Toxic Comment Categorization using Bidirectional LSTMs with Attention

Michael Baumer
Department of Physics
Stanford University
mbaumer@stanford.edu

Anthony Ho
Department of Applied Physics
Stanford University
ahho@stanford.edu

Abstract

Toxic online comments are a challenge for websites that accept user-submitted content, and human moderation is a psychologically difficult task that does not scale well. We present a framework for using deep learning to automatically identify and categorize toxic online comments into 6 overlapping sub-categories. We employ a bidirectional LSTM architecture with an attention mechanism to achieve a mean ROC AUC of .9778 on a blind test set. We use the normalized attention weights to visualize the activation of our network in response to each word within a sentence, and find qualitative agreement with human intuition.

1 Introduction

Many websites that display user-submitted content must deal with toxic or abusive comments. Human moderation may carry psychological risk to the moderators [1], and is difficult to implement at scale. Potential automated solutions will require not just binary classification (acceptable vs. blocked) but fine-grained comment classifications to maintain civility without interfering with normal discourse and provide explanations to users when their posts are censored. Different websites may wish to have different policies for dealing with different kinds of toxic content. For example, a website may want to allow insulting comments while still blocking threats and identity-based hate speech.

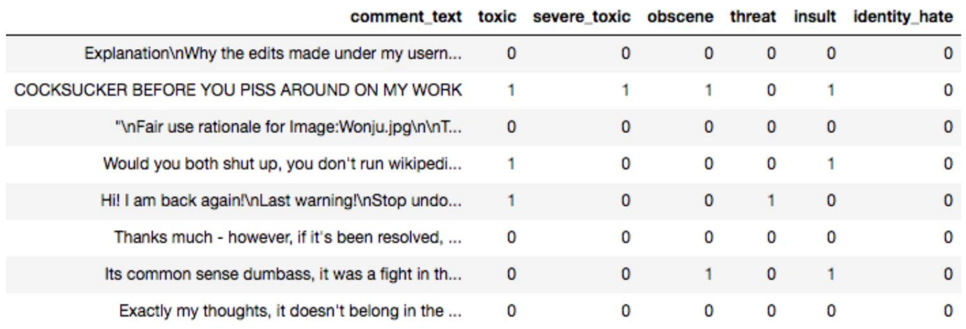
In this project, we attempt to improve the identification and fine-grained classification of toxic online comments in a dataset provided by a Kaggle challenge [1] Our dataset consists of 159,571 comments from Wikipedia talk page edits which have been labeled by human raters for the presence of toxic behavior. The 6 types of toxicity are: `toxic`, `severe_toxic`, `obscene`, `threat`, `insult`, and `identity_hate`. A sample of entries from the dataset is shown in Figure 1.

In this report, we will describe a baseline approach to this multi-label classification task using a three-layer feed-forward neural network with averages of word-vector sentence embeddings as inputs. We will then describe our experiments with uni- and bidirectional long short-term memory (LSTM) and gated recurrent unit (GRU) architectures, and demonstrate a significant improvement in performance with the addition of an attention mechanism. Finally, we present the results of applying our framework to the dataset, including visualizations of the model's weighting of input words via the attention mechanism.

2 Background and Related Work

The earliest work on machine learning-based detection of online harassment can be traced back to Yin *et al.* [2], where they fed content, sentiment and contextual features into a support vector

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



comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
Explanation\nWhy the edits made under my usern...	0	0	0	0	0	0
COCKSUCKER BEFORE YOU PISS AROUND ON MY WORK	1	1	1	0	1	0
"\nFair use rationale for Image:Wonju.jpg\n\nT...	0	0	0	0	0	0
Would you both shut up, you don't run wikipedi...	1	0	0	0	1	0
Hi! I am back again!\nLast warning!\nStop undo...	1	0	0	1	0	0
Thanks much - however, if it's been resolved, ...	0	0	0	0	0	0
Its common sense dumbass, it was a fight in th...	0	0	1	0	1	0
Exactly my thoughts, it doesn't belong in the ...	0	0	0	0	0	0

