Replication of Deep Learning Approaches to Classifying Types of Toxicity in Wikipedia Comments

Mu Li
mul026@ucsd.edu

Sung-Yin Yang
suyang@ucsd.edu

Chih-Hsuan Kao
c4kao@ucsd.edu

Abstract

As the use of written expression on the Internet ran wild over the past few years, an alarming increase in toxic communication has brought to our attention. This antisocial behavior embodies negative psychological effects and furthermore triggers more antisocial behavior. Thus, the task of identifying and removing toxic comments manually seems more infeasible and unrealistic. Therefore, researchers are prompted to automate the task of detecting toxic 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 forms of toxicity. We explore the accuracy of a variety of models in detecting toxicity on a test set classified by humans, and facilitate a discussion based on our observation.

1 Introduction

1.1 Background

Nowadays, as the threat of abuse and harassment online arises significantly, people refrain from expressing themselves or seeking different opinions through writing things online. Online platforms, such as Wikipedia, have been struggling to effectively facilitate conversations, leading many communities to limit user comments, which is the last thing our modern society desires. Inspired by the Toxic Comment Classification Challenge on Kaggle Machine Learning Competition in 2018[1], and a final project report published later by a group of Stanford graduate students of the course CS224n: Natural Language Processing with Deep Learning[2], we aspire to build a similar multi-headed model that's capable of detecting different types of of toxicity like threats, obscenity, insults, etc.

Using a dataset of comments from the talk page edits of Wikipedia, we primarily refer to the approaches and structures specified in the project paper by Stanford students, while making some adjustments to the model which we regard as ideal to the overall performance. Hopefully, we aim to make improvements to existing models and contribute to the research field and a better online discussion environment.

1.2 Approach

As mentioned, our project focuses on replicating and making adjustments to the result of the paper, Deep learning Approaches to Classifying Types of Toxicity in Wikipedia Comments[2]. The main idea of this paper is to develop a multi-headed deep learning model to compete in the Toxic Classification on Kaggle. Similarly, given a specific dataset, we try to build a model that could possibly predict the possibility that each sentence contains certain types of the 6 forms of toxicity, as specified later.

By performing text classification tasks on our NLP model, we hope to further learn about the strengths and weakness of different NLP models that we utilize. Specifically, the models we developed for this project are as follows:

1. LSTM (Long Short Term Memory cells)
2. Bidirectional RNN with LSTM cells
3. Convolutional Neural Network (CNN)

2 Structure

2.1.1 Data Set

The dataset we use comes from the Kaggle competition - Toxic Comment Classification Challenge[1]. We are given a training set of consisting of **159,572 Wikipedia comments**. Each of them are labelled, by human raters for toxic behavior, as any number of the the following type.

a. Toxic
b. Severe Toxic
c. Obscene
d. Threat
e. Insult
f. Identity Hate

We are also given a **test set** consisting of 153,165 Wikipedia comments. However, as a competition data set, only **63,978 data** set are labelled for scoring to deter hand labelling. We are not able to test on the unlabelled data; therefore, for testing, we only use the labelled data.

2.1.2 Model Input and Word Embeddings

The input for each of the Word-level model is a fixed-size list of the first 100 words of a Wikipedia comment. Comments that are shorter than 100 words were padded with NULL words.

We use the GloVe word embedding. Our word choices are words which appeared at least 20 times in the whole dataset in order to limit the word that we need to learn. The words that are not appeared at least 20 times are all treated as the same "<UNKNOWN>". The final word embedding size is 103: the first 100 columns are the word embedding, and the last three columns record lower case, mixed case and upper case respectively using one-hot encoding. The reason is that we are unsure about how capital letters would affect the result of our training. It is possible that capital letters could be used as certain form of toxicity. Unlike our reference, in which a batch size of 100 is used, we use a batch size of 512, which is a multiplication of 2. Most of the research papers consider the multiple of 2 such as 32, 64 and 128 elements per mini batch. For the size of epoch, we set it to be 100.

In the first experiment, the GloVe vectors that we use are pretrained vectors from the Wikipedia 2014 dataset. In subsequent parts, we will also utilize another set of pretrained vectors from the Twitter dataset.

2.2 Model Structures

The loss function we use is the binary cross-entropy. For the optimizer, we use Adam, an algorithm for first-order gradient-based optimization of stochastic objective functions. Then, the activation function that we apply for our output layer is sigmoid.

2.2.1 CNN

The CNN model is initially designed for processing picture. The idea of using CNN for natural language process is to treat each vectorized sentence as a picture and to use a filter of dimension n (embed-dim) to look at each of the n-word phrase.

