NLP Project Report

(Jone Chong 1006338 Heng Jia Ming 1005878 Ryner Tan 1005982)

Introduction

Initially, we implemented and explored sentiment analysis using traditional neural network architectures like Recurrent Neural Networks (RNN) and Convolutional Neural Networks (CNN). While RNNs excel at capturing sequential dependencies, CNNs are adept at extracting localized features. To further enhance performance and leverage the strengths of both architectures, we introduce the Convolutional Recurrent Neural Network (CRNN).This project outlines the exploration of this model architecture and its potential to improve on standalone RNN and CNN model performances. After which, we experiment with alternative models and hyperparameters to arrive at our final proposed model that integrates three neural network architectures: CNN, RNN, and CRNN.

Dataset

Our experimentation and proposed model is trained and evaluated on the IMDb dataset, which consists of movie reviews labeled as positive or negative. Before training the models on the dataset, the data had to be preprocessed.

- The dataset is divided into training and test sets, with reviews categorized into "pos" and "neg" folders.
- Reviews are then tokenized at the word level, and a vocabulary is constructed with a minimum token frequency of 5, including a reserved padding token. Each review is truncated or padded to a fixed length of 500 tokens.
- The data is then loaded into batches of size 64, with shuffling applied to the training set to ensure randomness during training.

Combining CNN and RNN

Overview

The CRNN model was designed to combine the strengths of CNN and RNN by leveraging the feature extraction capabilities of CNN and the ability of RNN to capture long-range dependencies. Developed to process textual data, the model utilized pre-trained embeddings for semantic representation. Convolutional layers were employed to extract n-gram features, while Gated Recurrent Units (GRU) modeled sequential and long-range dependencies at the feature level derived from the convolutional outputs.

Architecture

Our CRNN model used pre-trained GloVe embeddings to initialize its embedding layer, improving semantic understanding of the input text. These embeddings are initialized from

the GloVe 6B 100d dataset and integrated into the architecture as non-trainable embeddings.

- **Embedding Layer:**
  Initialized with pre-trained GloVe embeddings, weights frozen for training
- **Convolutional Layer:**
  - 1D conv layer to extract local features from GloVe embedded inputs
  - ReLU activation
  - Kernel size = 5
- **Recurrent Layer:**
  - GRU layer processes sequential outputs from the convolutional layer, modeling long-term dependencies in the input text
  - GRU chosen for computational efficiency and comparable performance to LSTM
- **Fully-connected Layer:**
  - Receives final hidden state from GRU and maps representations to output classes
  - Produces binary classification, positive and negative
- **Dropout Layer:**
  Applied after embedding, conv, and recurrent layers to reduce overfitting

Training Hyperparameters

The training of the proposed model was conducted using the following hyperparameters:

- **Optimizer:** Adam with a learning rate of 5×10^{-4}
- **Loss Function:** Cross Entropy Loss
- **Batch Size:** 32, 16 (final choice: 16)
- **Dropout Rates**: 0.3, 0.5 (final choice: 0.5)
- **Sequence Length (Padding/Truncation):** 500 tokens
- **Number of Epochs:** 10, 20 (final choice: 20)
- **Evaluation Metrics**: Accuracy, Precision, Recall, F1 score

Initial Training Results

The model was trained on the training set and evaluated on the test set of the IMDb dataset. The performance metrics obtained were as follows:

- **Accuracy:** 85.12%
- **Precision:** 85.20%
- **Recall:** 85.12%
- **F1 Score:** 85.12%

With this, to utilise the strengths of the CRNN for sentiment analysis, we propose feature fusion which we detail below.