Figure 1: A selection of comments from our training dataset, showing the large diversity of the corpus and the ability of a comment to have multiple labels. Also, note that while `toxic` is a superset of `severe_toxic`, it is not a superset of the other 4 labels.

machine to classify instances of harassment. More recently, a paper by the group that created the dataset used in the Kaggle challenge [3] focused on binary identification of toxic comments (no fine-grained classification). They found success with relatively simple n-gram NLP methods, and left more complex methods (like LSTMs) as future work.

In last year's CS224n class, a group of students looked at the same Wikipedia corpus and performed binary classification (personal attacks vs non personal attacks) using an LSTM architecture and word-level embeddings, a convolutional neural network (CNN) with word-level embeddings, and a CNN with character-level embeddings [4]. They found the CNN with character-level embeddings to be the most successful algorithm.

For our project, we decided to take a different approach and investigate whether applying attention to this problem could improve the performance of word-level embeddings, inspired by Yang *et al.* [5].

3 Approach

3.1 Data and Preprocessing

To convert the raw text data into usable form, we tokenize the input using the `nltk` package [6]. Since each of the comments is of a different length, we pad or crop the token lists from each comment to a uniform length and mask the $\langle$pad$\rangle$ tokens in the RNN output layer (or attention layer, once we started using it).

To represent each word token in the input comments, we perform word embedding using the 300-dimensional pretrained global vectors for word representation (GloVe) [7], which was trained using aggregated global word-word co-occurrence statistics on 6 billion tokens from Wikipedia and Gigaword corpora [8] with a vocabulary size of 400k.

Finally, we create input vectors of `int32` word indices using the `word2id` mappings included with the pretrained GloVe vectors. We held back a random 30% of the training data to use as a dev set for testing the out-of-sample performance of our models.

Many offensive words posted online are not a part of the 400k words that form the GloVe vocabulary, and therefore map to the $\langle$unk$\rangle$ token. Since identifying such offensive words should improve our performance, if a token contains as a substring any member of a list of common vulgarities that are not commonly found as substrings of inoffensive words (refer to our code for details), we map the composite token to the root vulgarity.

Another observation about our dataset is the large imbalance in number of examples of each label (see Table 1). We deal with this problem by weighting the contributions of labels inversely to their occurrence in the training data, normalized to 1 for toxic comments (the most common label). These weights cause larger gradients to backpropagate from mistakes on the rarer labels. In this way, we learn more from each example of the more uncommon labels.

Category	# of examples	% of dataset
No label	143346	89.8%
Toxic	15294	9.6%
Severe Toxic	1595	1.0%
Obscene	8449	5.3%
Threat	478	0.3%
Insult	7877	4.9%
Identity Hate	1405	0.9%

Table 1: Counts of occurences of each label category, and percentages of comments in the dataset with the given label (these do not sum to 100% because a single comment can have multiple labels).

3.2 Baseline model

We implemented a fully-connected feedforward neural network as our baseline model. The inputs of the model are the average of the 300d GloVe embeddings of all tokens in each comment. Note that no comment padding is necessary in this model, as the averaging procedure reduces comments of any length to a uniform size.

The neural network consists of 3 hidden layers with ReLU activation function [9], with 30, 20, 10 hidden units in each hidden layer, respectively. Since our goal is to perform multi-label classification (i.e. the toxicity labels are not mutually exclusive), the outputs of the last hidden layer is fed into a sigmoid output layer (rather than a softmax layer, which would force the 6 class probabilities to add up to 1) with 6 units, which correspond to the predicted probabilities of each of the 6 labels. The cross entropy function is used as the loss. The neural architecture can be defined as follows:

$$\begin{aligned}
\boldsymbol{h}_1 &= \text{ReLu}(\boldsymbol{x}\boldsymbol{W}_1 + \boldsymbol{b}_1) \\
\boldsymbol{h}_2 &= \text{ReLu}(\boldsymbol{h}_1\boldsymbol{W}_2 + \boldsymbol{b}_2) \\
\boldsymbol{h}_3 &= \text{ReLu}(\boldsymbol{h}_2\boldsymbol{W}_3 + \boldsymbol{b}_3) \\
\hat{\boldsymbol{y}} &= \sigma(\boldsymbol{h}_3\boldsymbol{W}_4 + \boldsymbol{b}_4) \\
J &= CE(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\sum_{i=1}^{6} y_i \log(\hat{y}_i)
\end{aligned}$$

where

$$\boldsymbol{x} \in \mathbb{R}^{B\times 300}, \boldsymbol{h}_1 \in \mathbb{R}^{B\times 30}, \boldsymbol{h}_2 \in \mathbb{R}^{B\times 20}, \boldsymbol{h}_3 \in \mathbb{R}^{B\times 10}, \hat{\boldsymbol{y}} \in \mathbb{R}^{B\times 6}, \boldsymbol{y} \in \mathbb{R}^{B\times 6}$$

and B is the batch size. In this baseline model and throughout this work, all bias terms are initialized to zero, while all weight matrices are initialized using Xavier initialization [10].

With this baseline model, we found relatively good dev set performance as measured by the ROC curves for each toxicity type (see Figure 2). The mean ROC AUC across the 6 labels on the blind test set was 0.9490, which is good performance for a baseline, but placed us 0.04 behind the leading Kaggle entry. The similarity between the train and dev set curves shows that we were not suffering from severe overfitting.

Since the number of true negatives (non-toxic comments) in our dataset is large (see Table 1), and ROC curves plot the true positive rate ($TPR = TP/(TP + FN)$) vs. the false positive rate ($FPR = FP/(FP + TN)$), their appearance is biased by this imbalance in our dataset. However, since the mean AUC for these ROC curves was the official metric of the Kaggle challenge, we used these as our primary evaulation metric.

For a more direct measure of our classifier's performance for each label, we also plotted the precision vs. recall performance of our baseline classifier (shown at right in Figure 2). Here we see that our performance is better for the labels that have more training examples.

3.3 Recurrent Architectures

Our baseline model achieved relatively good performance, but we thought that we could improve it by building a recurrent neural network (RNN) model that is able to use positional information of

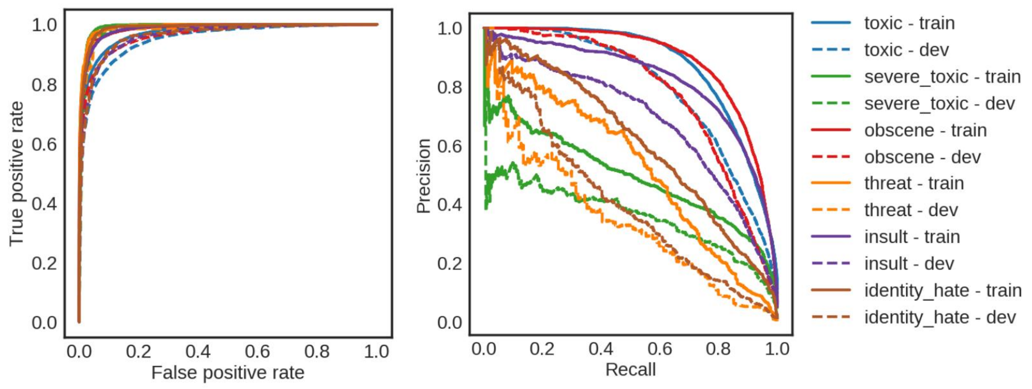


Figure 2: The ROC and precision-recall curves for our baseline model's performance on each toxicity type as classified by our baseline model.

each of the words rather than an average over the entire sentence. We found that RNNs with GRU and LSTM cells were superior to a vanilla RNN architecture, but the differences between LSTM and GRU were minor, so we decided to focus on networks with LSTM cells.