The CNN model we build includes a 1-dimensional convolutional neural network across the word embedding for each of the 100 words per input comment. The convolutional layer has 64 filters, a kernel size of 5*103 and a stride of 1. After the convolution layer, we have a multi-layer perceptron of 50 units, and then the output layer.

2.2.2 LSTM

The LSTM model is consist of 24 hidden states. The output is a 6-unit layer with a sigmoid activation to represent the predicted probabilities of each of the 6 types of toxicity. We use the Adam optimizer, a learning rate of 0.005, and a batch size of 512 for training.

The LSTM model is well suited for making prediction based on the sequential data, since the cell takes an input of the previous hidden state and the next input value.

Let h be hidden state vector and R be the recurrent network function.

Then, every next hidden state would depend on the previous hidden state and the next input.

$h_t = \mathrm{R}(h_{t-1}, X_t) = \mathrm{ReLU}\,(W_h * h_{t-1} + W_x * X_t + \mathrm{b})$

ReLU means ReLU activation over here.

W_h = hidden state matrix

W_x = input matrix

2.2.3 Bidirectional LSTM

Bidirectional RNN with LSTM cells connects two hidden layers of different directions to the same output. With this setup, the output layer could get information both from the forward and backward at the same time.

The BLSTM model is consist of 24 hidden states. The output is a 6-unit layer with a sigmoid activation to represent the predicted probabilities of each of the 6 types of toxicity. We use the Adam optimizer, a learning rate of 0.005, and a batch size of 512 for training.

3 Experiments

3.1 Preprocessing

For preprocessing, we needs to convert each comment into an ordered list of words. We simply split the comments by space, and consider every punctuation similarly as a word. Since we are classifying toxicity, we decide to only keep the punctuations "? ! *" that could possibly express toxicity. For example, people often use **** to implicitly express hatred, as well as the question mark and the exclamation mark to offer insult. The punctuations that we remove include "( ) ~ @ $ % ^ & _ + = { } [ ] \ | ' ' : ; / > < . , ". It is to be observed that we also remove the comma and period, which serve to break up English sentences into fragments. Since our dataset comes from the Internet, people seldom use formal language as for word choice. Some even use new lines or space as replacements of comma and period, and neither of those could be used to express toxicity.

3.2 Basic Experiments

We put more emphasis on the word preprocessing part, since identifying the toxicity that appears in a word or a certain phrase plays a bigger part in our task. Thus, instead of making a complicated model with a compacted structure, we spend more time preprocessing our dataset.

We decide to evaluate our model using Accuracy instead of AUROC. For the first preprocessing section, we do not apply any preprocessing. We use GloVe to embed the word vectors directly without removing any unnecessary punctuations or redundant words. We regard this as a baseline model.

Secondly, we apply basic preprocessing in the next part. We remove some punctuations such as "@#%^&" , but we keep all of other punctuations such as comma and period. One should bear in mind that as we consider that the result for this part is identical to the former one, this result is not shown in the table below (**Table 1**).

In the last preprocessing, we apply a more extensive preprocessing than the basic one, as we decide to only keep the punctuations "? ! *" that could possibly express toxicity.

We lastly put a column that represents the total loss after final epoch, which could inform us how well the model is trained.

Table 1: Comparison among different preprocessing (Training with default GloVe embedding**)**

Epoch size = 100 , Learning rate 0.005	No preprocessing	Remove unnecessary punctuations and infrequent words except for "? ! *"	Total loss after final epoch
LSTM	85.39%	88.06%	16.122
Bidirectional LSTM	87.63%	88.85%	10.735
CNN	83.92% (10,0.01) 84.58%	88.87% (10,0.01) 83.48%	6.411

The results above for LSTM and bidirectional LSTM are almost identical in both cases. We suppose the reason is that all words that influence the toxicity of a sentence don't quite depend on the context of later part of the sentence. So bidirectional LSTM does not offer any significant advantage over LSTM.

A lower total loss indicates a higher training accuracy. We believe that CNN results in the lowest total loss due to the relatively large amount of parameters used.

With an epoch size of 100 and a learning rate of 0.005, we see that the accuracy for both LSTM and BLSTM improve as we apply more preprocessing. Nevertheless, our accuracy for CNN decreases a little bit under this condition. We suppose that it is because our CNN model, at learning rate = 0.005 and epoch size = 100, is overfitted. Thus, When we increase our learning rate to 0.01 in CNN, the accuracy improvement is shown. However, we consider that overfitting still occurs after improvement because the total loss is still extremely low. Since we have more than 150,000 training data, total loss of around 6 is really small.

