Toxic Speech Detection

Animesh Koratana & Kevin Hu
Stanford University
Department of Computer Science
{koratana,huke}@stanford.edu

Abstract

Due largely to the ubiquitous rise of social media over the last decade, the mode of communication that people subscribe to has significantly changed. While it's allowed for a more connected, and informed world, it has also made room for a new phenomenon: toxic speech. An open platform for the world to produce, comment, and share content has opened has allowed rather opportunistic users to participate in hate speech: peddling sexist, racist, xenophobic, and all around negative comments. Our project is motivated by this trend. We aim to develop high accuracy classifiers on a hate speech datasets using modern deep learning techniques to primarily identify the existence of hate speech in comments and texts in an efficient manner. We explore the use of various approaches to this classification problem, including using logistic regression, CNN, and RNN based models. We also evaluate their inference efficiency and proceed to propose and test a cascade model that achieves high throughput in the average case while maintaining a high accuracy by combining multiple approaches.

1 Introduction

With the rapid growth of online platforms and forums, people have become increasingly aware of the problem of abusive language and hate speech. When an individual decides to engage in an online discussion on social media, blogs, or comment sections, they are exposing themselves to the risk of being harassed by trolls or rude commenters. Instances of offensive comments are quite common and have negatively impacts the dynamics of the online community as well as the user experiences of the targeted individuals. This phenomenon, in addition to the supposed structured misinformation bias 2016 election, has led to new vigor in the field of toxic comment classification. As such, most companies that run internet platforms filters their sites by deploying a combination of human moderators and automated abusive language detection algorithms. Our project is motivated by this trend. Through our project, we intend to build and evaluate a toxic speech classifier using a few of the state-of-the-art, deep learning approaches. To name specific leading text classification architectures, we intend to implement and evaluate the ability of a Convolutional Bidirectional GRU with Attention layers and Very Deep Convolutional Neural Networks (VDCNN) (Conneau et al. [2016]) to classify and detect hate speech efficiently.

Many of the simple abusive language detection systems use regular expressions and a blacklist (which is a pre-compiled list of offensive words and phrases) to identify comment that should be removed. However, the problem with these models – and ultimately the difficulty of this task – is that hate speech is more than keyword recognition and pattern identification in the grammar. For instance, one can get around the system by removing spaces between words, use alternative spellings, or create homonyms that will still make sense to humans but cannot be detected by machines. The task, then, as suggested by the title of the paper, is to build an accurate hate speech detection using a deep neural network. We define hate speech from the definition established by Davidson et al. [2017] as *any* *communication that disparages a person or a group on the basis of some characteristic such as race, color, ethnicity, gender, sexual orientation, nationality, religion, or other characteristic.*

Recent trends in machine learning and deep learning in the context of natural language processing have yielded significant improvements to the ability of machines to detect subtleties in natural language, including hate speech. However, these speedups often come at the cost of speed and scalibility. Modern deep learning architectures have millions of parameters with modern architectures such as VDCNNs approaching close to 8M parameters (Conneau et al. [2016]). Using hate speech detectors in practice often bears a significant challenge of implementation. As modern deep learning architectures follow the general trend of increasing representational power with depth, companies are forced to be more cautious about their deployed architectures.

In this project we intend to explore both these arenas together. We will begin by exploring a robust baseline approach to classification using logistic regression. We then will proceed to make our novel contribution by evaluating an attention-based RNN, and CNN based classifier on the same dataset. We finally will evaluate standard classification approaches on the task of toxic comment classification, and explore their computational efficiencies.

2 Related Work

Existing works in this domain have tackled the problem of toxic speech detection as a classification task, which is the logical approach the goal is to either performing binary classification to identify whether a comment is toxic, or performing multi-class classification to identify the type of toxic comment. Among the variants of hate speech detection models, they can be separated into what Zhang et al. called the "classical methods" and deep learning methods. Specifically, the classical methods are ones that require manual feature engineering, and include Logistic Regression, Bayesian models, SVM, and random forest. On the other hand, deep learning methods use different neural structures to learn abstract, high-level features from the training data.

