Deep Learning Approaches to Classifying Types of Toxicity in Wikipedia Comments

Ashe Magalhaes
ashe@cs.stanford.edu

Howard Small
hsmall@stanford.edu

Abstract

The increase in the accessibility and use of written expression on the Internet over the past decade has brought an alarming increase in toxic communication. This antisocial behavior has negative psychological effects and triggers more antisocial behavior. Thus, the task of identifying and removing toxic communication from public forums is critical. The task of analyzing a large corpus of comments is infeasible for human moderators. As a result, researchers are incentivized to automate the task of detecting inappropriate content. In this paper, we discuss our approach for using Natural Language Processing (NLP) techniques to develop a series of Deep Learning models capable of predicting whether or not internet comments contain various forms of toxicity. We begin by exploring the accuracy of a number of models in detecting toxicity on a test set classified by humans. We then tune the best model in order to achieve competitive results.

1 Introduction

1.1 Background

Today a substantial amount of user activity on the Internet involves antisocial behavior. According to a 2014 survey, 40% of Internet users were victims of online harassment [1]. An analysis of personal attacks at scale demonstrates that the majority of personal attacks on Wikipedia are not the result of a few malicious users nor the consequence of allowing anonymous contributions. Rather, it has been shown that people can be influenced by their environment to act aggressively; that is to say, a forum which includes toxic comments is likely to provoke more toxic comments from its users [2].

This complicates the problem of toxic comment classification, as it suggests that online moderators must detect toxicity early in order to mitigate situational factors that may influence users to react negatively. Moreover, many platforms struggle to effectively facilitate conversations. It has been shown that less than a fifth of personal attacks currently trigger any action for violating Wikipedia's policy [3].

1.2 Related Work

Prior work from the Conversation AI team, a research initiative within Alphabet, focuses on classifying toxic comments through a range of models served through the Perspective API [4]. However, the current models still make errors and they do not allow users to select which types of toxicity they are interested in finding. This limits online discussion; for example, some platforms may be fine with profanity that is not malicious.

We build on the work of Kolhatkar and Taboada [5] who generated a long short term memory model (LSTM) "constructiveness classifier" for news comments by extending their work to focus on the inverse, toxicity. Similar to their general architecture, we initialize the embedding layer with pretrained GloVe vectors that are fed into a bidirectional LSTM.

2 Approach

Our project focused on developing a series of Neural Network models to compete in the Toxic Comment Classification Challenge on Kaggle. The goal of this competition is to build a model which can, for a given set of Wikipedia comments, predict the probability that each comment contains each of 6 different forms of toxicity (described in more detail in the **Experiments** section). We chose this particular field to work in because it provided us with a large dataset of labeled comments, allowing us to focus on Deep Learning and NLP instead of web scraping and hand labeling our own data. Additionally, the Kaggle infrastructure gave us a concrete way to judge how well our models performed relative to some state-of-the-art models.

The primary goal of our project was to learn about the strengths and weakness of different Deep NLP models on text classification tasks. We naturally wanted to develop the best model possible, but winning the Kaggle competition was secondary to our goal of exploring a breadth of Deep NLP techniques and building up our intuitions for research we will do in the future. The specific models we developed for this project are as follows:

1. Convolutional Neural Network (CNN)
2. Recurrent Neural Network (RNN) with Long Short Term Memory (LSTM) cells
3. Multi-Layer, Bidirectional RNN with LSTM cells
4. Character-Level RNN

We now describe the specifics of the network architectures we used for each of these models.

2.1 Structures Shared By All Models

2.1.1 Model Input/Output

Although the models vary, the input for each of the Word-Level models was a fixed-size list of the first 100 words of a Wikipedia comment. Comments shorter than 100 words were padded with NULL words. We chose to use 100 words, as this was approximately the largest number of words we could use without running into consistent "Out of Memory" errors. For the Character-Level RNN the input was 500 characters. The output for each model was a 6-unit layer with a sigmoid activation to represent the predicted probabilities of each of the 6 types of toxicity. We use a cross entropy loss. We use the Adam Optimizer for training, a learning rate of 1e-4, and a batch size of 100.