We implemented our models in Python using tensorflow [11]. For the LSTM cells that form the backbone of our architecture, we use the tensorflow implementation of the original LSTM equations from [12], shown below:

$$\begin{aligned}
\boldsymbol{i}_t &= \sigma\left(\boldsymbol{x}_t \boldsymbol{W}^{(i)} + \boldsymbol{h}_{t-1}\boldsymbol{U}^{(i)}\right)\\
\boldsymbol{f}_t &= \sigma\left(\boldsymbol{x}_t \boldsymbol{W}^{(f)} + \boldsymbol{h}_{t-1}\boldsymbol{U}^{(f)}\right)\\
\boldsymbol{o}_t &= \sigma\left(\boldsymbol{x}_t \boldsymbol{W}^{(o)} + \boldsymbol{h}_{t-1}\boldsymbol{U}^{(o)}\right)\\
\tilde{\boldsymbol{c}}_t &= \tanh\left(\boldsymbol{x}_t \boldsymbol{W}^{(o)} + \boldsymbol{h}_{t-1}\boldsymbol{U}^{(o)}\right)\\
\boldsymbol{c}_t &= \boldsymbol{f}_t \circ \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \circ \tilde{\boldsymbol{c}}_t\\
\boldsymbol{h}_t &= \boldsymbol{o}_t \circ \tanh(\boldsymbol{c}_t)
\end{aligned}$$

The LSTM architecture allows information to flow long distances across the unrolled network (if $f_t = 1$ and $i_t = 0$, the previous state flows straight through the network), but also prevent exploding gradient problems (o_t modulates strength of output at each step, and $f_t = 0$ allows prior states to be forgotten so backpropagation need not continue all the way back to $t = 0$). We also investigated the use of GRU cells (which have similar properties) for this task and found comparable performance to LSTM cells, so we omit their equations here [13].

For a unidirectional LSTM architecture, it is possible to use the hidden state of the final time step as the overall output state $\tilde{\boldsymbol{h}}$ that is fed into the final output layer. We found better performance, however, when we use the element-wise average over the hidden states of all time steps (masking any hidden states at time steps where the $\langle \text{pad} \rangle$ token was input) as $\tilde{\boldsymbol{h}}$, the input to the final layer of our architecture.

With a bidirectional LSTM architecture, we have one set of parameters for a forward-unrolled LSTM and a separate set of parameters for a backward-unrolled LSTM that reads input token lists starting at the end rather than the beginning. For each token position, we therefore have two output states at each time step, which we either take the average of before averaging across all time steps, or concatenate to form a single joint output state which is then feed into the the attention layer described below.

To prevent overfitting, we add regularization to the overall output state of the LSTMs (or attention layer) by applying dropout [14] with $p_{\text{drop}} = 0.5$:

$$\boldsymbol{h}_{\text{dropout}} = \text{Dropout}(\tilde{\boldsymbol{h}}, p_{\text{drop}})$$

Finally, we use a fully-connected layer with sigmoid activations to convert the output of the dropout layer to the probabilities of each label:

$$\hat{\boldsymbol{y}} = \sigma\left(\boldsymbol{h}_{\text{dropout}}\boldsymbol{U} + \boldsymbol{b}\right)$$

where $\boldsymbol{h}_{\text{dropout}} \in \mathbb{R}^{B\times H}, \boldsymbol{U} \in \mathbb{R}^{H\times 6}$ and $\hat{\boldsymbol{y}} \in \mathbb{R}^{B\times 6}$, with H as the hidden state size of the RNN/LSTM/GRU and B as the minibatch size.

