Sentiment Analysis Design Challenge

AUTHORS	STUDENT ID
Aditya Kumar	1006300
Chin Wei Ming	1006264
Caitlin Daphne Tan Chiang	1006537

Singapore University of Technology and Design

50.040: Natural Language Processing

Prof. Lu Wei

December 13, 2024

Overview

For the design challenge portion of the project, the team decided to explore a range of models, to find out which will produce the highest in the evaluation metrics.

Before the experimentations, we had to set the records of the original base metrics of both the TextCNN and BiRNN models included in the final_project notebook. With the test set provided to the cohort, we utilized this to check our models:

Model	**Loss**	**Accuracy**	**Precision**	**Recall**	**F1 Score**
BiRNN	0.3195	0.8648	0.8385	0.9034	0.8697
TextCNN	1.1734	0.5494	0.5261	0.9888	0.6867

Table 1: Results of BiRNN in Comparison with the TextCNN Model

With these metrics established, the team wanted to explore what could happen if we leverage the strengths of both of these models to then produce an even higher accuracy.

Models

Ensemble Model

The team decided to analyze if we can leverage the strengths of both models and create an Ensemble model. We wanted to further break this down explore three types of Ensemble methods:

(a) A base ensemble that simply combines predictions from sub-models (TextCNN and BiRNN) and uses static aggregation methods to produce the final output.
(b) An ensemble model with added attention mechanism that assigns weights to the sub-model outputs. The attention layer provides interpretability by revealing the importance of each sub-model's contribution to the final decision.
(c) The same ensemble attention model as (b) but instead with custom embeddings into the framework. This is different from the other models that used the glove embeddings. These embeddings, such as positional or domain-specific features, enrich input data representations, enabling the ensemble to better align with task-specific characteristics. The attention mechanism leverages these enriched embeddings to assign more context-aware weights to the sub-models, resulting in more accurate predictions.

We have conducted experiments for all three. But the best outcome in the training set came from the ensemble attention model with custom embeddings. With that, we have decided to focus on testing more with this model.

The overall architecture is as follows:

1. **Custom Embedding Layer**: Generates dense vector representations for the input tokens using a pre-trained Word2Vec model or a similar approach.
2. **Attention Mechanism**: Dynamically assigns weights to sub-model outputs by evaluating their importance for the current input.
3. **Weighted Aggregation Layer**: Uses the attention weights to combine sub-model outputs into a single representation.
4. **Linear Layer**: Transforms the aggregated representation into logits for the classification task.
5. **Activation Layer**: Converts logits into probabilities using softmax or sigmoid for final predictions.

As the custom embeddings also needed a slightly different version of the data pre-processing pipeline as shown in the original final_project notebook, we made the following changes:

The pipeline starts by reading and organizing the IMDB dataset, each containing subfolders for positive and negative sentiment labels. Text data from these files is read, stripped of whitespace, and stored alongside corresponding sentiment labels. To enable custom embeddings, all texts from both the training and testing datasets undergo tokenization using a basic preprocessing function to generate a unified tokenized corpus.

For further dataset preparation, the training and testing data are tokenized into word-level sequences using a utility function, and a vocabulary is constructed with a specified minimum word frequency, reserving a special token <pad> for padding. Sentences are then transformed into sequences of numerical indices corresponding to the vocabulary, truncated or padded to a fixed length. Labels are converted into binary format for sentiment classification (positive as 1, negative as 0).

Finally, the data is wrapped in a custom dataset class and divided into batches using PyTorch DataLoader, ensuring efficient sampling and shuffling. This comprehensive preprocessing pipeline facilitates the integration of custom embeddings, which are built separately using a Word2Vec model to create dense vector representations for the vocabulary. Words not found in the embedding model are initialized with random vectors, ensuring a complete embedding matrix for use in the Attention Model.

