DEEP LEARNING FOR HATE SPEECH DETECTION

Betzalel Moskowitz
MSDS458-DL: Artificial Intelligence and Deep Learning
August 27, 2023

Abstract

Hate speech detection is crucial for maintaining a safe and inclusive online environment, countering the spread of prejudice, and preventing the escalation of online vitriol into real-world harm. This research paper delves into hate speech detection in online platforms, exploring the complexities and challenges of accurately identifying offensive content. Contrary to expectations, the study finds that bag-of-words (BoW) models outperform sequence models in this context. Through rigorous experiments, the study highlights the effectiveness of BoW models, particularly bag of bigrams with TF-IDF vectorization, the value of large and diverse datasets, and the effectiveness of CNN/RNN hybrid models, pretrained embeddings, and transformers. This work enriches understanding of hate speech detection, offering insights for future research and contributing to the quest for safer online environments.

Note: As this paper focuses on hate speech, the paper contains profanity and words that may be offensive or hurtful to certain individuals.

Introduction

In today's digital age, online platforms and social media have become prominent mediums for communication and information exchange. The rapid proliferation of these platforms has democratized speech, providing individuals with the ability to express their thoughts, opinions, and feelings more than ever before. However, this freedom of expression has also enabled an unfortunate rise in hate speech. Hate speech is characterized by discriminatory, offensive, or derogatory language targeting individuals or groups on the basis of race, religion, ethnicity, gender, or sexual orientation. It produces real-world harm, introduces division, and instigates violence, degrading the very platforms that seek to bring people together.

However, the rise of hate speech has far-reaching consequences that extend beyond the online realm. It contributes to the spread of prejudice, reinforces stereotypes, and creates hostile environments that deter individuals from participating in online discourse. When left unchecked, online hate speech has resulted in real-life harm, as individuals radicalized through online mediums incite offline violence or discrimination. In 2012, Wade Michael Page murdered six people at a Sikh temple in Wisconsin after posting in hateful online forums. Dylan Roof, the murderer of nine people at a South Carolina Black church was said to be "self-radicalized online" by prosecutors. Robert Bowers, the killer of 11 people at the Tree of Life Synagogue in 2018 had been active on a Twitter-like site used by white supremacists (Hatzipanagos, 2018).

Hate speech also affects business. After buying Twitter in October 2022, Elon Musk allowed anyone to be "verified" on Twitter as long as they paid a few. This offer was even available to groups whose accounts were previously removed for being associated with US-designated terrorist organizations and hate groups. Following Musk's acquisition of Twitter, daily slurs against Black Americans have tripled on Twitter. Slurs against gay men on the app had increased by 58%, and antisemitic posts soared by 61%. These trends did not go unnoticed by advertisers, who began to reduce spending on the app as they awaited the implications of Musk's acquisition of the company. These moves hurt the bottom line of the company, as 90% of Twitter's revenue are provided by advertisers (Frenkel & Conger, 2022).

As a result of the soaring rates of hate speech on online platforms, the development of effective hate speech detection mechanisms is crucial to mitigate these negative outcomes. Automated hate speech detection systems offer several benefits. They enable platforms to maintain a healthier and more inclusive online environment by swiftly identifying and removing harmful content, promoting positive interactions and encouraging meaningful conversations. These systems also provide a safeguard against the inadvertent normalization of hate speech, which have shown to occur when such content goes unaddressed. Additionally, by identifying patterns and trends in hate speech, these systems can provide valuable insights into societal attitudes and biases, aiding researchers and policymakers in addressing systemic issues.

Hate speech detection is a complex task – it involves navigating the nuances of language, context, and cultural variations, which can pose challenges for automated systems. Striking the right balance between protecting freedom of expression and curbing hate speech requires ongoing refinement of these systems. This underscores the importance of interdisciplinary collaboration between linguists, ethicists, technologists, and community members to ensure that hate speech detection tools are accurate, fair, and respectful of diverse perspectives.

This paper offers insights into the construction of effective hate speech detectors by examining different model architectures, transfer learning techniques, and regularization methods conducted to develop a hate speech classifier.

Literature Review

Hate speech detection has gained substantial attention due to its significance in maintaining inclusive online environments and combating the spread of harmful content. While there are many datasets available for hate speech detection, it is difficult to benchmark hate speech detector models against each other, as many are trained on different datasets with slightly nuanced tasks. Papers with Code, a website that tracks performance benchmarks on machine learning tasks, lists 18 English datasets for hate speech detection (Papers with Code 2023).

“Hate Speech Dataset”, proposed by De Gibert et al. (2018), contained over 10,000 hand-annotated samples of hateful and benign text extracted from sentences on Stormfront, an online white supremacist forum. An evenly balanced subset of the data (2,000 total samples) was taken and split into 80% train and 20% test to train several binary classifiers: a Support Vector Machine (SVM) over Bag-Of-Words vectors, a one-dimensional Convolutional Neural Network (CNN) using sequence embeddings, and a Recurrent Neural Network (RNN) with a sequence word embedding of size 300 and a Long Short-term Memory (LSTM) layer of size 128. Without hyperparameter tuning, the SVM, CNN, and LSTM models yielded 74%, 70%, and 78% test accuracies respectively, demonstrating the impressive performance of the SVM bag-of-words approach compared to the CNN and LSTM sequence models (De Gibert et al. 2018).

The best performing model on this dataset (hate_speech18 dataset) was deberta-v3-small-finetuned-hate_speech18, a DeBERTa-v3-small transformer model fine-tuned on the hate_speech18 dataset. The model achieved a test accuracy of 91.61%, and is available on Hugging Face via PyTorch (Hugging Face, 2023).

Other datasets like HateXplain (2021) were created in attempt to produce hate speech detection models with better interpretability and reduced bias. The researchers produced a dataset consisting of 20,000 online posts, with each sample assigned a three-class label ('hate', 'offensive', 'normal'), the community victimized by the post, and the rationale behind the label (the portion of the post that is hateful). The researchers discovered that even state of the art-models performing well in classification tasks struggle on metrics of explainability. The researchers experimented with CNN-GRUs, Bidirectional RNNs (BiRNNs), BiRNNs with Attention, and BERT models. They found that they were able to attain an AUC/ROC score of 85.1% and achieved strong results in bias and explainability using their new dataset (Matthew et al., 2021).

The hybrid CNN/LSTM architecture proposed by Zhang et al. (2018) provided a sophisticated approach to multi-class hate speech modeling, exploiting the strengths of both CNNs and LSTMs to create a more comprehensive and effective model for analyzing text-based data. Their model takes advantage of the CNN's ability to identify important local features and the LSTM's capability to capture contextual information and dependencies between words. This combination enhances the model's capacity to comprehend both fine-grained linguistic nuances and broader semantic context in hate speech detection tasks (Zhang et al., 2018).

While generating a multi-class dataset with labels of normal, hate speech, or offensive language provides further explainability, it can be quite challenging when one cannot rely on heuristic approaches. For instance, it may be feasible to distinguish offensive language from normal language as language often contains offensive words, but the same is not always true for hate speech. For these reasons, multi-class hate speech datasets are challenging and expensive to produce. Additionally, they require human annotators to make subjective judgements on posts, introducing bias and noise into the data.

For these reasons, this paper will place its emphasis on the binary classification of hate speech versus regular speech, where the category of hate speech encompasses both offensive language and discriminatory content directed towards specific individuals or groups. This strategic choice is underpinned by its notable advantages, primarily stemming from the ability to side-step the complexities inherent in multi-class labeling across diverse forms of hate speech.

By adopting a binary classification framework, this paper capitalizes on several significant benefits. Chief among them is the capacity to curate a considerably larger and more diverse training dataset. This approach draws from a wide spectrum of existing hate speech datasets, regardless of the distinct categories of hate speech they encompass. Consequently, the research avoids the intricacies and potential inconsistencies arising from dealing with multi-class labels representing varying types of hate speech.

This binary classification strategy focuses on the overarching dichotomy between hate speech and non-hateful content, providing a clearer demarcation instrumental in training accurate and robust models. This approach optimizes the use of available data resources, streamlines the research process, and underscores the significance of addressing the broader issue of hate speech in online platforms.

