in the model.

Table 1. Test Accuracy for Different Tokenizer and Word Embedding Combinations

The above table show the accuracy of the test data based on different tokenizers and embeddings. The charts below illustrate how the training and testing loss evolved over epochs during the training process, as well as how the test accuracy changed across epochs. These graphs represent the top three models with the highest accuracy. If you're interested in the details of the implementation, feel free to check out my full code.

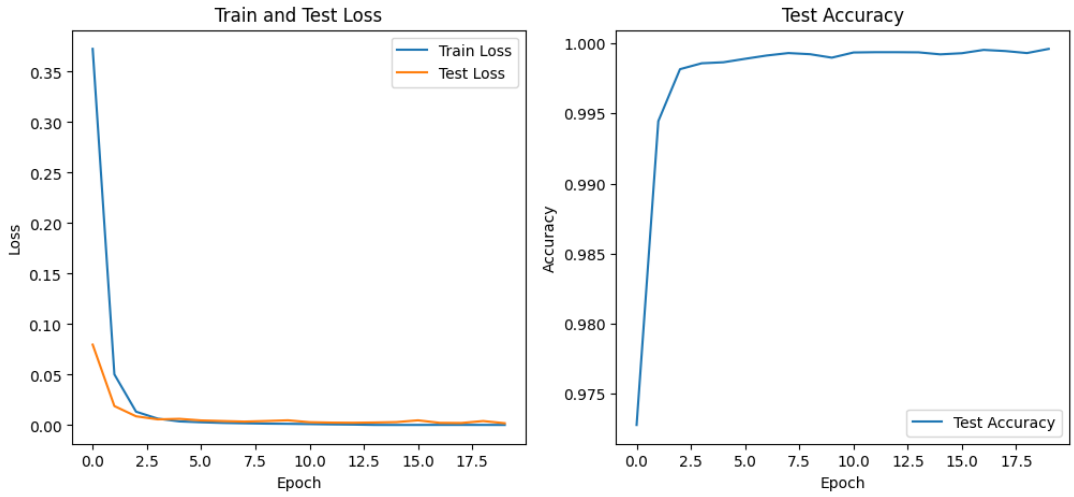

Figure 1. Training and testing loss(left), and test accuracy(right) over epochs for the basic method using an LSTM model.

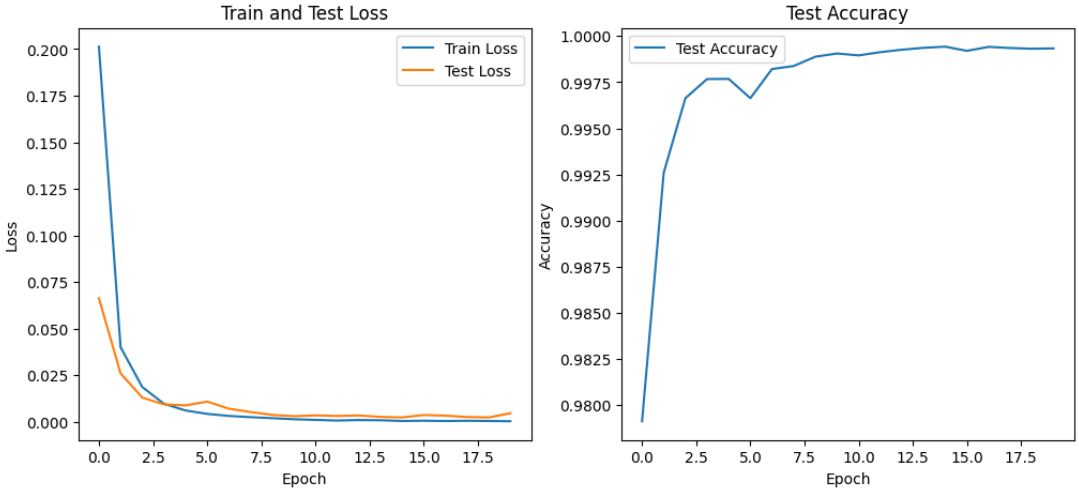

Figure 2 . Training and testing loss(left), and test accuracy(right) over epochs for the basic method using an CNN-LSTM model.

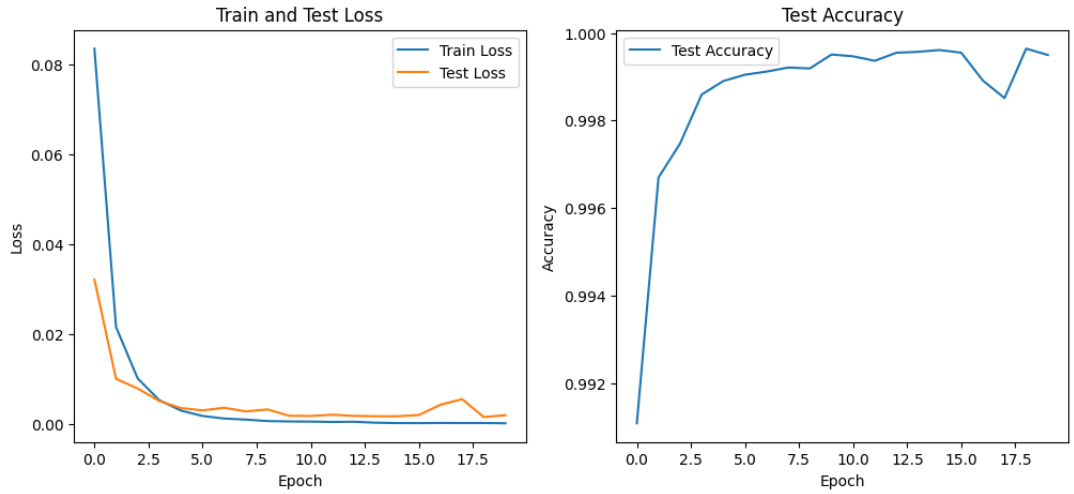

Figure 3 . Training and testing loss(left), and test accuracy(right) over epochs for the basic method using an BiLSTM model.

Conclusion

Experimental results showed that all word embedding methods achieved over 98% test accuracy in text classification, with the custom Basic embedding unexpectedly outperforming others—reaching 99.96% accuracy with the LSTM model. Despite being simpler than pre-trained embeddings like BERT, Word2Vec, and FastText, Basic embedding may have benefited from dataset characteristics, preprocessing steps, or better compatibility with specific model architectures. This suggests that in some cases, simpler embeddings can lead to better performance. Future work should investigate this further through cross-validation, hyperparameter tuning, and testing on other datasets.