To train the model, we used the following constant configurations:

weight_decay: 0.00001,
num_epochs: 5,
embedding_dim: 300,
num_layers: 2,
kernel_sizes: [3, 4, 5],
num_channels: [100, 100, 100],
hidden_size: 128,
vocabulary_size: len(vocab)

For experimentation purposes in finding the best configuration for this model in particular, we have decided to hyperparameter tune the following:

Learning Rate: [0.001, 0.01]

Batch Size: [64, 128]
Hidden Dimension: [128, 256]

Learning Rate	Batch Size	Hidden Dim	Train Loss	Train Acc	Test Loss	Test Acc	Precision	Recall	F1 Score	Best Test F1	Epoch Best
0.001	64	128	0.1821	0.9360	0.3712	0.8647	0.9015	0.8190	0.8582	0.8582	5
0.001	64	256	0.0634	0.9803	0.4174	0.8694	0.8973	0.8344	0.8647	0.8817	3
0.001	128	128	0.1345	0.9550	0.3743	0.8570	0.9135	0.7886	0.8465	0.8639	4
0.001	128	256	0.0870	0.9715	0.3609	0.8827	0.8664	0.9050	0.8853	**0.8885**	3
0.01	64	128	0.1889	0.9267	0.3467	0.8543	0.8749	0.8267	0.8502	0.8614	3
0.01	64	256	0.6832	0.5568	0.7023	0.5176	0.5317	0.2945	0.3790	0.6500	2

Table 2: Results of Hyperparameter Tuning of Ensemble Attention Model with Custom Embeddings

The best result turned out to be the model with the learning rate of 0.001, batch size 128, and hidden dimensions of 256. This produced the best F1 score of 0.8885 at epoch 3.

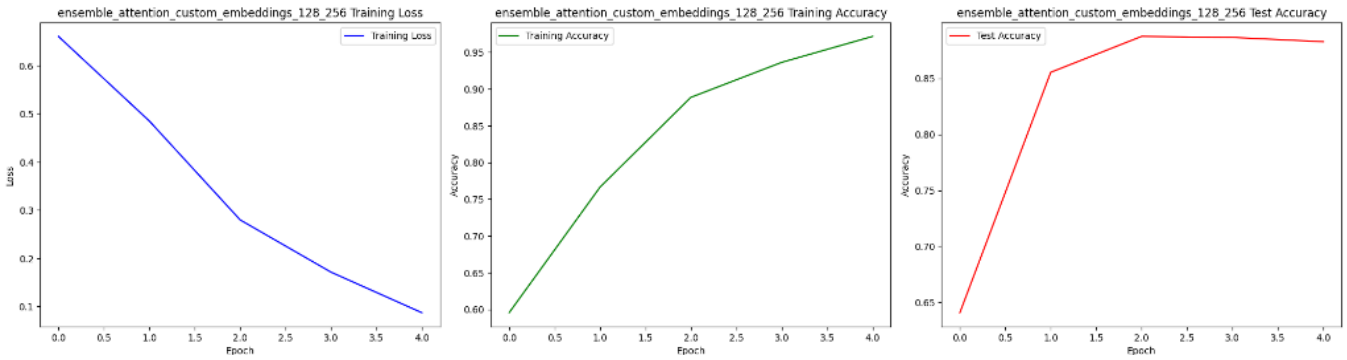

Figure 1: Result Graphs of the Best Configured Variant of the Ensemble Attention Model

With the best model and configuration, it was then further tested with the test set:

Loss	Accuracy	Precision	Recall	F1 Score
0.2002	0.9267	0.9190	0.9358	0.9273

Table 3: Test Results of the Best Configured Variant of the Ensemble Attention Model

The results show that the model performed better than the original TextCNN and BiRNN models, but we decided that we can explore even better models.

Roberta-BiLSTM Model

The second idea is to use RoBERTa, to generate deep contextual embeddings by capturing complex language patterns. By adding a Bidirectional LSTM (BiLSTM) layer on top of RoBERTa, the model processes these embeddings sequentially in both directions, enabling it to capture a more comprehensive contextual information across the entire text. To obtain the sentiment prediction, a linear layer is applied to the BiLSTM outputs, mapping the learned representations to the desired sentiment classes.