Methods

Dataset

The dataset used in this paper – "A curated dataset for hate speech detection on social media text" – was developed by Mody et al. (2023) with the intent to identify hate speech on text from social media. It contains 451,709 hate speech and non-hate speech sentences in English and is labeled with two classes – hateful content (1) and non-hateful content (0). The dataset was sourced from Kaggle, GitHub, and other existing hate speech datasets, and includes valuable data not always present in other datasets, including emojis, emoticons, slang, and contractions (Mody et al., 2023).

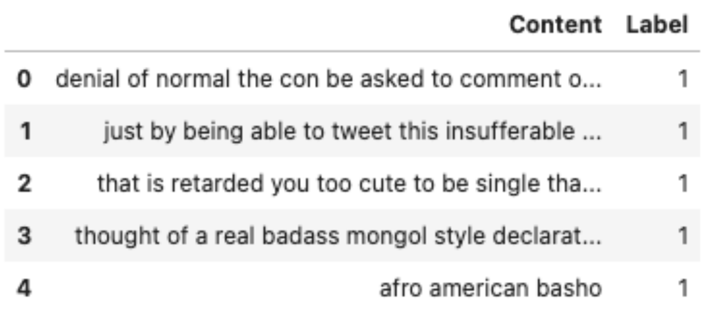
The dataset was preprocessed using the following steps. First, any emojis or emoticons from the raw data were converted to text form. Any hyperlinks, user mentions, multiple spaces, and new line characters were removed. Any contractions in the dataset were expanded, and the researchers used the Google News Word2Vectors and an open-source Gensim model to correct grammatical errors. Next, all date and time occurrences were removed, as well as accented numbers and encoded characters beginning with ampersands. Any remaining numbers were converted into word form, and special characters were removed. The researchers then used a whitelist of profanities to correct any misspelled profanity words, and any duplicate sentences were removed (Mody et al., 2023).

The resulting dataset is imbalanced – 371,452 of the sentences are non-hateful speech, while the other 80,250 are hateful. To generate a balanced dataset, the researchers undersampled the majority class (non-hateful) and augmented the sentences from the hateful class using "contextual word embeddings from BERT models with substitution and insertion methods as well as the synonym augmentation using WordNet embeddings" (Mody et al., 2023).

The augmented dataset contains 726,119 samples (361,594 non-hateful vs 364,525 hateful sentences) providing an alternative to smaller datasets that are highly specialized, allowing this dataset to generalize and serve as a generic benchmark for natural language processing models.

Several rows of the dataset can be seen below in *Figure 1*. Please be advised that there may be profanity or offensive words present as this is a hate speech dataset.

Figure 1. Example Rows of the Hate Speech Dataset



	Content	Label
0	denial of normal the con be asked to comment o...	1
1	just by being able to tweet this insufferable ...	1
2	that is retarded you too cute to be single tha...	1
3	thought of a real badass mongol style declarat...	1
4	afro american basho	1

Since deep learning techniques require text data to be represented as numeric data, text vectorization must be performed to prepare the data for modeling. Vectorizing the entire dataset each time a new experiment is run in a new Google Colab notebook is extremely time-consuming, so a 50% subset of the full dataset was used for the training experiments conducted in this paper. The subset selected for the dataset were stratified-sampled to preserve the roughly balanced nature of the class distribution. The subset of the dataset contained 180,797 non-hateful samples as well as 182,262 hateful samples. The subset was split into 80% train, 10% validation, and 10% test using stratified methods. The class distribution of the training, validation, and test sets are visualized in *Figure 2*.

Figure 2. Class Label Distribution of Train, Validation, and Test Sets

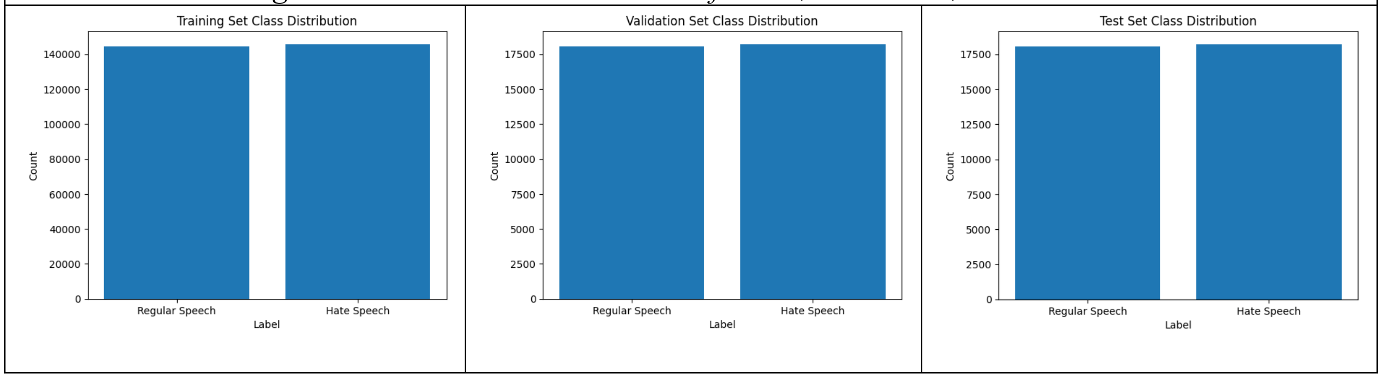
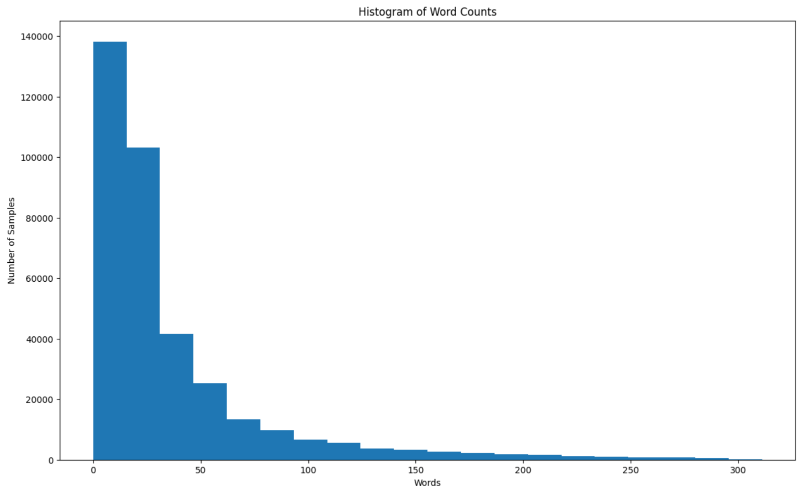


Figure 3 shows that the mean and median sentence lengths of the dataset subset are 36.3 and 21 respectively, indicating that it might be possible to reduce the maximum length of the sentence and still preserve much of the information present. The distribution of word counts is visualized in *Figure* 4, where the histogram of word counts per sample reveals a long right-tail, indicating that most of the samples have sentence lengths of less than 100 with outliers. The dataset subset had 108,505 unique vocabulary words.

Figure 3: Descriptive Statistics of Tokens Per Sample

Mean	36.3
Median	21
Standard Deviation	44.1
Minimum	1
Maximum	310

Figure 4: Histogram of Words Per Sample



In many classification tasks such as topic classification, one might decide to reduce the max length since the majority of the samples' information is contained in these smaller lengths, making it easy to distinguish between classes in a lower dimension. However, hate speech detection presents a more challenging problem, where the difference between hateful speech and non-hateful speech might only be distinguishable from the last few words in a longer sequence, making the last words of larger texts important to preserve.

For these reasons, a maximum length of 300 and a vocabulary size of 20,000 tokens were initially selected for text vectorization in an attempt to preserve as much information as possible from each of the samples.

After splitting, the datasets had any remaining punctuation or stop words removed, and the datasets were each vectorized using a vectorizer adapted from only the training set to eliminate leakage of validation and test data into the training process.