2.1.2 Word Embeddings

We experimented with two types of word embeddings, random and GloVe. We also experimented with one-hot character embeddings. We chose our vocabulary to be all words which occurred at least 20 times in the dataset in order to limit it to words which the models can learn from. All other words were treated as the same "unknown word" (UNK). We chose our word embedding size to be 200 since this was the largest GloVe vector size available. For the random word embeddings, the entire embedding matrix was initialized using Xavier Initialization. For the GloVe initialization, we augmented our vocabulary to also include every word that appears in both the GloVe dataset and our comments, regardless of word count. The intuition is that the models can still extrapolate to words which it saw rarely due to the initialization. We then assigned the word embedding for each word to be the GloVe vector if there was one, and a random initialization with the same mean and standard deviation as the GloVe vectors otherwise. An important note about the GloVe vectors is that we chose to use the Twitter GloVe vectors since they better account for misspellings and the type of informal language that is used on the Internet [6].

We also decided to include a second component in our word embeddings, capitalization. For each word, we appended a 3-dimensional representing whether the word was lower case, mixed case, or upper case. We did this so that the network would not lose information about capitalization, which is often important in determining the tone of an online comment. This resulted in a final word embedding size of 203.

2.2 Model Structures

2.2.1 CNN

The CNN model we built consists of one 1-dimensional convolutional layer across the concatenated word embeddings for each of the 100 words in the input comment. The convolutional layer has 64 filters, a kernel size of 5 x 203 and a stride of 203 so that each convolution will consider a window of 5 word embeddings. This layer is followed by a fully-connected layer with 50 units, which is then followed by the output layer.

2.2.2 LSTM

The LSTM model is attractive because it the individual cell states in the model have the ability to remove or add information to the cell state through gates layers. This is useful in practice because it allows the model to remember insights derived from words throughout the comment [7]. The LSTM model consisted of one densely connected layer with 50 units and 24 hidden states across the concatenated word vectors for each of the 100 words in the comment.

2.2.3 Bidirectional LSTM

The Bidirectional LSTM model we build consists of 3 layers of bidirectional LSTM cells with 256 units each. The output of the third layer is fed into a max-pool layer to take the maximum activation across all timesteps for each unit. The max-pool layer is then fed into a fully-connected layer with 64 units and ELU activation, which is then followed by the output layer.

2.2.4 Character-level modeling

A character level model will use the character as the smallest entity. In general, character-level modeling can help deal with common misspellings, different permutations of words, and languages that rely on context for word conjugations [8]. The model reads characters one by one, including spaces, and creates a one-hot embedding of the comment. We choose 500 characters as our max entity length and tested it with our LSTM model.

3 Experiments

3.1 Dataset

The dataset we used for our project comes directly from the Toxic Comment Classification Challenge. Competitors are provided with a **training set** consisting **159,572 Wikipedia comments** labeled by human raters as containing any number of the 6 types of toxicity.

1. Toxic
2. Severe Toxic
3. Obscene
4. Threat
5. Insult
6. Identity Hate

Competitors are also provided with a **test set** consisting of **153,165 unlabeled Wikipedia comments**, which is used by the competition to evaluate the relative performance of competitor models. All of the comments in both the training and test sets are provided in raw text format, meaning that they include the comments exactly as they are on Wikipedia (with misspellings, spacing, formatting, etc.). Data like this requires a bit of preprocessing so that our models can be given more consistent data to learn from.

