Midterm Project Report - CSE 587

1. Introduction

1.1 Problem Definition

The task is to develop a multi-headed model capable of accurately detecting different types of toxicity, such as **threats, obscenity, insults, and identity-based hate.** The model will be trained on a dataset of comments from Wikipedia's talk page edits. The goal is to enhance toxicity detection to promote more constructive and respectful online discussions.

1.2 Objective

The primary goal of this project is to develop a robust multi-headed model capable of detecting various forms of toxicity, including threats, obscenity, insults, and identity-based hate. This will be achieved by leveraging word embeddings and convolutional/recurrent neural networks (CNNs/RNNs) to effectively capture contextual and sequential relationships in text data.

2. Dataset Curation

2.1 Dataset Selection

The dataset used in this project consists of a large collection of Wikipedia comments that have been annotated by human raters for various types of toxic behavior. The toxicity categories include toxic, severe toxic, obscene, threat, insult, and identity-based hate. Each comment is labeled with the presence or absence of these toxicity types, allowing for multi-label classification. There are total 153164 data points.

3. Methodology

3.1 Word Embeddings

In my methodology, I employ word embeddings to transform textual data into dense vector representations, allowing the model to capture semantic relationships between words. I experiment with both untrained Word2Vec embeddings, trained on my dataset, and pre-trained Google News Word2Vec embeddings to evaluate their effectiveness in contextual understanding. I initialize the embedding layer in my RNN model with these word vectors, enabling efficient feature extraction while either learning from scratch or leveraging pre-trained linguistic knowledge.

3.2 Model Architecture

In my model architecture, I compare three different approaches to text classification using word embeddings: a pure RNN model (GRU-based), a CNN model, and a hybrid CNN-RNN model. The RNN model leverages sequential dependencies with stacked GRU layers, while the CNN model extracts local text features through convolutional layers before classification.

The hybrid CNN-RNN model combines the strengths of both architectures by using CNN layers for feature extraction followed by bidirectional GRU layers to capture temporal dependencies, enabling a more comprehensive understanding of textual data.

3.3 Training Procedure

Training Process Explanation

Optimization Algorithm

I use the **Adam optimizer** for training all models, as it combines the benefits of both Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp), leading to efficient and adaptive learning. The Adam optimizer is well-suited for handling sparse gradients and noisy data, making it effective for natural language processing tasks.

Loss Function

Since the problem involves **multi-label classification**, I use **binary cross-entropy loss**, which calculates the loss for each independent class separately. This is appropriate because each comment in the dataset can belong to multiple toxicity categories simultaneously.

Hyperparameters

I fine-tune the model using the following hyperparameters:

- **Learning Rate**: 0.0010.001 (default Adam setting)
- **Batch Size**: 3232
- **Epochs**: 44 (early stopping is used to prevent overfitting)
- **Validation Split**: 0.10.1 (10% of training data used for validation)
- **Dropout Rate**: 0.20.2 (applied to embeddings for regularization)
- **GRU Units**: 6464 (first GRU layer), 3232 (second GRU layer in RNN and CNN-RNN models)
- **CNN Filters**: 128128 (first Conv1D layer), 6464 (second Conv1D layer)
- **Kernel Sizes**: 55 and 33 (used in CNN-based models)
- **Max Pooling Size**: 22

4. Results and Analysis

4.1 Model Performance

Model Performance Discussion

The comparison between different models—**Pretrained Google Word2Vec with Pure RNN (GRU), Non-pretrained Word2Vec with Pure RNN (GRU), Non-pretrained Word2Vec with Pure CNN, and Non-pretrained Word2Vec with Hybrid CNN + RNN**—reveals important insights into their effectiveness in toxic comment classification.

The **Pretrained Word2Vec + GRU** model achieves **high validation accuracy (~98.8%) with low validation loss (0.0478)** while keeping the **embedding layer frozen**, reducing the number of trainable parameters. This leads to **faster training and efficient generalization**. However, **the Non-pretrained Word2Vec + GRU model**, which fine-tunes the embedding layer, shows **slightly higher accuracy (~99.3%)** but with **significantly higher training time (165ms/step vs. 36ms/step)** and a much larger number of trainable parameters, indicating that fine-tuning embeddings may improve performance at the cost of computational efficiency.

The **Pure CNN model** exhibits strong **validation accuracy (~99.1%) but takes longer per step (~128ms)** compared to GRU-based models. While CNN is effective for **short-range dependencies and extracting local features**, it may struggle with longer textual sequences, leading to **slightly higher validation loss (~0.0538)** compared to RNN-based models.

The **Hybrid CNN + RNN model**, which combines **CNN for local feature extraction and Bi-GRU for sequence modeling**, achieves **strong validation accuracy (~99.3%) and the lowest validation loss (~0.0497)** among all models. This suggests that integrating CNN and RNN provides **the best of both worlds**, capturing both **local and long-range dependencies** effectively.

In conclusion, **if computational efficiency is a priority**, **Pretrained Word2Vec + GRU** is the best choice due to **fast training and strong generalization**. However, **for maximizing accuracy and feature extraction**, the **Hybrid CNN + RNN** model is the most effective, albeit slightly slower. The **Non-pretrained Word2Vec + GRU model** is beneficial when **fine-tuning embeddings** is necessary, but it comes at a high computational cost.

5. Discussion and Insights

5.1 In-depth Analysis

The results reveal an unexpected trend where the non-pretrained CNN and hybrid CNN-RNN models outperform the pretrained Word2Vec RNN model in terms of accuracy and validation loss. This suggests that CNN-based models are highly effective at capturing local text patterns and extracting discriminative features, even without pre-trained embeddings. The CNN layers in both models likely act as strong feature extractors, allowing them to learn robust token representations directly from the dataset rather than relying on pretrained embeddings that might not fully align with the toxic comment classification task.

5.2 Error Analysis

A key challenge observed in the pretrained Word2Vec + RNN model is its relatively higher validation loss despite its generalization ability. This indicates that the fixed pretrained embeddings may not perfectly align with the dataset vocabulary, causing inefficiencies in feature learning. In contrast, the CNN and CNN-RNN models appear to better adapt to the dataset, likely because their filters dynamically learn hierarchical features, reducing reliance on external embeddings and leading to improved accuracy. However, despite their superior performance, CNN-based models may still struggle with long-range dependencies, which could be mitigated by incorporating attention mechanisms.

6. Lessons Learned and Future Work

6.1 Key Takeaways

A major takeaway from this study is that pretrained embeddings are not always the optimal choice, especially when the dataset significantly differs from the corpus used to train them. The superior performance of non-pretrained CNN and CNN-RNN models suggests that learning embeddings from scratch on the target dataset can sometimes yield better results, particularly when the dataset is large enough to provide sufficient training signals. Additionally, CNNs are extremely efficient at capturing local patterns, making them well-suited for tasks like text classification.

6.2 Future Improvements

To further enhance performance, combining CNN feature extraction with Transformer-based architectures (such as BERT or attention mechanisms) could allow the model to leverage both local feature extraction and long-range dependencies. Additionally, experimenting with dynamic fine-tuning of pretrained embeddings rather than freezing them might allow the RNN model to learn representations that are more relevant to the classification task. Finally, using larger and more diverse datasets could help refine the learned embeddings and improve generalization across different toxic comment styles.