Word Clouds were also generated providing a quick visual summary of the most common words within each class. The larger the word, the more often it is used. These Word Clouds are available in *Figures 5 and 6*. Please be advised that there are profanity and offensive words present in these visuals.

Figure 5: Word Cloud of Non-Hate Speech

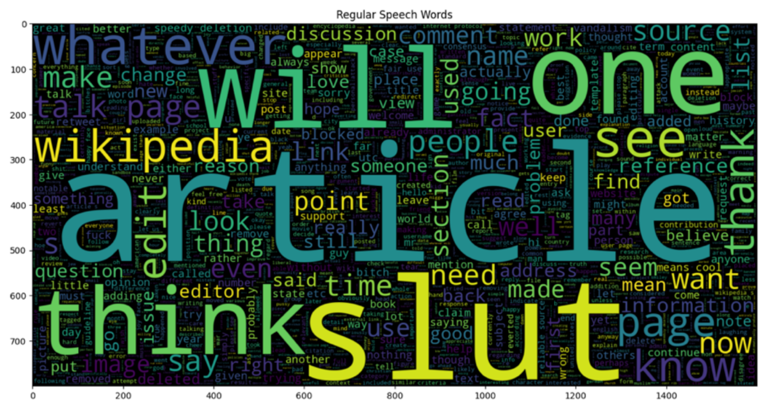
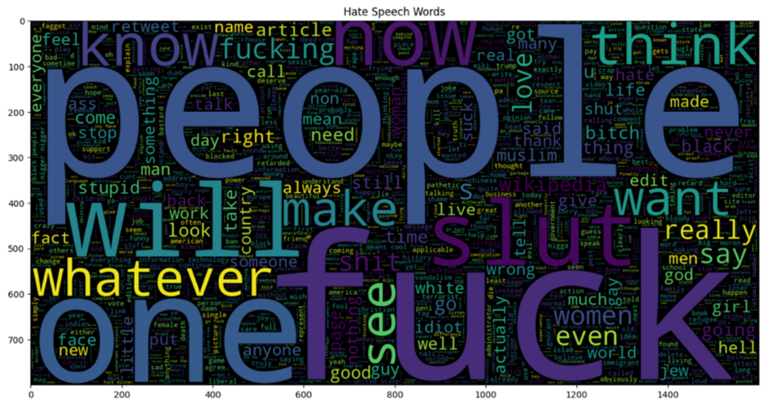


Figure 6: Word Cloud of Hate Speech



Implementation and Programming

Because the problem is binary classification and the dataset is almost equally balanced, a benchmark of roughly 50% test accuracy was used to benchmark the experiments, since one would be correct roughly 50% of the time when guessing the class at random. The goal is to beat this benchmark to produce a model capable of predicting hate speech at a rate higher than random chance.

The experiments were carried out via Google Collaboratory notebooks, utilizing Google's A100 GPUs on a hosted Python runtime. All modeling was conducted using Keras within TensorFlow (version 2.12). All visualizations were created using Matplotlib or Seaborn. Additional Python libraries used included the Pandas and NumPy libraries for data manipulation.

The experiments were conducted in an attempt to answer the following questions:

- How does the vocabulary size affect the performance of the model?
- Do sequence models perform better than bag of words approaches?
- Do certain sequence model architectures (LSTM, GRUs, 1D CNN, CNN/RNN Hybrid, Pretrained Embeddings, Transformers) perform better than others for hate speech detection?
- Can using pretrained embeddings or fine-tuning pretrained embeddings help improve performance?
- How effective are regularization techniques at reducing overfitting?

Due to time constraints, some hyperparameters were not tuned and remained fixed throughout all experiments. ReLU and RMSProp were chosen as the activation function and optimizer respectively. Since the labels were binary, binary cross-entropy was selected for the loss function. Each experiment used a batch size of 128 and with early stopping (patience=2). Early stopping occurred if the validation accuracy did not improve after two epochs, preserving the best model and avoiding additional overfitting.

This study focused on training two families of models – sequence models and bag of words (BoW) models.

Sequence Models

For sequence models, a Bidirectional LSTM model with a word embedding layer, Bidirectional LSTM layer with 64 Units, 50% Dropout, and a dense output layer was trained with a vocabulary size of 20,000 and a max length of 300. This model was repeated to determine the effectiveness of a reduced max length of 75. With the 300-max length model performing better, this length was used for the remainder of the experiments. The same architecture was repeated for vocabulary sizes of 10,000, 50,000, and 100,000. The 50,000-vocabulary size performed best, so this vocabulary size was also used for the remainder of the experiments.

With the vocabulary size and the max length set, the following models were trained:

- Word Embedding, Bidirectional LSTM with 64 Units, 50% Dropout, 64 Unit Dense Layer, 50% Dropout, Dense Output Layer

- Word Embedding, Bidirectional GRU with 64 Units, 50% Dropout, 64 Unit Dense Layer, 50% Dropout, Dense Output Layer
- Word Embedding, 1D CNN with 128 Filters, 50% Dropout, Global Average Pooling, 64 Unit Dense Layer, 50% Dropout, Output Dense Layer
- Word Embedding, 1D CNN with 256 Filters, 50% Dropout, Global Average Pooling, 64 Unit Dense Layer, 50% Dropout, Output Dense Layer

Inspired by Zhang et al. (2018), the following CNN/RNN models were also trained:

- Word Embedding, 50% Dropout, 1D CNN with 100 filters kernel size 4 padding same, MaxPooling 1D with Pool Size 4, Bidirectional LSTM with 100 units, Global Max Pooling 1D, 64 Unit Dense Layer, 50% Dropout, Output Dense Layer
- Word Embedding, 50% Dropout, 1D CNN with 100 filters kernel size 4 padding same, MaxPooling 1D with Pool Size 4, LSTM with 100 units, Global Max Pooling 1D, 64 Unit Dense Layer, 50% Dropout, Output Dense Layer
- Word Embedding, 50% Dropout, 1D CNN with 100 filters kernel size 4 padding same, MaxPooling 1D with Pool Size 4, Bidirectional GRU with 100 units, Global Max Pooling 1D, 64 Unit Dense Layer, 50% Dropout, Output Dense Layer
- Word Embedding, 50% Dropout, 1D CNN with 100 filters kernel size 4 padding same, MaxPooling 1D with Pool Size 4, GRU with 100 units, Global Max Pooling 1D, 64 Unit Dense Layer, 50% Dropout, Output Dense Layer

Several Transformer models were trained as well, using Positional Embeddings and a Transformer Encoder with different numbers of heads:

- Positional Embedding, Transformer Encoder with 2 heads, Global Max Pooling, 50% Dropout, Output Dense Layer
- Positional Embedding, Transformer Encoder with 4 heads, Global Max Pooling, 50% Dropout, Output Dense Layer

Out of these models, the hybrid CNN + BiLSTM architecture worked the best, so that architecture was used to experiment with pretrained GloVe Embeddings (Pennington et al., 2014). One of these models used frozen GloVe embeddings (trainable=False) while the other model finetuned these embeddings. The finetuned model's embedding layer was significantly stronger than any of the other embeddings trained previously, so it was used as the embedding layer to another transformer model with a Transformer Encoder, 50% Dropout, and a dense output layer.