The overall architecture is as follows:

1. **RoBERTa Encoder**: The idea was to use RoBERTa as an encoder to generate deep contextual embeddings for the input tokens. The encoder takes in the tokenized input IDs and attention masks and processes them through the **roberta-base** model. This produces a last_hidden_state tensor with rich representations of the input text.
2. **Bidirectional LSTM (BiLSTM) Layer**: The output from the encoder was passed to a BiLSTM layer to capture sequential dependencies and contextual information from both directions of the text.
3. **Fully Connected (Linear) Layer**: After the BiLSTM layer, a dropout layer is applied to prevent overfitting by randomly zeroing some of the features. The output is then averaged across the sequence length to create a single vector for each input. This vector is fed into the fully connected layer, which produces logits corresponding to the two sentiment classes (positive and negative).

The pre-processing data pipeline has also been changed from the original one provided. The raw movie reviews are first collected from the ACL IMDB dataset's directory structure, separating them into training and test sets based on their parent folders. Each review is then passed through a preprocessing pipeline that converts the text to lowercase, removes URLs and non-alphabetic characters, and eliminates English stopwords. The remaining terms are lemmatized to reduce them to their base forms, ensuring a more normalized input representation. Subsequently, these cleaned reviews are tokenized with the RoBERTa tokenizer, which encodes the text into subword units, handles padding, and truncated sequences to a predefined length. Finally, the binary sentiment labels are mapped to 0 for negative and 1 for positive.

To train the model, we used the following constant configurations:

Learning Rate: `0.00001`
Batch Size: `64`
Hidden Dimension: `128`
Dropout Rate: `0.1`
Number of LSTM Layers: `1`
Number of Epochs: `5`

For experimentation purposes in finding the best configuration for this model in particular, we have decided to hyperparameter tune the following:

Learning Rate: [0.001, 0.01]
Batch Size: [64, 128]
Hidden Dimension: [128, 256]

Learning Rate	Batch Size	Hidden Dim	Train Loss	Train Acc	Test Loss	Test Acc	Precision	Recall	F1 Score	Best Test F1
0.0001	64	128	0.6938	0.4973	0.6944	0.5000	0.2500	0.5000	0.3333	0.3333
0.0001	64	256	0.6935	0.4975	0.6932	0.5000	0.2500	0.5000	0.3333	0.3345
0.0001	128	128	0.1001	0.9658	0.4156	0.8705	0.8756	0.8705	0.8701	0.8831
0.0001	128	256	0.0901	0.9679	0.3713	0.8760	0.8763	0.8760	0.8760	0.8854
0.00001	64	128	0.1275	0.9536	0.2883	0.8975	0.8979	0.8975	0.8975	**0.9042**
0.00001	64	256	0.1288	0.9517	0.2941	0.9019	0.9023	0.9019	0.9019	0.9019
0.00001	128	128	0.1631	0.9374	0.2656	0.9022	0.9023	0.9022	0.9022	0.9022
0.00001	128	256	0.1621	0.9375	0.2649	0.8994	0.9009	0.8994	0.8993	0.9015
0.000001	64	128	0.2804	0.8853	0.2754	0.8861	0.8863	0.8861	0.8861	0.8861
0.000001	64	256	0.2775	0.8860	0.2709	0.8869	0.8873	0.8869	0.8869	0.8869
0.000001	128	128	0.2994	0.8761	0.2844	0.8822	0.8822	0.8822	0.8822	0.8822
0.000001	128	256	0.2949	0.8762	0.2809	0.8829	0.8831	0.8829	0.8829	0.8829