The classical methods have proven to be very successful in various circumstances, but also have its shortcomings. In particular, the feature engineering process takes enormous effort and is always inexhaustive, as people can easily warp their messages and find ways around being detected by the system. Some of the most common features include word vectors, character n-grams, lexical features (based on the assumption that hateful speeches contain negative or offensive words), linguistic features (such as POS tags), and knowledge-based features (like the relative appearance of "kill" and "Jews") which cannot be easily constructed and is always evolving. We think that while classical approaches have their merits, there is great potential for deep learning models in this field because its feature extraction is automated and can perhaps capture patterns and trend that humans cannot think of. As a result, logistic regression is a fitting choice of baseline. Not only is it simple to implement, it gives us a general idea of the performance of these classical methods which we are trying to outperform.

The two models that we implemented and tested in this project are inspired by existing text classification models but are slightly different from the ones that are tailored to this specific test. The idea of a VDCNN initially came to us from the success of CNNs in image processing. The particular model that we implemented is based on Conneau's VDCNN for text classification. A problem with the method is that it is character-based – as we increase the depth of the VDCNN, the training time increases exponentially, which we cannot fully test due to limitations in computational power. As such, we modified Conneau's model into a word-based model and used pre-trained word embedding from FastText. This drastically improves the training time at the sacrifice of the number of features (words instead of chars), and makes the project more manageable. Additional deep learning approaches include Zhang et al.'s use of a combination of CNN and GRU and Founta et al.'s use of a combination of text data (using RNN) and the online user's metadata to predict toxic comments. Given the success of RNN variants in "interpretting" meanings in text and its use in the various related works, we decided to implement a bi-directional GRU with attention to tackle the problem of hate speech detection.

3 Approach

For this project, we take two deep learning approaches to solve the toxic speech classification problem. The first model that we are using is a GRU RNN with attention, which is based off of the state-of-the-art LSTM and RNN paradigms, but uses specific modifications. The second approach is using a VDCNN (Very Deep Convolutional Neural Network), which is motivated by the success of very deep networks in computer vision, and based on the theory that deeper networks can encapsulate more information and achieve higher test accuracy. We have chosen two commonly used baseline models for NLP text classification: logistic regression with word and char n-gram features.

3.1 Baselines

Most of the recent papers in text classification and hate speech detection uses efficient, linear models as baselines. Badjatiya et al. uses logistic regression as a strong baseline to evaluate for text classification tasks.

The logistic regression baseline uses our familiar gradient descent update rule:

$$\theta_j := \theta_j + \frac{\alpha}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_y^{(i)}$$

What's more interesting is the process of feature extraction. The features that we use include both the words, as well as character n-grams (where $2 \leq n \leq 6$) present in the training data. Moreover, we weigh the features by applying a TF-IDF (term frequency – inverse document frequency) transform, which is a measure of the importance of each of the words and n-grams in the corpus. The TF-IDF score of a feature is calculated by multiplying its TF (term frequency) and IDF (inverse document frequency), both of which can have various interpretations. Our particular baseline uses the feature vectors extracted by TfidfVectorizer from the sklearn package. The baseline was implemented and tested by us.

3.2 GRU & LSTM with Attention

One type of model that we will be testing is an rnn based model. Specifically we will be testing variants of GRU and LSTM based models. Variants of the RNN have proven to be successful in many NLP tasks and we think that it will perform well with classification tasks. The current model that we implemented is showing promising results and will be further tuned for the final report.

Each comment is represented as a sequence of words, which is padded to the maximum length of the sentences in the training data. To reduce the time of training word embeddings, we start by looking up 300-dimensional vector representations using the FastText library. Then we pass through the word embeddings through a bi-directional GRU or LSTM to obtain the sentence embedding.

The embedding is then fed into a scaled dot product attention layer. This is slightly different from the dot product attention that we learned in class by incorporating a scale factor into the calculation of the attention scores. Specifically, we take the current decoder state to be the query q and take each of the encoder hidden states to be the keys. For each of the keys k_i, we calculate the attention score with the following formula:

$$\text{Attention Score} = \frac{q^T k_i}{\sqrt{d_k}}$$

where d_k is the dimension of the key vectors. The motivation of using scaled dot product attention is that the value of the dot product becomes very large with higher dimensions, which leads the softmax function to go into regions with exceptionally small gradients. Scaling the attention score by a factor of $\frac{1}{\sqrt{d_k}}$ will allow us to avoid or at least alleviate the issue (Vaswani et al.).