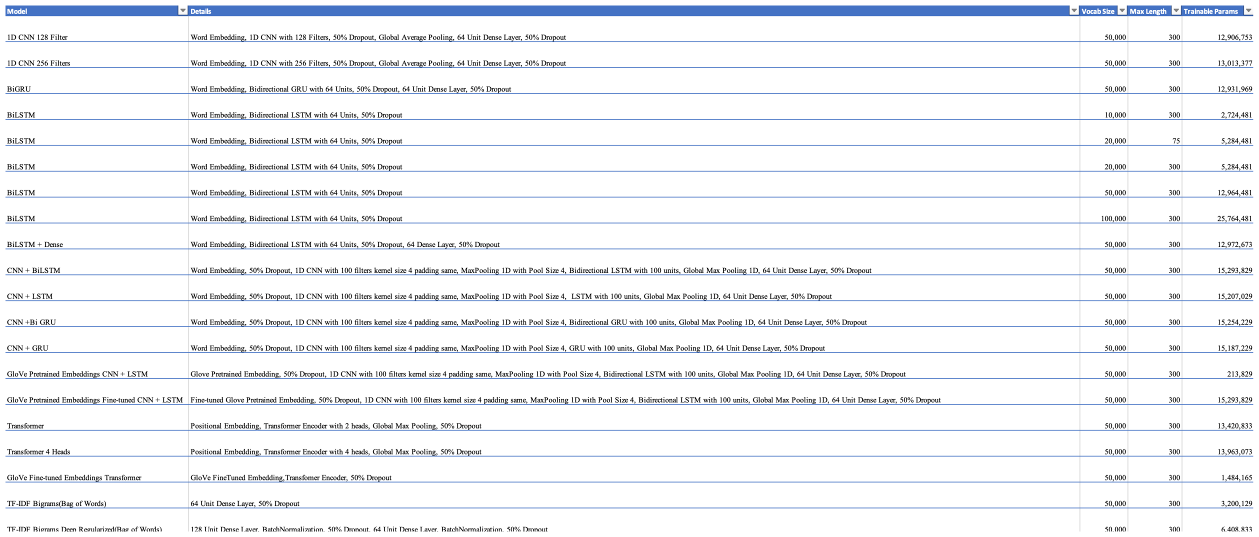
Bag of Words (BoW) Models

Bag of Words models have been shown to remain effective for NLP tasks despite its simplicity and age. The data for these models were vectorized as bags of bigrams using TF-IDF. Two models were trained:

- 64 Unit Dense Layer, 50% Dropout, and a Dense Output Layer
- 128 Unit Dense Layer, Batch Normalization, 50% Dropout, 64 Unit Dense Layer, Batch Normalization, 50% Dropout, Dense Output Layer

Figure 7 contains a breakdown of all experiments conducted. For each of the experiments run, performance metrics and a confusion matrix were recorded to examine training and performance.

Figure 7: Description of Experiments

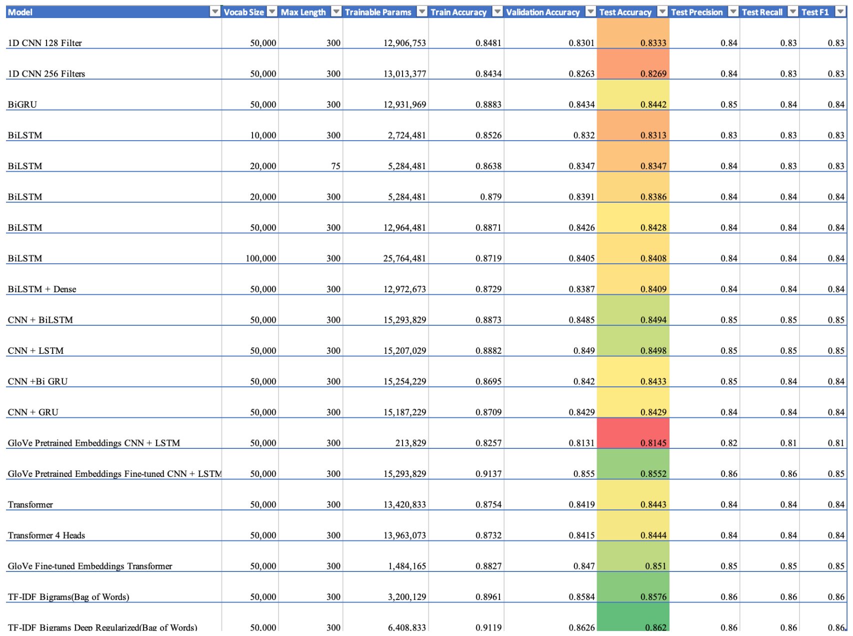


Model	Details	Vocab Size	Max Length	Trainable Params
1D CNN 128 Filter	Word Embedding, 1D CNN with 128 Filters, 50% Dropout, Global Average Pooling, 64 Unit Dense Layer, 50% Dropout	50,000	300	12,906,753
1D CNN 256 Filters	Word Embedding, 1D CNN with 256 Filters, 50% Dropout, Global Average Pooling, 64 Unit Dense Layer, 50% Dropout	50,000	300	13,013,377
BiGRU	Word Embedding, Bidirectional GRU with 64 Units, 50% Dropout, 64 Unit Dense Layer, 50% Dropout	50,000	300	12,931,969
BiLSTM	Word Embedding, Bidirectional LSTM with 64 Units, 50% Dropout	10,000	300	2,724,481
BiLSTM	Word Embedding, Bidirectional LSTM with 64 Units, 50% Dropout	20,000	75	5,284,481
BiLSTM	Word Embedding, Bidirectional LSTM with 64 Units, 50% Dropout	20,000	300	5,284,481
BiLSTM	Word Embedding, Bidirectional LSTM with 64 Units, 50% Dropout	50,000	300	12,964,481
BiLSTM	Word Embedding, Bidirectional LSTM with 64 Units, 50% Dropout	100,000	300	25,764,481
BiLSTM + Dense	Word Embedding, Bidirectional LSTM with 64 Units, 50% Dropout, 64 Dense Layer, 50% Dropout	50,000	300	12,972,673
CNN + BiLSTM	Word Embedding, 50% Dropout, 1D CNN with 100 filters kernel size 4 padding same, MaxPooling 1D with Pool Size 4, Bidirectional LSTM with 100 units, Global Max Pooling 1D, 64 Unit Dense Layer, 50% Dropout	50,000	300	15,293,829
CNN + LSTM	Word Embedding, 50% Dropout, 1D CNN with 100 filters kernel size 4 padding same, MaxPooling 1D with Pool Size 4, LSTM with 100 units, Global Max Pooling 1D, 64 Unit Dense Layer, 50% Dropout	50,000	300	15,207,029
CNN +Bi GRU	Word Embedding, 50% Dropout, 1D CNN with 100 filters kernel size 4 padding same, MaxPooling 1D with Pool Size 4, Bidirectional GRU with 100 units, Global Max Pooling 1D, 64 Unit Dense Layer, 50% Dropout	50,000	300	15,254,229
CNN + GRU	Word Embedding, 50% Dropout, 1D CNN with 100 filters kernel size 4 padding same, MaxPooling 1D with Pool Size 4, GRU with 100 units, Global Max Pooling 1D, 64 Unit Dense Layer, 50% Dropout	50,000	300	15,187,229
GloVe Pretrained Embeddings CNN + LSTM	Glove Pretrained Embedding, 50% Dropout, 1D CNN with 100 filters kernel size 4 padding same, MaxPooling 1D with Pool Size 4, Bidirectional LSTM with 100 units, Global Max Pooling 1D, 64 Unit Dense Layer, 50% Dropout	50,000	300	213,829
GloVe Pretrained Embeddings Fine-tuned CNN + LSTM	Fine-tuned Glove Pretrained Embedding, 50% Dropout, 1D CNN with 100 filters kernel size 4 padding same, MaxPooling 1D with Pool Size 4, Bidirectional LSTM with 100 units, Global Max Pooling 1D, 64 Unit Dense Layer, 50% Dropout	50,000	300	15,293,829
Transformer	Positional Embedding, Transformer Encoder with 2 heads, Global Max Pooling, 50% Dropout	50,000	300	13,420,833
Transformer 4 Heads	Positional Embedding, Transformer Encoder with 4 heads, Global Max Pooling, 50% Dropout	50,000	300	13,963,073
GloVe Fine-tuned Embeddings Transformer	GloVe FineTuned Embedding,Transfomer Encoder, 50% Dropout	50,000	300	1,484,165
TF-IDF Bigrams(Bag of Words)	64 Unit Dense Layer, 50% Dropout	50,000	300	3,200,129
TF-IDF Bigrams Deep Regularized(Bag of Words)	128 Unit Dense Layer, BatchNormalization, 50% Dropout, 64 Unit Dense Layer, BatchNormalization, 50% Dropout	50,000	300	6,408,833

Results

Figure 8 provides the results for the experiments. A full version is available in Appendix A1.