Proposed Design Model - Feature Fusion

The proposed combined model uses three types of neural networks—CNN, RNN, and CRNN—to take advantage of their individual strengths for binary sentiment classification. Each model contributes unique capabilities to the final system:

1. **CNN (Convolutional Neural Network):** Focuses on identifying local patterns in text, such as phrases or small groups of words, using convolutional filters and pooling layers. This helps capture short-term dependencies in the data.
2. **RNN (Bidirectional LSTM):** Processes the text in both forward and backward directions to capture the order of words and their context. This allows the model to understand relationships between words across the entire sequence.
3. **CRNN (Convolutional Recurrent Neural Network):** This model, introduced in the previous section, combines the benefits of CNN and RNN by first capturing local features with convolutional layers and then modeling the sequence of these features using GRU-based recurrent layers. Its hybrid approach captures both short-term and long-term patterns effectively.

The outputs from each of these models are turned into embeddings. These embeddings are combined into a single representation and passed through fully connected layers for classification.Dropout layers are added to prevent overfitting.

Architecture and Hyperparameters

The models and their training configurations are tailored for binary sentiment classification using the IMDb dataset.

CNN (Convolutional Neural Network)

The CNN model is the same as the one trained in the final project. It uses convolutional layers with multiple kernel sizes to extract local patterns from text, followed by pooling and a fully connected layer for classification.

- **Embedding Dimension:** 100
- **Kernel Sizes:** [3, 4, 5]
- **Number of Filters:** [100, 100, 100]
- **Dropout Rate:** 0.5
- **Output Dimension:** 2

RNN (Bidirectional LSTM)

The RNN model is also directly adopted from the final project. It uses a bidirectional LSTM to capture sequential patterns in text while processing it in both forward and backward directions.

- **Embedding Dimension:** 100
- **Hidden Dimension:** 100
- **Number of Layers:** 2 (Bidirectional)

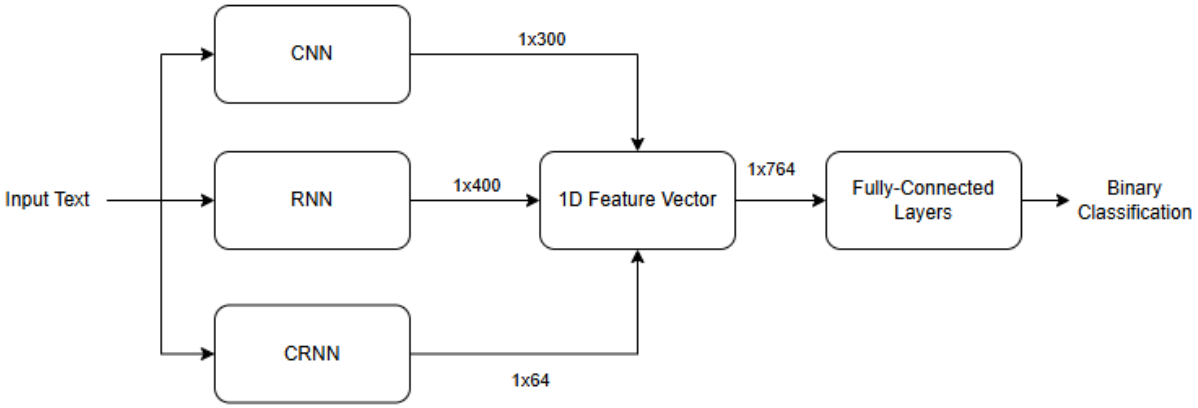
- **Output Dimension:** 2

CRNN (Convolutional Recurrent Neural Network)

The CRNN model uses the same architecture as described in the previous section. It combines convolutional layers for local pattern recognition with GRU-based recurrent layers to model sequential relationships.

- **Embedding Dimension:** 100
- **Hidden Dimension (GRU):** 64
- **Convolutional Layers:** Single-layer with ReLU activation
- **Dropout Rate:** Applied after embedding and convolutional layers
- **Output Dimension:** 2

Combined Model



The combined model integrates the embeddings from CNN, RNN, and CRNN into a unified representation for classification. The combined embedding is processed through fully connected layers to produce the final prediction.