Its result of the model is mapped into a n-dimensional output space (n = 7 is the number of classes, including “clean”, “toxic”, “severe toxic”, “obscene”, “threat”, “insult”, and “identity hate”) through a fully connected linear decoder and converted into probabilities using the softmax function. The model is trained using CE loss and SGD optimizer. We implemented this model by hand and from scratch with the exception of the accuracy computing function, which was adapted from PyTorch's default ResNet training code [1].

[1]https://github.com/pytorch/vision/tree/master/torchvision/datasets

3.3 VDCNN

Our VDCNN references the architecture of Conneau et al. which has proven to be successful in the field of text processing. In the future, we will be looking to test different configurations and adjustments to the model in order to improve performance; but for now, we are starting by writing our own implementation of VDCNN-9 as noted in the paper. The architecture is as followed:

Each text comment is represented as a sequences of characters, which is either truncated or padded to a fixed length of 1024. For each char, we look up an embedding of size 16. The embeddings go through a 1D convolutional layer with window size 3 and 64 output channels. Now the data matrix has size 64 $\times$ 1024, and we will pass it through 4 covoluntional blocks, each containing two 1D convolutional layers. The idea of each convolutional block is to double the output channel after which a pooling layer will reducing the number of embeddings by half through downsampling. This prevents the exponential growth of the number of parameters and keeps the memory usage consistent. The internal structure of a convolutional block with input dimension D is: Conv1d(D, 2D, 3) –> Batch Norm –> ReLU –> Conv1d(2D, 2D, 3) –> Batch Norm –> ReLU (the third parameter of Conv1d is window size). Note that the four convoluntional blocks contains 8 convolutional layers; adding the initial convolutional layer, we reach a total depth of 9. After the convolutional blocks, the output is of size 512×128. We now apply a k-max pooling layer with $k = 8$ to reduce the dimension to 512×8. This is flattened into a vector with size 4096 and passed through three linear layers in this order: Linear(4096, 2048) –> ReLU –> Linear(2048, 2048) –> ReLU –> Linear(2048, n), where n is the number of classes. Currently, we have set $n = 2$, meaning that either a comment is toxic or clean; but we can create more categories for the text labels such as "insult", "threat", and "identity hate" comments. Finally, we put the output through the softmax function to predict the probability that a comment belongs to each class:

$$P(\text{Class} = x_j) = \text{Softmax}(x_j) = \frac{\exp(x_j)}{\sum_{i=1}^{n} \exp(x_i)}$$

The model is trained using CE loss and Adam optimizer. We implemented this model by hand so that we could adapt it to our use cases.

4 Experiments

4.1 Data

In 2017 Google Jigsaw published a dataset on Kaggle labeled "Toxic Comment Classification Challenge". The dataset includes 223,549 user comments, annotated with labels that fall into one or more of the following categories: clean, toxic, obscene, insult, identity hate, severe toxic, and threat [2]. The dataset is from a real world example, and a majority (about 200,000) of the samples in the dataset fall under the "clean" label. Our classifier's intent is to be able to distinguish between these different categories at two levels of granularity.

The task we are going to attempt is to construct a binary classifier to distinguish between toxic speech and clean speech. For this we will consider all elements that fall under any one of the categories of toxic, obscene, insult, identity hate, severe toxic, and threat as toxic speech and clean as the other label.

The dataset itself provides a standardized test set to compute accuracies on. However to account for overfitting, we wil split our training set randomly into an 80/20, test/dev split. The evaluation of the model on the dev split at each epoch will be used to choose the final model that will then be tested on the test set.

4.2 Evaluation Method

We evaluate our models and approaches on two key criteria: the F1 and test accuracy. The F1 score is a measure of the models accuracy as it is a harmonic average of precision and recall. We compute the F1 score for each of the outputted categories and average the F1 scores for each category to compute an composite F1 score to represent the overall precision and recall of the system. The F1 score is a

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

standard measurement used for NLP classification tasks, which we can use to compare with existing models. The second evaluation method we use is the accuracy of the model's predictions on the test set. This is our stronger metric to test as it gives us a simple and direct measure of the ability of our models to detect toxic comments. Together the F1 score and test accuracy paint a good picture about the performance of the models.

4.3 Experimental Details

4.3.1 Architectures

Our baseline model consists of a logistic regressor trained on word embeddings. This is a simple method that takes as input the average of the word embeddings and outputs a probability of a specific class membership.