Figure 8: Experiment Results



Model	Vocab Size	Max Length	Trainable Params	Train Accuracy	Validation Accuracy	Test Accuracy	Test Precision	Test Recall	Test F1
1D CNN 128 Filter	50,000	300	12,906,753	0.8481	0.8301	0.8333	0.84	0.83	0.83
1D CNN 256 Filters	50,000	300	13,013,377	0.8434	0.8263	0.8269	0.84	0.83	0.83
BiGRU	50,000	300	12,931,969	0.8883	0.8434	0.8442	0.85	0.84	0.84
BiLSTM	10,000	300	2,724,481	0.8526	0.832	0.8313	0.83	0.83	0.83
BiLSTM	20,000	75	5,284,481	0.8638	0.8347	0.8347	0.84	0.83	0.83
BiLSTM	20,000	300	5,284,481	0.879	0.8391	0.8386	0.84	0.84	0.84
BiLSTM	50,000	300	12,964,481	0.8871	0.8426	0.8428	0.84	0.84	0.84
BiLSTM	100,000	300	25,764,481	0.8719	0.8405	0.8408	0.84	0.84	0.84
BiLSTM + Dense	50,000	300	12,972,673	0.8729	0.8387	0.8409	0.84	0.84	0.84
CNN + BiLSTM	50,000	300	15,293,829	0.8873	0.8485	0.8494	0.85	0.85	0.85
CNN + LSTM	50,000	300	15,207,029	0.8882	0.849	0.8498	0.85	0.85	0.85
CNN +Bi GRU	50,000	300	15,254,229	0.8695	0.842	0.8433	0.85	0.84	0.84
CNN + GRU	50,000	300	15,187,229	0.8709	0.8429	0.8429	0.84	0.84	0.84
GloVe Pretrained Embeddings CNN + LSTM	50,000	300	213,829	0.8257	0.8131	0.8145	0.82	0.81	0.81
GloVe Pretrained Embeddings Fine-tuned CNN + LSTM	50,000	300	15,293,829	0.9137	0.855	0.8552	0.86	0.86	0.85
Transformer	50,000	300	13,420,833	0.8754	0.8419	0.8443	0.84	0.84	0.84
Transformer 4 Heads	50,000	300	13,963,073	0.8732	0.8415	0.8444	0.84	0.84	0.84
GloVe Fine-tuned Embeddings Transformer	50,000	300	1,484,165	0.8827	0.847	0.851	0.85	0.85	0.85
TF-IDF Bigrams(Bag of Words)	50,000	300	3,200,129	0.8961	0.8584	0.8576	0.86	0.86	0.86
TF-IDF Bigrams Deep Regularized(Bag of Words)	50,000	300	6,408,833	0.9119	0.8626	0.862	0.86	0.86	0.86

Vocabulary Sizes

While it would be best to have tested different vocabulary sizes for each model in addition to the Bidirectional LSTMs (BiLSTM), we can see from the BiLSTM that the test accuracy increased as the size of the vocabulary size increased, up until 50,000, after which the performance diminished when vocabulary size 100,000 was tested. This boost in performance is very slight and may be due to random chance or luck with random initialization of weights. Proper ablation and hyper-parameter tuning should be conducted on this dataset in the future to validate this finding.

If the vocabulary size findings can be replicated, the decrease in performance at lower vocabulary sizes is likely because limiting the vocabulary size too much may result in many words being treated as out-of-vocabulary (OOV) tokens. By increasing the vocabulary size, however, more word variations were captured, providing the model with richer information. However, it is worth noting that using all 108,505 vocabulary words may add noise and may not necessarily provide features that are useful – this could explain what is seen from the drop-off at higher values than vocabulary sizes of 50,000.

Sequence Models vs Bag of Words (BoW) Performance

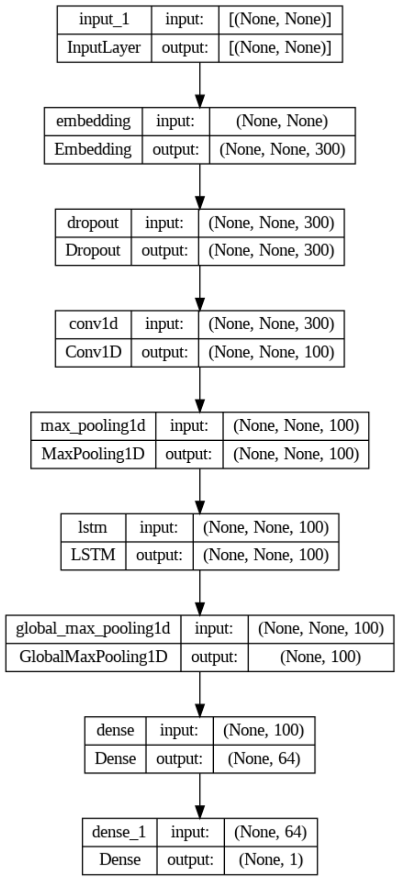
All models trained performed somewhere in the range of 81% test accuracy to just above 86% accuracy. Overall, the two BoW models outperformed any of the sequence models. This was rather surprising – a general rule-of-thumb provided by Chollet (2022) has shown that sequence models are more performant in text classification tasks when the ratio of the number of samples to mean sequence lengths is greater than 1,500. In the training split of the subset dataset there are 290,447 samples with a mean words per sample of 18.8. This equates to a ratio of almost 15,451, suggesting that the sequence models should perform better in this context. It is possible that using the entire dataset may yield different results, as sequence models are very data-hungry and work best with larger amounts of data. This should be explored in future studies.

Sequence Model Performance

All of the pure RNN models (Bidirectional LSTMs, Bidirectional GRUs) achieved test accuracies in the 83% to low 84% range. The GRU model (test accuracy of 84.42%) on its own outperformed any of the vanilla LSTMs. The two 1D CNNs performed worse than the RNNs, with the 256-filter model achieving a test accuracy of 82.69% and the 128-filter model achieving a test accuracy of 83.33%.

However, a breakthrough in performance occurred when the CNN and RNN models were combined into hybrid models. While the 1D CNN + GRU-based models only achieved 84.29% and 84.33% test accuracies, the LSTM versions of these models were significantly better at 84.94% and 84.98% test accuracies. In these models, the Bidirectional nature of RNN layers did not make a significant difference in performance. However, the CNN + LSTM models were the best models achieved at the time and inspired experimentation with using pretrained-embeddings within the architecture. *Figure 9* displays the architecture of the CNN + LSTM hybrid model.

Figure 9: Architecture of CNN + LSTM Model


input_1 | input: | [(None, None)]
InputLayer | output: | [(None, None)]
embedding | input: | (None, None)
Embedding | output: | (None, None, 300)
dropout | input: | (None, None, 300)
Dropout | output: | (None, None, 300)
conv1d | input: | (None, None, 300)
Conv1D | output: | (None, None, 100)
max_pooling1d | input: | (None, None, 100)
MaxPooling1D | output: | (None, None, 100)
lstm | input: | (None, None, 100)
LSTM | output: | (None, None, 100)
global_max_pooling1d | input: | (None, None, 100)
GlobalMaxPooling1D | output: | (None, 100)
dense | input: | (None, 100)
Dense | output: | (None, 64)
dense_1 | input: | (None, 64)
Dense | output: | (None, 1)

Pretrained Embeddings with GloVe

The CNN + LSTM model trained with the frozen pretrained GloVe embeddings performed the poorest out of all models, achieving a test accuracy of 81.45%. This is likely because the embedding space was not allowed to adjust to the new task and underfit. When the same model was trained but allowed to fine-tune its embedding layer, it yielded the highest test accuracy seen across all sequence models at 85.52%. This demonstrated the power of transfer learning, using weights learned on an adjacent task with large amounts of data and allowing the weights to be tweaked slightly to adapt well to a new task.