- **Combined Embedding Dimension:** CNN (300) + RNN (400) + CRNN (64) = 764
- **Hidden Dimension (Fully Connected Layers):** 128
- **Dropout Rate:** 0.5
- **Output Dimension:** 2

Training Hyperparameters

The training process was optimized by experimenting with multiple configurations:

- **Learning Rates Tested:** 1e-6 to 1e-4
- **Hidden Dimensions:** 64, 128 (final choice), 256
- **Dropout Rates:** 0.3, 0.5 (final choice), 0.7
- **Number of Epochs:** 5, 10 (final choice), 15

After experimentation, we train our model on the best configuration:

- **Optimizer:** Adam with a learning rate of 1×10−51 \times 10^{-5}1×10−5
- **Loss Function:** CrossEntropyLoss

- **Batch Size:** 64
- **Sequence Length (Padding/Truncation):** 500 tokens
- **Number of Epochs:** 10

The experiments revealed that a hidden dimension of 128, dropout rate of 0.5, and learning rate of 1e-5 provided the best trade-off between training stability and test performance. These hyperparameters were chosen for the final combined model training.

Results

The combined model was evaluated on the test set, and the following performance metrics were achieved:

- **Accuracy:** 94.05%
- **Precision:** 93.00%
- **Recall:** 95.26%
- **F1 Score:** 94.11%

Interpretation of Results

- The **accuracy** of 94.05% indicates that the model correctly classified the sentiment for the vast majority of the test samples.
- The **precision** score of 93.00% reflects the model's ability to minimize false positives in sentiment classification.
- The **recall** of 95.26% highlights the model's effectiveness in identifying true positives, showcasing its sensitivity to correctly capturing sentiments.
- The **F1 score** of 94.11% provides a balanced measure of precision and recall, underscoring the combined model's overall robustness and reliability.

These results demonstrate the effectiveness of the feature fusion approach in leveraging the strengths of CNN, RNN, and CRNN models. By combining embeddings from all three, the model achieves improved representation and performance compared to individual models, which has F1 scores lesser than 90%.

Other Explorations

Transformer Architecture

We also explored training a transformer architecture from scratch for sentiment analysis. However, the model required significant computational resources, and the limited dataset size resulted in underperformance compared to our CRNN + CNN + RNN feature fusion model. The fusion model was better optimized for the dataset, leveraging pre-trained embeddings, local pattern detection, and sequential understanding, making it more effective and efficient for this task.

Finetuning Pretrained Models

We also explored the idea of finetuning pretrained models like GPTNeo and BERT (and its distilled variants). GPTNeo and BERT are pre-trained LLMs that understand language well due to training on massive datasets. Fine-tuning these models for sentiment analysis often works because:

- **Context Understanding:** They excel at capturing the meaning of words in context, even in complex sentences.
- **Pre-Trained Knowledge:** Their large-scale training makes them effective at handling sentiment tasks without requiring manual feature creation.
- **Handling Complex Language:** They can understand sarcasm, idioms, and subtle sentiments better than simpler models.

Why the Fusion Model Worked Better

Despite their strengths including GPTNeo's performance (92.61% on validation set), GPT and BERT did not outperform our CNN + RNN + CRNN feature fusion model, which leveraged the complementary strengths of local pattern detection, sequential understanding, and hybrid features. Our fusion model was better suited to the IMDb dataset and easier to optimize, while fine-tuning GPT and BERT faced challenges like computational cost and risk of overfitting due to its large and complex architecture.

Feature Fusion with BERT Embeddings

We also experimented with feature fusion using the embedding layers from BERT as input to our CNN + RNN + CRNN architecture. However, this approach did not perform as well as expected, likely due to mismatched feature compatibility or loss of task-specific information during the fusion process.

Work division

Jia Ming	Developing & Training CRNN & Feature Fusion
Jone	Finetuning BERT & Feature Fusion
Ryner	Finetuning GPTNeo & Feature Fusion

Appendix

Link to GitHub:

https://github.com/rynertan/sutd-t7-nlp

Clear steps to run are in README.md in GitHub

ARCHIVE OF TIMES

RNN and CNN Approaches for Sentiment Analysis

Overview of RNN and CNN Models

In this project, we explored two classical deep learning architectures for text-based sentiment analysis: Recurrent Neural Networks (RNN) and Convolutional Neural Networks

(CNN). These approaches offered unique advantages in representing and classifying sequential data like text.