We consider two deep learning/ neural network based approaches to solving our task: VDCNN (Conneau et al. [2016]) and GRU/LSTM based method (Chung et al. [2014]).

We initialize a VDCNN-9 as constructed in the paper with maxpooling blocks, and no residual connections. VDCNN has an initial convolutional layer and four blocks of convolutional layers (each block has the same number, but vary between types of VDCNNs, similar to ResNets). Thus, a VDCNN-9 has an initial convolutional layer and two convolutional layers in each subsequent block. We use the standard VDCNN (Conneau et al. [2016]) on the Jigsaw dataset.

We also initialize a bidirectional GRU and LSTM with attention as an alternate classifier for our dataset (Chung et al. [2014], Lin et al. [2017]). The rnn-based model has a hidden size of 500, and two layers. The loss function and gradient steps are evaluated as detailed above. The bidirectional GRU/LSTM takes as input the word embeddings, and its output as a "sentence embedding". The sentence embedding is then fed through the attention layer as detailed by Lin et al. [2017] and then fed into a fully connected classifier.

We modify both networks to have the option to classify on word embeddings instead of character embeddings and thus give ourselves the option to train faster and also use a consistent source of embeddings across trials and architectures. The pretrained word embeddings we use are 300 dimensional word vectors trained on Wikipedia 2017, UMBC webbase corpus and statmt.org news dataset (16B tokens, 1M word vectors) (Mikolov et al. [2018]). These embeddings also allow us to use sub-word embeddings to generate embeddings from words that would otherwise be classified as unk in the dataset.

Each of these networks and approaches are implemented and executed using Pytorch 1.0.1.

4.3.2 Hyperparameters

For all experiments we are running with VDCNN, we use a batch size of 128, SGD with a momentum of 0.9 and weight decay of 1e-4. Similar to the paper we start with a learning rate of 0.01 for 15 epochs with milestones for learning rate decay of a factor of 10 at epochs 3,6,9,12, and 15.

For the GRU and LSTM based models, we use a batch size of 64, SGD with a momentum of 0.9 and weight decay of 1e-4 with gradient clipping. We start with a learning rate of 0.001 with a decay by factor of 10 on plateaus.

5 Results

Our experiments evaluated the performance of the described models on our task with many variants. As mentioned for each model, we computed the F1 score and the accuracy of the model on the test set in the binary classification setting.

We note that when it comes to accuracy, the Bi-LSTM with attention and pretrained embeddings seems to be the top contender with a test accuracy score 0.989. The LSTM adn GRU with attention and FastText embeddings also seems to achieve the highest F1 score above the other three trials.

	F1 Score	Accuracy
Baseline: Logistic Regression	0.44	0.967
Bi-GRU	0.57	0.971
Bi-GRU (FastText Embeddings)	0.61	0.974
Bi-GRU (Attention + FastText Embeddings)	0.66	0.987
Bi-LSTM	0.60	0.975
Bi-LSTM (FastText Embeddings)	0.62	0.980
Bi-LSTM (Attention + FastText Embeddings)	**0.66**	**0.989**
VDCNN-9 (FastText Embeddings)	0.62	0.975
VDCNN-17 (FastText Embeddings)	0.62	0.978

5.1 Pretrained Embeddings

A key observation is the effect of the use of pretrained, FastText embeddings on the final computed scores. We find that using pretrained word embeddings actually boosts the accuracy of the trained models by a nontrivial amount. We can see this effect by contrasting the accuracies of the Bi-GRU/LSTM and the Bi-GRU/LSTM with pretrained embeddings (Figure 1). The addition of embeddings seems to give a substantial boost in accuracy to the underlying models.

We believe these pretrained embeddings offer a significant boost in accuracy for two central reasons:

1. The FastText pretrained embeddings are able to calculate subword embeddings. With a manual survey of our dataset we noticed that the frequency of tokens was relatively small, with about 38A further inquiry into this show that tokens like "sucklol" and "f**kings**t" (sampled from our dataset) are counted as discrete tokens. This is not a surprise due to our dataset being sourced from a number of social media forums. Our data preprocessing removes these tokens from the vocabulary.

   By adding the pretrained embeddings by FastText, we are able to compute embeddings for these types of tokens, encoding more of the meaning within the post.
2. The pretrained embeddings are also trained on an extremely large corpus. In fact the embeddings are trained on 16B tokens, 1M word vectors (Mikolov et al. [2018]), which is significantly larger than the training set we have available to us.