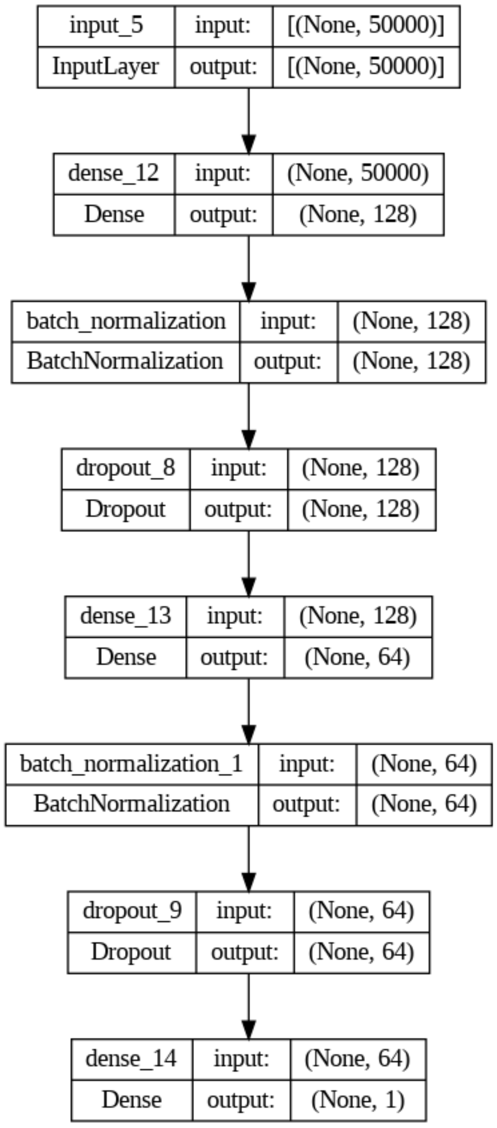
Transformer Models

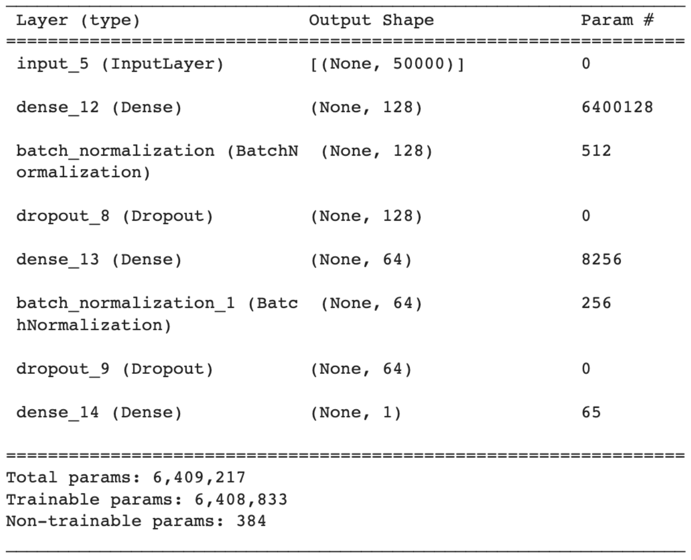
The two Transformer models were trained with two heads and four heads, achieving a test accuracy of 84.43% and 84.44% respectively. This was particularly underwhelming, especially given the hypothesis that self-attention might lend itself well to this task by focusing on small pieces of text that are highly predictive of hateful content. That said, the Transformer model with four heads whose positional embedding layer was switched out for a GloVe embedding layer achieved a test accuracy of 85.1%. The previous knowledge gained by the GloVe embedding on massive amounts of data helped the performance of the Transformer model, suggesting that using the rest of the dataset might help improve performance further. This should be investigated in future work.

Bag of Bigrams Models

The Bag of Bigrams models were surprisingly performant. This is likely due to hateful speech containing offensive keywords, which bag of words models excel on picking up on. The basic model, with a 64-unit dense layer and a 50% dropout layer produced a test accuracy of 85.76%. The deeper model with a 128-unit dense layer, batch normalization, 50% dropout, 64-unit dense layer, batch normalization, and 50% dropout produced the best model with a test accuracy of 86.2%, with precision, recall, and F1 scores of 86%. The model architecture, confusion matrix, and training progress can be seen in *Figures 10, 11, and 12.*

Figure 10: Best Model Architecture


input_5 | input: | [(None, 50000)]
InputLayer | output: | [(None, 50000)]
dense_12 | input: | (None, 50000)
Dense | output: | (None, 128)
batch_normalization | input: | (None, 128)
BatchNormalization | output: | (None, 128)
dropout_8 | input: | (None, 128)
Dropout | output: | (None, 128)
dense_13 | input: | (None, 128)
Dense | output: | (None, 64)
batch_normalization_1 | input: | (None, 64)
BatchNormalization | output: | (None, 64)
dropout_9 | input: | (None, 64)
Dropout | output: | (None, 64)
dense_14 | input: | (None, 64)
Dense | output: | (None, 1)



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_5 (InputLayer)        [(None, 50000)]           0

 dense_12 (Dense)            (None, 128)               6400128

 batch_normalization (BatchN  (None, 128)              512
 ormalization)

 dropout_8 (Dropout)         (None, 128)               0

 dense_13 (Dense)            (None, 64)                8256

 batch_normalization_1 (Batc  (None, 64)               256
 hNormalization)