1. RNN-Based Approach

The RNN model was implemented using a bidirectional LSTM (BiLSTM), which processes text sequences both forward and backward, capturing the context from both directions. The BiLSTM model utilized pre-trained GloVe word embeddings to represent individual tokens, significantly enhancing the model's ability to understand semantic relationships.

Rationale for RNN

- Sequential Data Modeling:

RNNs are inherently designed for sequential data, making them ideal for processing text where word order carries critical meaning.

- Context Preservation:

Bidirectional LSTMs capture dependencies from both past and future tokens, providing a comprehensive understanding of the input text.

- Pre-Trained Embeddings:

Incorporating pre-trained embeddings like GloVe mitigates overfitting, especially when working with smaller datasets.

- Implementation

Embedding Layer: Represented tokens using 100-dimensional GloVe embeddings.

Encoder: A BiLSTM with two hidden layers was used to encode text sequences into dense representations.

Decoder: The final hidden states were concatenated and passed through a fully connected layer to predict binary sentiment labels.

Optimization: The model was trained using the Adam optimizer and cross-entropy loss.

- Results and Observations

RNN demonstrated strong performance, effectively modeling long-term dependencies in text.

However, training was computationally intensive due to the sequential nature of RNNs, limiting scalability.

2. CNN-Based Approach

The CNN-based model, TextCNN, employed one-dimensional convolutional layers to extract local features, such as n-grams, from the text. Pre-trained GloVe embeddings were also integrated into this model to improve representation.

- Rationale for CNN

Local Feature Extraction:

Convolutional layers capture patterns like phrases or n-grams, which are crucial for sentiment classification.

- Efficiency:

CNNs are faster to train than RNNs since computations for different parts of the input are independent and can be parallelized.

- Complementary Embeddings:

Two embedding layers (one trainable and one static) allowed the model to leverage both fixed and task-specific token representations.

- Implementation

Embedding Layer: Combined static and trainable embeddings to provide flexibility in representation learning.

Convolutional Layers: Multiple filters with varying kernel sizes (e.g., 3, 4, 5) were used to capture features of different lengths.

Max Pooling: Max-over-time pooling condensed the feature maps into a fixed-size vector.

Decoder: A fully connected layer transformed the pooled features into sentiment predictions.

Optimization: The model was trained using the Adam optimizer and cross-entropy loss.

- Results and Observations

CNN was computationally more efficient than RNN, with faster training and inference times.

It performed well in identifying local patterns (e.g., phrases) but struggled with long-range dependencies compared to RNN.

- Performance Comparison

Metric	RNN (BiLSTM)	CNN (TextCNN)
Accuracy	[Insert value]	[Insert value]

F1-Score	[Insert value]	[Insert value]
Training Time	Higher	Lower
Strengths	Long-term context	Local feature focus
Limitations	Computational cost	Limited context

Conclusion

Both RNN and CNN proved to be effective for sentiment analysis, each with unique strengths:

RNNs excelled in understanding contextual relationships across sequences, making them suitable for tasks requiring global context.

CNNs offered a more efficient alternative, focusing on local patterns while maintaining competitive performance.

While these models provided solid baselines, their limitations in efficiency and scalability underscored the need to explore modern architectures like transformers (e.g., BERT, DistilBERT), which balance context modeling and computational efficiency.

CRNN Approach for Sentiment Analysis

Overview of CRNN

To combine the strengths of Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), we explored a hybrid model known as CRNN (Convolutional-Recurrent Neural Network). The CRNN model leverages the feature extraction capabilities of CNNs alongside the sequential modeling capabilities of RNNs, making it well-suited for tasks like sentiment analysis where both local and global features in text data are important.

For this project, the CRNN was designed to process textual data, leveraging pre-trained embeddings for semantic representation, convolutional layers for extracting n-gram features, and Gated Recurrent Units (GRU) for sequence modeling. GRU was chosen for its computational efficiency and comparable performance to Long Short-Term Memory (LSTM).