3.2 Preprocessing

For preprocessing, we needed to convert each comment into an ordered list of the words it contains. A trivial way to do this is to just split the comment on whitespace, but this does not account for punctuation. For example, using this method would mean that almost every word at the end of a sentence would have a period appended to the end. However, we do not want to treat these words as different because they are obviously the same word. To handle this we decided to split up comments using the following regular expression: **[“\w’-]+ | [.,!?;]”**, which captures all words (with apostrophes and/or dashes included) and also treats the most common types of punctuation as words. This breaks up words in the same way that most English-speakers would think to do so, and isolates punctuation so that it does not get mixed up with words and also allows the model to utilize more syntactic information. One limitation of the dataset is that, because consists of Internet comments, there are a fair amount of misspelled words in the dataset. We, however, decided not to directly address this problem as we felt it would be quite tedious to come up with a set of rules to do spell correction and provide fairly marginal benefit.

3.3 Evaluation Metrics

Intrinsically, we evaluate our models using the **mean Area Under the Receiver Operating Characteristic (mean-AUROC)** across the 6 toxicity classes on a held-out development set. We chose this metric because, due to the imbalanced nature of the labels in the dataset, accuracy is a poor metric to gauge performance while having a high mean-AUROC requires our classifier to have high precision and recall. Additionally, this is the metric used by the Kaggle competition, which helps us optimize our results for doing well in the competition. Extrinsically, we evaluate our models by using the scores generated by Kaggle's leaderboard and our relative leaderboard positioning.

3.4 Experimental Setup

For the purposes of development, we train our models on **140,000** of the comments in the training set, and evaluate performance on the remaining **19,572** comments. Given that NLP tasks usually require a huge amount of training data, we wanted to maximize the size of our training set while still maintaining a large enough development set to have a statistically significant evaluation of how well our model generalizes.

The training process for each of our models involves minimizing the **mean binary cross entropy loss** across the training set, the formula for which is:

$$-\frac{1}{N}\sum_{n=1}^{N}\left[y_n \log \hat{y}_n + (1-y_n)\log(1-\hat{y}_n)\right] \tag{1}$$

Initially we thought to use mean squared error as our loss function, but found trouble in that our models would frequently converge to predicting near zero for every single class for every single input. Mean binary cross entropy loss avoided this problem by better handling each toxicity class independently instead of as a 6-dimensional vector.

3.5 Tuning the Best Model

Bidirectional LSTM: We found that our single layer, bidirectional LSTM model slightly outperformed our basic LSTM and CNN models. Because of this, we decided to focus more of our effort on tuning the bidirectional model.

The main steps we took were to try different ways of combining the output of the LSTM layer across all of the timesteps. We tried taking an average over the the outputs, a concatenation of the first and last timesteps, and an element-wise max of the outputs. The difference in performance was marginal between the 3, but the max-pooling consistently achieved a mean-AUROC score of about 0.01 above the other two methods. We suspect that this is maybe because max-pooling better captures the extremes across all of the timesteps, which facilitates the network learning to recognize extreme language.

Table 1: Performance of our Models on the Development Set with Various Word Embeddings

	Random Word Embeddings		GloVe Word Embeddings	
	Non-Trainable	Trainable	Non-Trainable	Trainable
CNN	AUROC: 0.9145	**AUROC: 0.9769**	AUROC: 0.9174	AUROC: 0.9749
	Dev Loss: 0.0868	**Dev Loss: 0.0506**	Dev Loss: 0.0876	Dev Loss: 0.0525
LSTM	AUROC: 0.8841	**AUROC: 0.9718**	AUROC: 0.9396	AUROC: 0.9653
	Dev Loss: 0.1007	**Dev Loss: 0.0545**	Dev Loss: 0.0749	Dev Loss: 0.0575
Bidirectional	AUROC: 0.9222	AUROC: 0.9764	AUROC: 0.9756	**AUROC: 0.9802**
LSTM	Dev Loss: 0.0801	Dev Loss: 0.0508	Dev Loss: 0.0559	**Dev Loss: 0.0493**

			One-Hot Embeddings
Character-Level	**AUROC: 0.6715**	AUROC: 0.6692	AUROC: 0.6709
LSTM	**Dev Loss: 0.1321**	Dev Loss: 0.1314	Dev Loss: 0.1318

After tuning the basic LSTM structure, we experimented with adding more bidirectional LSTM layers to see how it would improve the performance. We trained models with 1, 2, 3, and 4 bidirectional LSTM layers, and found significantly diminishing returns as we added more layers. Before adding dropout to the networks, the 2 layer model outperformed all of the others, as the 3 and 4 layer models were highly susceptible to overfitting. However, after adding a dropout of 50% throughout the entirety of the network, with the exception of the input and output layers, the 3 layer network was able to outperform all of the other networks by at least a 0.005 in terms of mean-AUROC scores.