5.2 Attention

We notice that the addition of scaled dot-product, self-attention on the GRU or LSTM gives marginal increases in accuracy (Figure 1). For example the addition of attention boosts the test accuracy by about a 1.3We postulate that this boost is due largely to the ability of attention to prioritize specific parts of the sentence over other parts. This may be particularly helpful in this classification task because in identifying hate speech, only a few parts of the sentence may be necessary. For example in the sentence "go home you idiot", the part of the sentence that matters is the negative assertion ("idiot") and the part of the sentence that makes it pointed to a person or group ("you").

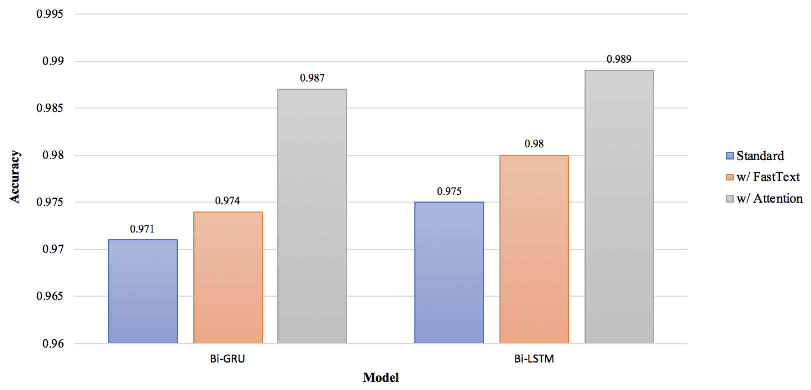

Figure 1: Effects of pre-trained embeddings and attention on test accuracy

5.3 Inference Speeds

One of the largest barriers to using deep NLP methods in industry is the computational cost associated with deep learning. To get a sense of the magnitude of this cost, we ran a few experiments to estimate the inference speed of each. We use the same preprocessing on each of these models (with FastText embeddings) and evaluate a forward pass with one document for 10 trials of 100,000 runs each. With this we calculate the average runtime and the standard deviation of that runtime.

	Inference Speed (ms)	**St. Dev.**
Baseline: Logistic Regression	2.03 ms	0.02 ms
Bi-GRU (Attention + FastText Embeddings)	22 ms	1.02 ms
Bi-LSTM (Attention + FastText Embeddings)	28 ms	1.19 ms
VDCNN-17 (FastText Embeddings)	26 ms	0.65 ms

Each trial was controlled on the same machine with no other processes substantial processes running. The tests were run on a 6 core i7 Intel CPU.

As shown in Figure 2, the logistic regression model actually has a latency of 2.03 ms which is about 11x faster than its RNN and CNN based counterparts. This does not come as a surprise at all due to the extremely efficient and small computation required to run the forward pass of a logistic regressor compared to a 17 layer CNN or a RNN.

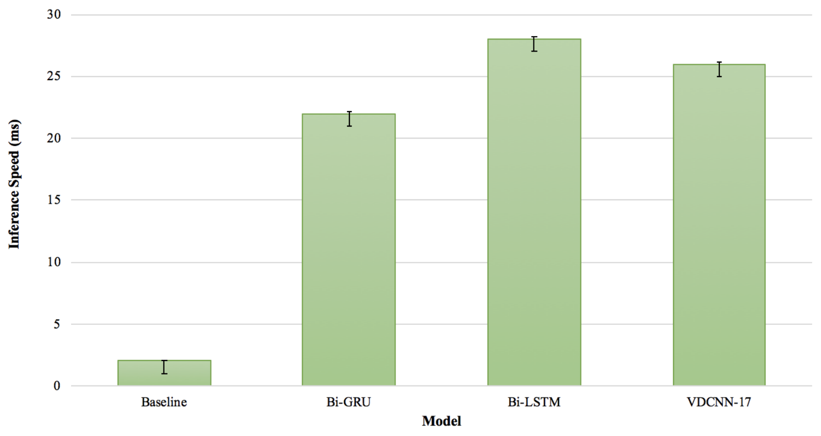

Figure 2: Inference speed of our deep learning models compared to baseline

5.3.1 Cascading Model

As one of our future directions, we propose a simple cascading model that combines multiple of our tested models to optimize for accuracy and speed in the average case. Cascading models use intermediate steps and confidence scores at each step. Each subsequent step has a more substantial computation cost. We test a simple cascading model composed of a logistic regression as the first step.