 dropout_9 (Dropout)         (None, 64)                0

 dense_14 (Dense)            (None, 1)                 65

=================================================================
Total params: 6,409,217
Trainable params: 6,408,833
Non-trainable params: 384
_________________________________________________________________
```

Figure 11: Best Model Training/Validation Accuracy and Loss

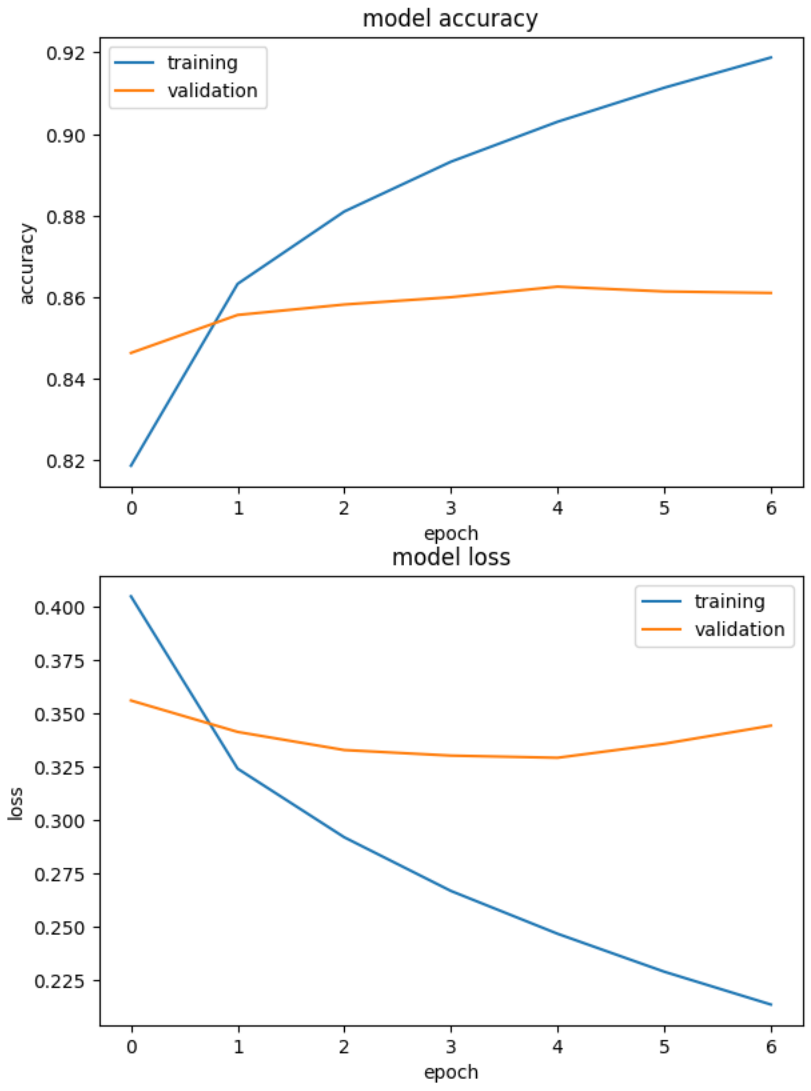
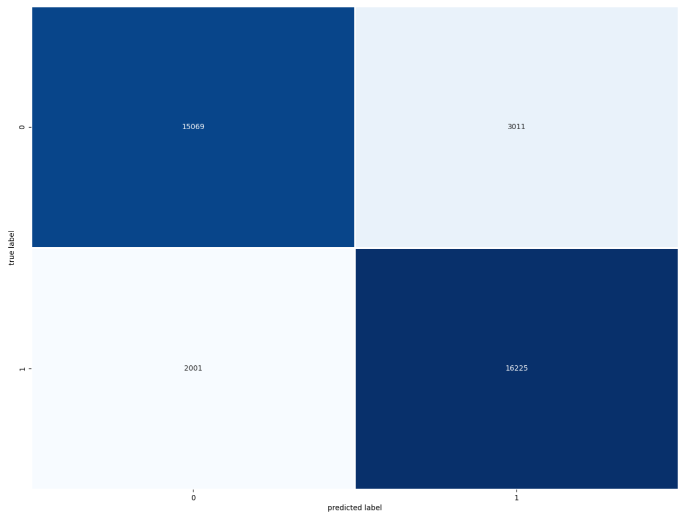


Figure 12: Best Model Confusion Matrix



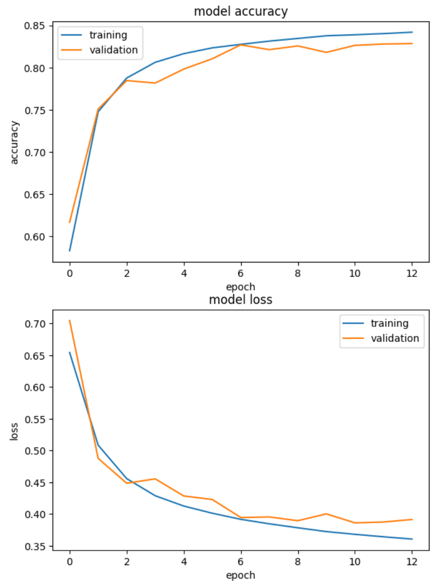
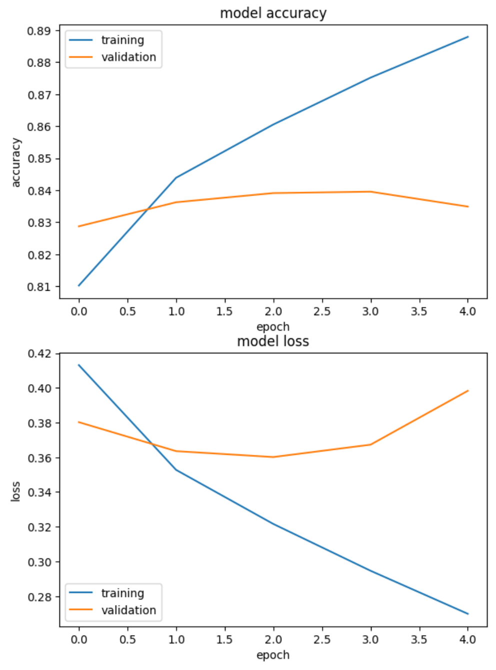
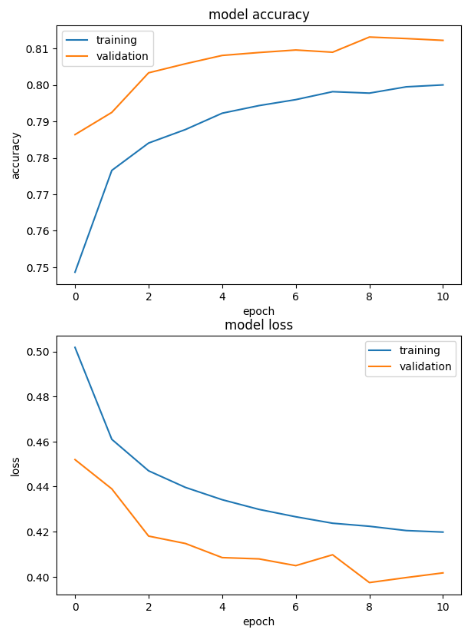
Regularization and Overfitting

These experiments saw all three types of model fitting – good fits, overfitting, and underfitting. They can be visualized in *Figure 13*.

The vanilla 1D CNNs experienced very little overfitting, although these models did not perform particularly well relative to others. The pretrained embedding model underfit as it only had 213,829 trainable parameters. This suggests that the model was not complex enough to learn the training data, and was improved by making the embedding layer trainable, increasing the parameter count of the model by 15 million parameters. The model then performed better despite eventually overfitting.

Because most of the other models had millions of parameters, they began to overfit quickly, despite each employing 50% dropout layers and global average pooling layers to fight overfitting. Overfitting was not necessarily detrimental as long as the validation accuracy and loss continued to increase and decrease respectively, but often times the training and validation curves would diverge from each other, suggesting that the model was no longer learning useful patterns that are generalizable. In these cases, early stopping with a small patience of two was the only remedy to divergent overfitting – the models had saved a checkpoint at the best training point and could be reverted back to that state following overfitting. Batch normalization and L2 normalization were experimented with, but they seemed to not make a difference in preventing overfitting or hurt learning altogether.

Figure 13. Different Fits Across Architectures

Good Fit 1D CNN	**Overfitting** RNN Models	**Underfitting** Pretrained Embeddings CNN/LSTM
		

Future Work

While solid performance was achieved using half of the full dataset, additional performance may be achieved from training on the entire dataset. This would have been experimented with next if time permitted, as more diverse data is ultimately the key to better generalization. Also due to time constraints, this study did not have the opportunity to perform hyperparameter tuning, which could boost model performance. Because this study was conducted using Keras within TensorFlow, several pretrained models from Hugging Face only supported in PyTorch were unavailable for transfer learning. A PyTorch implementation of this study could benefit from the increased flexibility to try other pretrained models. Additional experimentation should also be done to assess the effectiveness of fine-tuning BERT on the dataset, testing more values for max sequence lengths, and assessing the ethical considerations of deploying a model to a production environment capable of censoring content that may or may not be hateful. The models should also be tested out on other hate-speech datasets to benchmark against other models, as performance can only be compared when sizing up on the same datasets.

Conclusion

In conclusion, hate speech detection is a critical issue in the context of today's digital age marked by widespread online communication. Characterized by offensive language targeting specific groups, hate speech poses serious challenges to maintaining inclusive and positive online environments. The rise of hate speech has significant implications beyond the virtual realm, influencing societal attitudes and fostering real-world harm. The detrimental impact of hate speech is evidenced by instances such as online radicalization leading to offline violence.

This study underscores the necessity of robust hate speech detection mechanisms to counter these negative outcomes. Automated detection systems hold the potential to swiftly identify and remove harmful content, promoting healthier interactions and discouraging the normalization of hate speech. The paper systematically explores various model architectures, transfer learning methods, and regularization techniques to construct effective hate speech classifiers.

Despite theoretical expectations favoring sequence models, the results demonstrated that bag-of-words (BoW) models surprisingly outperformed sequence models in hate speech detection tasks. This unexpected outcome might be attributed to factors such as data quality, feature extraction, task complexity, model configuration, and more.

Through a comprehensive array of experiments involving different model architectures and approaches, the study reveals that BoW models, particularly those utilizing bag of bigrams with TF-IDF vectorization, exhibit robust performance in hate speech detection. These models benefit from the discriminatory and offensive language characteristic of hate speech, effectively distinguishing between hateful and non-hateful content. Furthermore, the paper explores the potential of hybrid models combining convolutional neural networks (CNNs) and recurrent neural networks (RNNs), as well as the impact of pretrained embeddings and transformer architectures.

Despite achieving promising results, avenues for future research and development are acknowledged. This includes the exploration of hyperparameter tuning, training on larger datasets, the deployment of models in real-world scenarios, and the consideration of ethical implications tied to content censorship. Overall, this study advances understanding of hate speech detection and provides valuable insights into constructing effective hate speech classifiers, contributing to the ongoing efforts to create safer and more inclusive online environments.