3.6 Discussion of Results

To get a good sense of how the model type and word embedding initialization affect the performance on our task of toxic comment classification, we evalutated each of our models with each of our word embedding initializations. For each type of word embedding, we also experimented with making the word embeddings trainable and non-trainable to see how much impact this had on the model performance. The results of these experiements can be seen in **Table 1**. The bolded cell in each row denotes the word embedding initialization which performed best for each model.

What we found to be the biggest surprise was that our CNN model consistently outperformed our LSTM model. We know that LSTMs (or some variant) are generally the go-to network structure for most NLP related problems, so to see the CNN get better performance at first seems unintuitive. Our intuition for this result has to do with the specific nature of our problem, compared to general NLP tasks. What makes LSTMs so powerful is that they are able to capture long-term dependencies by mitigating the vanishing gradient problem. On the other hand, what makes CNNs so powerful is that they allow you to generalize knowledge about local dependencies in the input to many different areas in the input. The fact that the CNN model outperformed the LSTM model implies that the local information is more important in this task than the long term information. This actually makes a good amount of sense because most toxic comments are characterized by outbursts of obscene language. *This means that toxicity can be often be identified locally from a few offensive phrases or curse words in a comment, rather than needing the more global view of the comment.* It is important to note, however, that the difference in performance between these two models is marginal, as they both do an excellent job of identifying toxicity. The CNN however required significantly less training and evaluation time, implying that it is generally the better choice for this task.

Our Bidirectional LSTM model outperforms all of our other models and this is largely due to it consisting of more layers than the other networks. Before we added multiple layers to increase its expressive power, we found that its performance was barely distinguishable from that of the CNN. After increasing the number of layers, the training time for the Bidirectional LSTM was dramatically more than the CNN, and the performance increase was still marginal. The Bidirectional LSTM, however, was better able to handle the issue of overfitting when we added dropout than the CNN was, implying that the CNN architecture is more prone to overfitting.

Another thing we found interesting was that for the CNN and LSTM models the Random word embedding initialization was better, while for the Bidirectional LSTM the GloVe word embeddings performed better. Perhaps the added complexity of the Bidirectional LSTM allowed it to take better use of the GloVe vectors, or this is just due to random noise, we are definitely not sure. We were unsurprised though to see that allowing the word vectors to be trainable made a big difference in performance. The difference was much larger for Random initialization, which is logical since the GloVe initialization provides a better initial starting point.

Our Character-Level LSTM had disappointing, but unsurprising results. Looking a the input character-by-character instead of word-by-word means that the model has to first reason about how to combine characters into words, and then reason about the implication of those words. This obviously makes the task more difficult in a significant way, so character level models are not effective for this task.