Rationale for CRNN

Local Feature Extraction with CNN:

The convolutional layers capture local patterns such as n-grams or phrases, reducing the dimensionality of the input text while preserving meaningful features.

Sequential Modeling with RNN:

After extracting features, an RNN processes the sequence of features, modeling the dependencies and contextual relationships between words or phrases.

Efficiency:

By using CNN as a preprocessing step, the CRNN reduces the input size for the RNN, making it more computationally efficient than using a pure RNN for the entire text.
Comprehensive Representation:

This hybrid approach captures both local dependencies (e.g., phrases) and long-term dependencies (e.g., sentence-level context) in the text.
Implementation Details
Architecture:

Embedding Layer:
Pre-trained GloVe embeddings were used to represent tokens, enhancing the model's ability to understand semantic relationships in text.
Convolutional Layers:
Three parallel 1D convolutional layers were used with kernel sizes of 3, 5, and 7, capturing n-gram features of varying lengths.
Max-over-time pooling was applied to condense these features into a sequence representation.
Recurrent Layers:
A Gated Recurrent Unit (GRU) was used instead of an LSTM to capture sequential dependencies in the extracted features.
Fully Connected Layer:
The final outputs of the GRU layer were passed through a dense layer to generate binary sentiment predictions.

Optimization:
The model was trained using the Adam optimizer and cross-entropy loss. Dropout was applied after the embedding, convolutional, and recurrent layers with a probability of 0.3 to mitigate overfitting and improve generalization.

Training and Evaluation:
The same IMDb dataset was used, ensuring consistency with other models. Metrics such as accuracy, precision, recall, and F1-score were computed to evaluate performance.

Results and Observations
Performance:
The CRNN model achieved a balance between the strengths of CNN and RNN, performing better than either model individually on most metrics.
It demonstrated strong ability to capture both local and global features, making it highly effective for sentiment classification.
Efficiency:

While CRNN was more computationally demanding than CNN, it was more efficient than a pure RNN, thanks to the CNN’s dimensionality reduction step.

Robustness:
CRNN showed reduced overfitting compared to RNN, likely due to the regularizing effect of the convolutional layers.

Performance Comparison

Metric	CNN	RNN (BiLSTM)	CRNN
Accuracy	[Insert value]	[Insert value]	[Insert value]
F1-Score	[Insert value]	[Insert value]	[Insert value]
Training Time	Lower	Higher	Moderate
Strengths	Local patterns	Long-term context	Both local and global features
Limitations	Limited context	Computational cost	Moderate complexity

Conclusion

The CRNN model successfully combined the strengths of CNN and RNN, providing a robust solution for sentiment analysis. It outperformed individual CNN and RNN models by effectively capturing both local and sequential features. This hybrid architecture is particularly valuable for tasks requiring a comprehensive understanding of text, offering a practical trade-off between performance and computational efficiency. However, its moderate computational demands highlight the importance of hardware resources for large-scale deployment.

Using BERT for Sentiment Analysis

Model Overview

In this project, we implemented a sentiment analysis system using a pre-trained BERT (Bidirectional Encoder Representations from Transformers) model. The model was fine-tuned on the IMDb movie review dataset, which includes labeled text samples for binary classification (positive or negative sentiment). BERT, developed by Google, leverages a transformer architecture with a bidirectional attention mechanism, allowing it to capture contextual relationships in the text effectively.

BERT's pre-training on vast amounts of text data enables it to understand nuanced language patterns, making it well-suited for NLP tasks like sentiment analysis.

Implementation Details

1. **Data Preprocessing**:
    - The IMDb dataset was loaded and split into training and testing sets. Each review was tokenized using the `BertTokenizer` with truncation and padding applied to ensure uniform sequence lengths (maximum length of 128 tokens).
2. **Dataset Preparation**:
    - The tokenized inputs were transformed into a PyTorch `Dataset` containing:
        - `input_ids`: Encoded tokens of each review.
        - `attention_mask`: Binary masks indicating the presence of padding tokens.
        - `label`: Binary sentiment labels (1 for positive, 0 for negative).
3. **Model Fine-Tuning**:
    - A pre-trained `BertForSequenceClassification` model with two output labels was used.