Table 4: Results of Hyperparameter Tuning of RoBERTa-BiLSTM Model

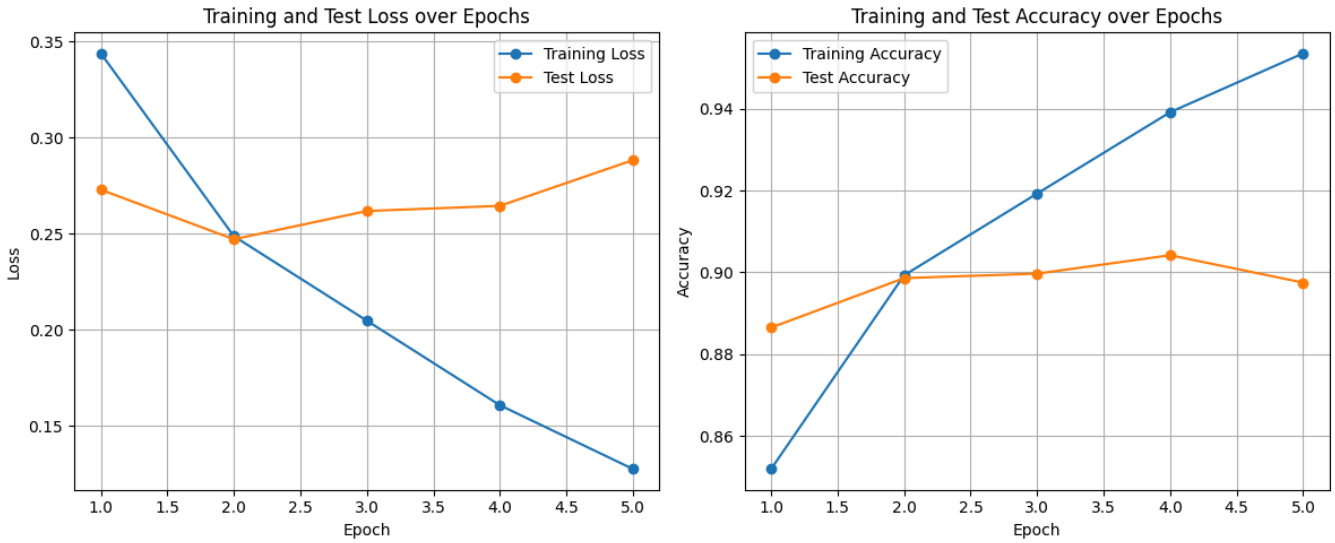

Figure 2: Result Graphs of the Best Configured Variant of the RoBERTa-BiLSTM Model

With the best model and configuration, it was then further tested with the test set:

Loss	Accuracy	Precision	Recall	F1 Score
0.1750	0.9392	0.9368	0.9418	0.9393

Table 5: Test Results of the Best Configured Variant of the RoBERTa-BiLSTM Model

The results show that the model performed better than the Ensemble we have built previously. But before settling on a model, we wanted to explore the possibility of an even better one.

RoBERTa-large Model

The team wanted to explore the leverage that Large Language Models have in sentiment analysis, and therefore pursued transformer-based architectures, specifically focusing on RoBERTa. While our previous RoBERTa-BiLSTM implementation showed promise, we hypothesized that the additional BiLSTM layer might be unnecessary given RoBERTa's capability to capture complex contextual relationships.

Given our computational resource constraints, we adopted Low-Rank Adaptation (LoRA) to fine-tune RoBERTa-large. While full fine-tuning would typically yield optimal results, LoRA offered an efficient alternative by updating only a small set of task-specific parameters while keeping the base model frozen. This approach significantly reduced the memory footprint during training while still allowing us to leverage RoBERTa-large's full capabilities. Through LoRA, we maintained strong model performance while achieving greater computational efficiency compared to traditional fine-tuning methods.

The overall architecture is as follows:

Base Model Architecture (RoBERTa-large)
RoBERTa comes in three versions: **base**, **large**, and **small**, which differ in size and number of parameters. These versions allow flexibility in choosing a model based on computational resources and task complexity. Below is a table detailing the parameters for each version:

Model (RoBERTa)	Parameters	Layers	Attention Heads	Hidden Size	Vocabulary Size	Maximum Sequence Length
Large	**355M**	**24**	**16**	**1024**	**50,265**	**512**
Base	125M	12	12	768	50,265	512
Small	84M	6	8	768	50,265	512

Table 6: Parameters of the Various Versions of the RoBERTa Model

RoBERTa-large was chosen as our base model architecture due to its superior capabilities in handling complex sentiment analysis tasks. Movie reviews often contain complex and mixed opinions that require sophisticated language understanding. With 355 million parameters (compared to 125M in base and 84M in small versions), RoBERTa-large provides enhanced capability to capture nuanced expressions and contextual relationships in text.