If the logistic regressor outputs a value between 0.3 and 0.7 (as a simple heuristic for the logistic regressor being unsure about the outputs), we feed the original input into the Bi-LSTM with attention and pretrained model (see Figure 3 below). On the test set, this cascading model gives us substantial speedups with an average latency of **5.18 ms**, still almost 6 times faster than using the Bi-LSTM w/ attention and embeddings alone. Only about 31% of the documents from the test set had a mediocre score on the logistic regressor and had to be pushed to the LSTM.

This also boosts the accuracy of the model to a higher accuracy than that of only a logistic regressor to **0.973**. This shows a promising result for the detection of hate speech in an efficient and scalable manner.

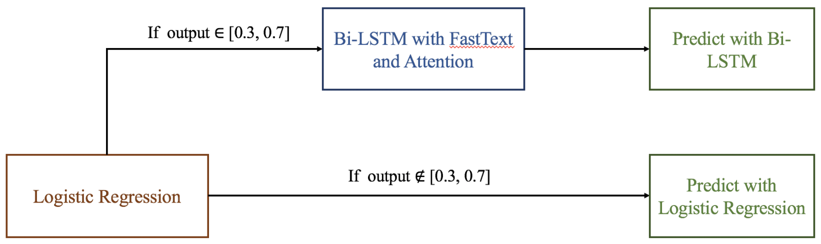

Figure 3: Diagram of a simple cascading model

6 Conclusion

From our experiments, we have shown that deep learning models can perform very well in the task of toxic comment detection. Both the Bi-GRU/LSTM and VDCNN models were able to produce higher F1 and accuracy compared to a fairly strong (albeit simple) logistic regression baseline. We have addressed the issue of computational time and cost for deep learning models by analyzing their inference speeds. Indeed, a huge disadvantage of CNNs and RNNs are there sluggish training and testing time compared to "classical methods." As such, we propose adopting a cascading model, which is made possible by the solid performance of time-efficient baseline models. The idea is that if a fast model like logistical regression or SVM is sufficiently confident in its prediction, then we use it; otherwise, we leave the decision to our deep learning model, which takes longer but is more accurate.

A limitation of our project is that our models are trained on Google Jigsaw's toxic comment dataset, which contains mostly long comments and seven types of toxic labels. As a result, our models are tailored to the dataset and can predictions for each of the seven categories. While we achieved high accuracy on our test set, we may not be able to achieve the same level of performance, if, say, the model is applied to tweets. In general, it is difficult to find sizable, cleanly labelled datasets for toxic comments, so we would have to compile and preprocess much more data if we were to build on this project. Moreover, we were not able to run a thorough analysis on the effect of depth in VDCNNs because the training time grows exponentially and we were only able to train up to a VDCNN-17. Ideally, we would like increase the depth up to 49 or higher to see if deep networks can further boost the accuracy.

For future work, there are many models and combination of deep learning paradigms that we would like to test and explore such as combining convolutional and recurrent neural networks, as well as state-of-the-art SVM models and feature extraction mechanisms used for toxic speech detection. We think that the cascading model shows promise. While we only implemented a simple version, it would be interesting to further optimize the criteria to determine the confidence of the baseline and to improve both the efficiency and accuracy of the model.

7 Code

This entire project's codebase can be found here: https://goo.gl/GgPnG1

References

Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014. URL `http://arxiv.org/abs/1412.3555`.

Alexis Conneau, Holger Schwenk, Loïc Barrault, and Yann LeCun. Very deep convolutional networks for natural language processing. *CoRR*, abs/1606.01781, 2016. URL `http://arxiv.org/abs/1606.01781`.

Thomas Davidson, Dana Warmsley, Michael Macy, and Ingmar Weber. Automated hate speech detection and the problem of offensive language. In *Proceedings of the 11th International AAAI Conference on Web and Social Media*, ICWSM '17, pages 512–515, 2017.

Zhouhan Lin, Minwei Feng, Cícero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. A structured self-attentive sentence embedding. *CoRR*, abs/1703.03130, 2017. URL `http://arxiv.org/abs/1703.03130`.

Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. Advances in pre-training distributed word representations. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*, 2018.