References

Chollet, F. (2022). Deep learning for text. In *Deep learning with python, second edition* (p. 349). Manning Publications.

De Gibert, O., Perez, N., García-Pablos, A., & Cuadros, M. (2018). Hate speech dataset from a white supremacy forum. arXiv preprint arXiv:1809.04444.

Frenkel, S., & Conger, K. (2022, December 2). *Hate speech's rise on Twitter is unprecedented, researchers find*. The New York Times. https://www.nytimes.com/2022/12/02/technology/twitter-hate-speech.html

Hatzipanagos, R. (2018, November 30). *Perspective | how online hate turns into real-life violence*. The Washington Post. https://www.washingtonpost.com/nation/2018/11/30/how-online-hate-speech-is-fueling-real-life-violence/

Hugging Face. (2023, March 27). *Narrativaai/deberta-V3-small-finetuned-hate_speech18 · hugging face*. Narrativaai/deberta-v3-small-finetuned-hate_speech18 · Hugging Face. https://huggingface.co/Narrativaai/deberta-v3-small-finetuned-hate_speech18

Mathew, B., Saha, P., Yimam, S. M., Biemann, C., Goyal, P., & Mukherjee, A. (2021, May). Hatexplain: A benchmark dataset for explainable hate speech detection. In Proceedings of the AAAI conference on artificial intelligence (Vol. 35, No. 17, pp. 14867-14875).

Mody, D., Huang, Y., & de Oliveira, T. E. A. (2023). A curated dataset for hate speech detection on social media text. Data in Brief, 46, 108832.

Papers with Code. (n.d.). *Machine Learning Datasets | Papers With Code*. Papers with Code: The Latest in Machine Learning. https://paperswithcode.com/datasets?task=hate-speech detection&lang=english&page=1

Pennington, J., Socher, R., & Manning, C. D. (2014, October). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* (pp. 1532-1543).

Zhang, Z., Robinson, D., & Tepper, J. (2018). Detecting hate speech on twitter using a convolution-gru based deep neural network. In The Semantic Web: 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3–7, 2018, Proceedings 15 (pp. 745-760). Springer International Publishing.

Appendix A1. Experiment Results

Model	Details	Vocab Size	Max Length	Trainable Params	Train Accuracy	Validation Accuracy	Test Accuracy	Test Precision	Test Recall	Test F1
1D CNN 128 Filter	Word Embedding, 1D CNN with 128 Filters, 50% Dropout, Global Average Pooling, 64 Unit Dense Layer, 50% Dropout	50,000	300	12,906,753	0.8481	0.8301	0.8333	0.84	0.83	0.83
1D CNN 256 Filters	Word Embedding, 1D CNN with 256 Filters, 50% Dropout, Global Average Pooling, 64 Unit Dense Layer, 50% Dropout	50,000	300	13,013,377	0.8434	0.8263	0.8269	0.84	0.83	0.83
BiGRU	Word Embedding, Bidirectional GRU with 64 Units, 50% Dropout, 64 Unit Dense Layer, 50% Dropout	50,000	300	12,931,969	0.8883	0.8434	0.8442	0.85	0.84	0.84
BiLSTM	Word Embedding, Bidirectional LSTM with 64 Units, 50% Dropout	10,000	300	2,724,481	0.8526	0.832	0.8313	0.83	0.83	0.83
BiLSTM	Word Embedding, Bidirectional LSTM with 64 Units, 50% Dropout	20,000	75	5,284,481	0.8638	0.8347	0.8347	0.84	0.83	0.83
BiLSTM	Word Embedding, Bidirectional LSTM with 64 Units, 50% Dropout	20,000	300	5,284,481	0.879	0.8391	0.8386	0.84	0.84	0.84
BiLSTM	Word Embedding, Bidirectional LSTM with 64 Units, 50% Dropout	50,000	300	12,964,481	0.8871	0.8426	0.8428	0.84	0.84	0.84
BiLSTM	Word Embedding, Bidirectional LSTM with 64 Units, 50% Dropout	100,000	300	25,764,481	0.8719	0.8405	0.8408	0.84	0.84	0.84
BiLSTM + Dense	Word Embedding, Bidirectional LSTM with 64 Units, 50% Dropout, 64 Dense Layer, 50% Dropout	50,000	300	12,972,673	0.8729	0.8387	0.8409	0.84	0.84	0.84
CNN + BiLSTM	Word Embedding, 50% Dropout, 1D CNN with 100 filters kernel size 4 padding same, MaxPooling 1D with Pool Size 4, Bidirectional LSTM with 100 units, Global Max Pooling 1D, 64 Unit Dense Layer, 50% Dropout	50,000	300	15,293,829	0.8873	0.8485	0.8494	0.85	0.85	0.85

CNN + LSTM	Word Embedding, 50% Dropout, 1D CNN with 100 filters kernel size 4 padding same, MaxPooling 1D with Pool Size 4, LSTM with 100 units, Global Max Pooling 1D, 64 Unit Dense Layer, 50% Dropout	50,000	300	15,207,029	0.8882	0.849	0.8498	0.85	0.85	0.85
CNN +Bi GRU	Word Embedding, 50% Dropout, 1D CNN with 100 filters kernel size 4 padding same, MaxPooling 1D with Pool Size 4, Bidirectional GRU with 100 units, Global Max Pooling 1D, 64 Unit Dense Layer, 50% Dropout	50,000	300	15,254,229	0.8695	0.842	0.8433	0.85	0.84	0.84
CNN + GRU	Word Embedding, 50% Dropout, 1D CNN with 100 filters kernel size 4 padding same, MaxPooling 1D with Pool Size 4, GRU with 100 units, Global Max Pooling 1D, 64 Unit Dense Layer, 50% Dropout	50,000	300	15,187,229	0.8709	0.8429	0.8429	0.84	0.84	0.84
GloVe Pretrained Embeddings CNN + LSTM	Glove Pretrained Embedding, 50% Dropout, 1D CNN with 100 filters kernel size 4 padding same, MaxPooling 1D with Pool Size 4, Bidirectional LSTM with 100 units, Global Max Pooling 1D, 64 Unit Dense Layer, 50% Dropout	50,000	300	213,829	0.8257	0.8131	0.8145	0.82	0.81	0.81
GloVe Pretrained Embeddings Fine-tuned CNN + LSTM	Fine-tuned Glove Pretrained Embedding, 50% Dropout, 1D CNN with 100 filters kernel size 4 padding same, MaxPooling 1D with Pool Size 4, Bidirectional LSTM with 100 units, Global Max Pooling 1D, 64 Unit Dense Layer, 50% Dropout	50,000	300	15,293,829	0.9137	0.855	0.8552	0.86	0.86	0.85
Transformer	Positional Embedding, Transformer Encoder with 2 heads, Global Max Pooling, 50% Dropout	50,000	300	13,420,833	0.8754	0.8419	0.8443	0.84	0.84	0.84
Transformer 4 Heads	Positional Embedding, Transformer Encoder with 4 heads, Global Max Pooling, 50% Dropout	50,000	300	13,963,073	0.8732	0.8415	0.8444	0.84	0.84	0.84
GloVe Fine-tuned Embeddings Transformer	GloVe FineTuned Embedding,Transfomer Encoder, 50% Dropout	50,000	300	1,484,165	0.8827	0.847	0.851	0.85	0.85	0.85
TF-IDF Bigrams(Bag of Words)	64 Unit Dense Layer, 50% Dropout	50,000	300	3,200,129	0.8961	0.8584	0.8576	0.86	0.86	0.86
TF-IDF Bigrams Deep Regularized(Bag of Words)	128 Unit Dense Layer, BatchNormalization, 50% Dropout, 64 Unit Dense Layer, BatchNormalization, 50% Dropout	50,000	300	6,408,833	0.9119	0.8626	0.862	0.86	0.86	0.86