Table 2: Training with Pre-trained Word Vectors of GloVe embedding from Twitter

	GloVe embedding from Twitter Dataset (Remove unnecessary punctuations and infrequent words except for "? ! *")	Total loss after final epoch
LSTM	86.69%	11.3191
Bidirectional LSTM	86.17%	10.0512

CNN	84.75%	4.9712

Table 2 shows the result that we get using another GloVe embedding, from Twitter dataset. We train our models with the same setting (epoch = 100, learning rate = 0.005). Under this condition, we see that the training accuracy of LSTM and BLSTM both decrease, while CNN increases slightly. The reason is simple, as overfitting occurs in our models seriously in this case. We can see this fact from the decreased total loss after final epoch, compared to the previous table. This indicates that the Twitter dataset actually trains our model better than the Wikipedia dataset.

Therefore, from these results, we figure out that preprocessing data in classifying toxicity is really important. A good pruning of redundant data could effectively increase our testing accuracy. However, the problem of overfitting often decrease our testing accuracy.

3.3 Avoid Overfitting

Since we have learned that overfitting occurs in our models, our next attempt is trying to adjust the model to avoid overfitting. The first parameter we adjust is the threshold. In general, we find out that the lower the final loss is, the higher the threshold should be. Specifically, when the total loss is around 5, the threshold should be 0.9 to achieve a test accuracy of about 89%. After testing, we find out that a loss at above 50 is reasonable for a threshold at 0.6.

We choose to tune the CNN model since it is more computationally efficient. Initially, I add a dropout layer with rate 0.2 at the end of convolutional layer and a dropout layer with rate 0.5 at the end of multi-layer perceptron. We also add batch normalization after the convolutional layer. However, after adding all those regularization, although the loss has increased during train, the accuracy only improves by a small amount. Specifically, with 100 epoch and learning rate 0.005, the test accuracy is 89.02%.

In order to figure out this mystery, We calculate the validation accuracy for each epoch. Surprisingly, the best validation accuracy of 90.23% is achieved at the 1st epoch when the learning rate is 0.005, and at around the 8th epoch with the learning rate of 0.0001.

We add an embedding layer to see whether training the word embedding could influence the result. We use both the GloVe pre-trained weight matrix and randomly-initialized weight matrix for our embedding layer. For the GloVe pre-trained weight matrix, the best validation accuracy of 90.67% is achieved at the 3rd epoch when the learning rate is 0.005, and at around the 22nd epoch when the learning rate is 0.0001. On the other hand, for the random-initialized weight matrix, the best validation accuracy is achieved at the 22nd epoch when the learning rate is 0.005, and at around the 35nd epoch when the learning rate is 0.0001.

3.4 Discussion

From the result above, it is clear that without an embedding layer, the train could be done within the first few epochs. We suppose the reason for that is that since we use GloVe pre-trained word embedding for our model, there isn't much to learn after a few epochs, and the rest of epochs are just simply overfitting the data.

With the embedding layer, using GloVe pre-trained weight achieves slightly better result, but since the model only makes small adjustments to the weight matrix, the train still ends fast. However, with the randomly-initialized weight matrix, the train lasts longer with a lower accuracy. It shows that nn.Embedding does a worse embedding compared to GloVe.

Our result shows that the most important thing for classifying toxic comment is preprocessing and word embedding. Neither nn.Embedding nor GloVe is suitable for this task. In order to achieve higher accuracy, a specialized Word2Vec algorithm may be needed. Besides, our preprocessing simply replaces misspelled and infrequent words with <UNKNOWN>, while in reality a lot of people intentionally misspell toxic words to avoid toxicity detection. A correct preprocessing of misspelled words could potentially ameliorate the result.

4 Conclusion

We conclude that preprocessing of the dataset and word embedding play the most critical role in classifying types of toxicity in our task. Since serious overfitting of the data occurs, as the latter epochs do not contribute much to the training for our models, the results we get are not as ideal. This indicates that tuning the best model for this task cannot be done without a specialized Word2Vec algorithm other than nn.Embedding and GloVe.

As for the structure of our models, we conclude that BLSTM does not significantly outperform the LSTM model, since words that influence the toxicity of a sentence are generally independent of the context of later part of the sentence. Again, as we discover, choosing and complicating the model structure don't do much for the overall performance of our task in the absence of a more accurate preprocessing of misspelled words, which obviously account for a relatively larger proportion amongst our dataset.

References

[1] https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge

[2] Magalhaes, Ashe, and Howard Small. "Deep Learning Approaches to Classifying Types of Toxicity in Wikipedia Comments." *CS224n: Natural Language Processing with Deep Learning 2018 Project Report*, Stanford University, Mar. 2018, web.stanford.edu/class/cs224n/reports/6838795.pdf.