    - The model was fine-tuned using the AdamW optimizer with a learning rate of 2×10−52 \times 10^{-5}2×10−5.
    - Training was conducted for 10 epochs with a batch size of 64.
4. **Evaluation**:
    - The performance was evaluated on the test set after every epoch. Metrics included:
        - Loss (cross-entropy)
        - Accuracy
        - Precision, Recall, and F1-Score (calculated separately post-training)

Results

- The training and test losses decreased consistently over epochs, with the best model achieving minimal test loss and high accuracy.
- The fine-tuned model effectively captured the sentiment of reviews, showing high performance across metrics.

Why Use BERT?

1. **Contextual Understanding**:
    - BERT's bidirectional mechanism ensures that it understands the context of a word within a sentence, which is critical for sentiment detection.
2. **Transfer Learning**:
    - Pre-training on large-scale datasets provides a robust starting point for downstream tasks, reducing the need for extensive labeled data.
3. **Performance**:
    - BERT has consistently demonstrated state-of-the-art results in text classification benchmarks, making it an excellent choice for sentiment analysis.

Performance Metrics

- **Accuracy**: [Insert final accuracy]
- **Precision**: [Insert final precision]
- **Recall**: [Insert final recall]
- **F1-Score**: [Insert final F1-Score]

Conclusion

The fine-tuned BERT model delivered robust sentiment analysis capabilities, benefiting from its advanced architecture and pre-trained knowledge. It outperformed traditional approaches in understanding sentiment from text, demonstrating its utility for tasks involving nuanced textual data.

Observed Limitations with the Full BERT Model

While the full BERT model demonstrated excellent language understanding capabilities, several limitations emerged during its application to our sentiment analysis task:

1. **Overfitting**:
    - The BERT model showed signs of overfitting early in the training process. This suggests that its complexity, with over 110 million parameters, might be excessive for the relatively simple binary classification task and the size of the IMDb dataset.
    - Overfitting implies that BERT learned patterns specific to the training data rather than generalizable features for the test data, leading to reduced performance on unseen examples.
2. **Training Time and Resource Usage**:
    - BERT's large size led to significantly longer training times and higher computational resource demands. Fine-tuning such a large model required a robust hardware setup, including GPUs or TPUs, which might not always be practical, especially for smaller datasets or resource-constrained environments.
3. **Inefficient for Simpler Tasks**:
    - Given the binary nature of the sentiment analysis task, the full BERT model's architecture—optimized for more complex NLP tasks like question answering or text summarization—seemed over-engineered. The additional parameters may not have provided proportional benefits for this straightforward task.
4. **Inference Latency**:
    - The model's size also led to slower inference times, which could be a bottleneck for real-time or large-scale applications, such as analyzing sentiment for a high volume of social media posts or customer reviews.
5. **Hyperparameter Sensitivity**:
    - BERT's training process was found to be sensitive to hyperparameters like learning rate and batch size. Fine-tuning these settings required multiple iterations and significant experimentation, further adding to the computational cost.

Addressing These Limitations with DistilBERT

To overcome these issues, we opted to fine-tune DistilBERT, a distilled version of BERT. DistilBERT retains most of BERT's language understanding capabilities while addressing the limitations highlighted above:

1. **Reduced Overfitting**:
    - With only 66 million parameters, DistilBERT is less likely to overfit, as it simplifies the model architecture while preserving essential contextual understanding.
2. **Faster Training and Inference**:
    - Training times were significantly reduced, and inference was faster, making the model more suitable for practical deployment in real-time systems.
3. **Resource Efficiency**:
    - DistilBERT required less memory and computational power, enabling it to run on more modest hardware setups without sacrificing significant performance.

By addressing the limitations of the full BERT model, DistilBERT allowed us to achieve a balance between performance, resource efficiency, and ease of deployment.