The model's architecture offers several key advantages for sentiment analysis. Its deeper layer structure (24 transformer layers versus 12 in base and 6 in small) enables more sophisticated processing of language hierarchies and better captures complex sentiment relationships across text. The larger hidden size of 1024 dimensions (compared to 768 in smaller versions) provides richer representation of sentiment features and increased capacity to store complex emotional patterns. Additionally, with 16 attention heads (versus 12 in base and 8 in small), the model can better process multiple aspects of sentiment simultaneously, making it particularly effective at understanding reviews with mixed or nuanced opinions.

LoRA (Low-Rank Adaption) Configuration
LoRA (Low-Rank Adaptation) is an efficient fine-tuning technique that significantly reduces the number of trainable parameters in our model. Instead of fine-tuning the entire RoBERTa-large model with its 355M parameters, LoRA selectively updates specific matrices, effectively reducing both memory usage and training time while maintaining strong performance on sentiment analysis tasks. This approach is particularly valuable when working with RoBERTa-large, where traditional full fine-tuning would be computationally expensive.

Due to limitation to our computational resource, we could only execute a single training run, below are the LoRA configuration that we trained the model with:

1. **Rank Parameter (r=128) :** The rank parameter defines how many dimensions we use in our low-rank matrices. We choose 128 to be our rank parameter to reduce the trainable parameters from 355M to a manageable size. It provides sufficient complexity to capture sentiment patterns while maintaining computational efficiency. A higher rank would increase the computational

demands, while a lower rank might not capture the complexity of sentiment expressions effectively.

2. **Alpha Parameter (512) :** This scaling factor controls the magnitude of LoRA updates during training. We selected a value of 512 to enable sufficiently big updates to the model's behaviour. Higher values could lead to training instability, while lower values might result in insufficient learning of task-specific patterns.
3. **Dropout Rate (0.05) :** We implemented a dropout rate of 0.05 for regularization. This relatively low value maintains model stability while providing sufficient regularization to prevent overfitting, ensuring the model generalizes well to new movie reviews.
4. **Target Modules (["query", "key", "value"]) :** Our configuration focuses on adapting the attention mechanism components - specifically the query, key, and value matrices.
5. **Task Type (SEQ_CLS) :** The sequence classification setting aligns with our binary sentiment analysis task of classifying movie reviews as positive or negative.

The pre-processing of data is also different from the original one presented in the final_project notebook.

The initial data preprocessing pipeline, similar to our base model, applies several fundamental techniques to both training and testing data. First, the text undergoes initial cleaning to standardize format and remove unnecessary whitespace. Next, sentences are broken down into individual words through tokenization. Common words that don't carry significant meaning are then removed through stop word filtering. Finally, lemmatization converts all words to their base forms to reduce vocabulary complexity.

After these core preprocessing steps, we leverage RoBERTa tokenizers to convert the processed text into numerical embeddings. These embeddings are crucial as they capture both word relationships and contextual information within sentences. RoBERTa's tokenization is particularly well-suited for sentiment analysis of movie reviews due to its ability to effectively handle informal language patterns commonly found in user-generated content. It excels at processing emotional expressions and intensifiers (such as "very", "really", "absolutely") that are crucial for sentiment understanding. The tokenizer also maintains important contextual relationships between words, allowing it to capture subtle sentiment indicators that depend on the surrounding text context.

To train the model, we used the following constant configurations:

training_batch_size: 16
evaluation_batch_size: 32
gradient_accumulation: 4
learning_rate: 1e-5
warmup_ratio: 0.1
weight_decay: 0.1

The following is the learning process:

1. **Loss and Optimization:** The model uses a cross-entropy loss function specifically designed for binary classification tasks.

2. **Linear learning rate scheduler with warmup:** Implements a gradual increase in learning rate during initial training.
3. **Gradient clipping at 1.0:** Sets maximum threshold for gradient values to prevent exploding gradients.