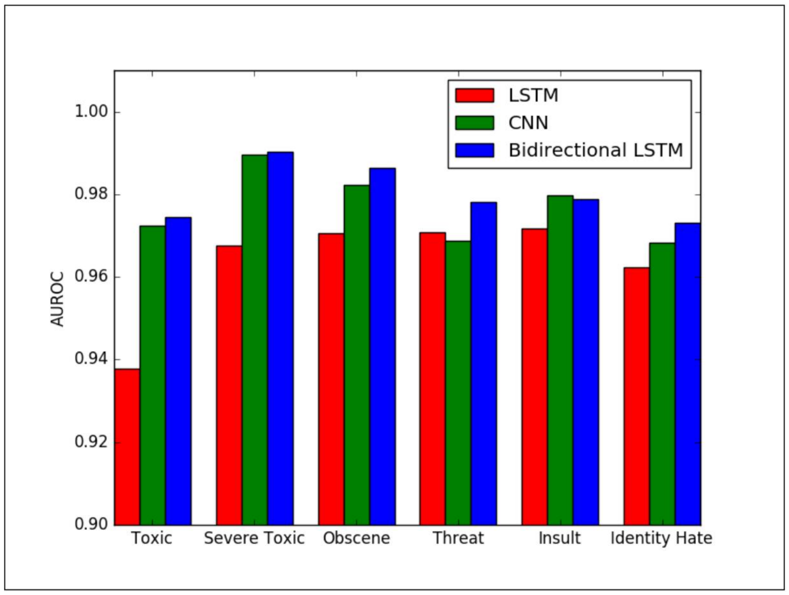

Figure 1: Performance of our best models on each toxicity class

Finally, as can be seen in **Figure 1**, certain classes of toxicity proved to be more easily identified by our models than others. Namely, the *severe toxic* and *obscene* classes had higher AUROC scores across the models than the *toxic* and *identity hate* classes. This makes sense because in general it should be easier to identify more extreme types of toxicity since it will require the use of extreme language, and obscene language is largely characterized by a handful of bad words. However, while identity hate is similarly related to a set of words about someones sexuality, religion, race, etc., words like “gay" or “jew" can be used in non-toxic contexts just as easily as non-toxic contexts, making it more difficult for models to identify whether it should be toxic or not.

4 Conclusion

In doing this project, we were able to get a much better appreciation for how to build, tune, and analyze the performance of a Deep NLP model. By working with different types of models and word embedding initializations, we were also able to learn about which models may be better suited for the task of toxic comment classification. We found that our best model was a **bidirectional LSTM**, however, its performance was only marginally better than our **CNN** model which required much less training and evaluation time. In situations where speed and availability of a model are critical, such as monitoring comments posted on a large website, it may make sense to have a more light-weight CNN model so that website traffic is not hindered. The high performance of our CNN model relative to our LSTM models was initially surprising, but upon thinking about the specific problem of toxic comment classification, we realized that the locality constraint of CNNs fits well with the way that toxicity is generally formatted as short outbursts of rage. Finally, we confirmed our hypothesis about the ineffective nature of Character-Level modeling for toxic comment classification.

Acknowledgments

We would like to thank our project mentor Kevin Clark for consistently answering our questions and guiding us towards relevant literature.

References

[1] Duggan, M., Rainie, L., Smith, A., Fuck, C., Lenhart, A., & Madden, M. (2014). Online harassment. Pew research center.

[2] Cheng, J., Bernstein, M., Danescu-Niculescu-Mizil, C., & Leskovec, J. (2017). Anyone can become a troll: Causes of trolling behavior in online discussions. arXiv preprint arXiv:1702.01119.

[3] Wulczyn, E., Thain, N., & Dixon, L. (2017, April). Ex machina: Personal attacks seen at scale. In Proceedings of the 26th International Conference on World Wide Web (pp. 1391-1399). International World Wide Web Conferences Steering Committee.

[4] "Perspective." Perspective. Accessed March 17, 2018. https://perspectiveapi.com/#/.

[5] Kolhatkar, Varada, and Maite Taboada. "Constructive Language in News Comments." In Proceedings of the First Workshop on Abusive Language Online, pp. 11-17. 2017.

[6] Pennington, Jeffrey. GloVe: Global Vectors for Word Representation. Accessed March 17, 2018. https://nlp.stanford.edu/projects/glove/.

[7] "Understanding LSTM Networks." Understanding LSTM Networks – Colah's Blog. Accessed March 17, 2018. http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[8] Chiu, Jason PC, and Eric Nichols. "Named entity recognition with bidirectional LSTM-CNNs." arXiv preprint arXiv:1511.08308 (2015).