For experimentation purposes in finding the best configuration for this model in particular, we have decided to hyperparameter tune the following:

This model is complex and is computationally heavy to train. Therefore, we decided to not perform hyperparameter tuning for the model. The following were the results:

Train Loss	Train Acc	Test Loss	Test Acc	Precision	Recall
0.1799	0.9344	0.1781	0.9411	0.9362	0.9323

Table 7: Results of Lora-RoBERTa Model with the IMDB Dataset

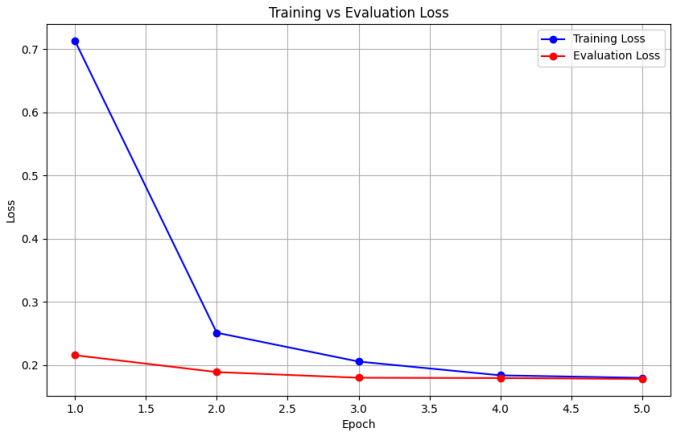

Figure 3: Result Graph of Training vs. Evaluation Loss for Lora-RoBERTa Model

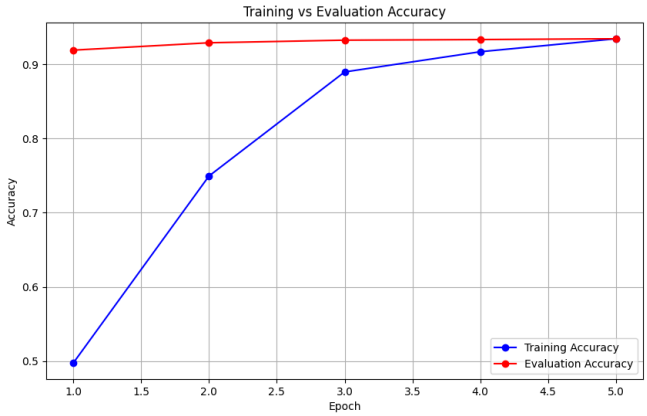

Figure 4: Result Graph of Training vs. Evaluation Accuracy for Lora-RoBERTa Model

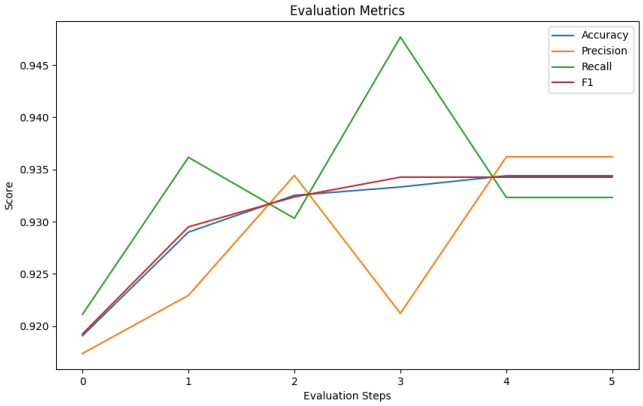

Figure 5: Result Graph of Evaluation Metrics for Lora-RoBERTa Model

To ensure that the model performs just as well with the test set, we have tested it on there as well:

Loss	**Accuracy**	**Precision**	**Recall**	**F1 Score**
0.1634	0.9408	0.9425	0.9388	0.9407

Table 8: Test Results of the Lora-RoBERTa Model

With that, the team has decided to choose Lora-RoBERTa as our final model due to the high metrics it displays in both training and test.

Conclusion

The team has showcased a progression starting from the very base BiRNN and TextCNN models, and explored various models that kept improving. We have displayed constant improvement in all the models that we have experimented with, which demonstrates the effectiveness of iterative experimentation and refinement in achieving superior performance. By systematically evaluating and comparing models, such as BiRNN, TextCNN, and other advanced architectures, we highlighted the incremental gains achieved through each upgrade. The progression culminated in the Lora-RoBERTa model, which excelled in sentiment analysis tasks due to its robust pretraining and fine-tuning capabilities, making it the optimal choice for this application.