3.4 Attention Mechanism

To extract and add importance to informative words that might be relevant to the classification, we utilized a word attention mechanism as described by Yang *et al.* [5]. We implemented the attention mechanism by modifying publicly available code [15] in order to allow for masking comments with variable length. In our neural architecture, the attention mechanism can be optionally applied to reduce the set of hidden vectors $\{\boldsymbol{h}_t\}$ from every RNN/LSTM/GRU time steps $t = \{1, \ldots, T\}$ to an attention output vector $\tilde{\boldsymbol{h}}$, as opposed to the averaging as described above. The attention output vector $\tilde{\boldsymbol{h}}$ is then fed as the input to the dropout layer described above. The mathematical formulation of this attention mechanism is described below:

$$\begin{aligned}
\boldsymbol{v}_t &= \tanh(\boldsymbol{h}_t\boldsymbol{W}_a + \boldsymbol{b}_a)\\
s_t &= \boldsymbol{v}_t\boldsymbol{u}_a^\top\\
\alpha_t &= \frac{\exp(s_t)}{\sum_{t=1}^{T}\exp(s_t)}\\
\tilde{\boldsymbol{h}} &= \sum_{t=1}^{T}\alpha_t\boldsymbol{h}_t
\end{aligned}$$

In this attention mechanism, the hidden vectors of each word $\boldsymbol{h}_t$ from the recurrent time step t is feed into a single tanh non-linearity to produce a new word representation $\boldsymbol{v}_t$. An attention score s_t is then computed by measuring the dot product similarity between $\boldsymbol{v}_t$ and the word context vector $\boldsymbol{u}_a$. The attention weights α_t can then be computed by normalizing the attention score across all time steps via a softmax function. Finally, an attention output vector $\tilde{\boldsymbol{h}}$ is computed as the weighted sum of the original hidden states $\boldsymbol{h}_t$ of each word, weighted by the normalized attention weights.

4 Experiments

4.1 Experiment Configurations

We ran a variety of models with a range of hyperparameter settings on an Azure NV6 node. We focused on expansions to our model rather than a systematic search for optimal hyperparameter settings, keeping the learning rate for our Adam optimizer [16] fixed at 0.0005. The steady decline of our training loss (as shown at left in Figure 3) shows that this choice of learning rate was reasonable.

Table 2 shows the results for a set of experiments we did before implementing the attention layer, to assess the improvement from using larger hidden states, more layers, and uni- vs. bidirectional architectures. We found that bidirectional architectures were in all cases better than unidirectional, more than 1 layer in each direction (if bidirectional) led to overfitting, and LSTM and GRU cells gave nearly equivalent performance. We therefore used a single layer, bidirectional LSTM as our fiducial model for implementing attention.

Experiments took anywhere from 1-3 hours, depending on the size of the model and number of epochs. Initially, we ran all models for 50 epochs, but found that the model began to overfit after $\sim$ 10 epochs (see Figure 3). We therefore implemented early stopping, where the model would progressively save the model after each epoch if it had better dev set performance (measured by mean column-wise ROC AUC) than any epoch before it.

We also implemented gradient clipping, to ensure that our gradients didn't explode during back-propagation, but due to the LSTM architecture (see discussion in section 3.3), the norm remained low enough during training that clipping was unnecessary (see right hand panel of Figure 3).

Cell type	Directionality	# layers	Hidden size	Dev ⟨ROC AUC⟩	Dev ⟨AP⟩
LSTM	Single	1	50	.9610	.5009
LSTM	Single	1	100	.9683	.5473
LSTM	Single	2	50	.9546	.4470
LSTM	Bidirectional	1	50	.9747	.5788
LSTM	Bidirectional	2	50	.9747	.5882
GRU	Single	1	50	.9659	.5180
GRU	Bidirectional	1	50	.9755	.5837
GRU	Bidirectional	2	50	.9761	.5619

Table 2: This table shows the results of a hyperparameter search among cell type, hidden state size, number of layers, and directionality for our RNN model (without attention)

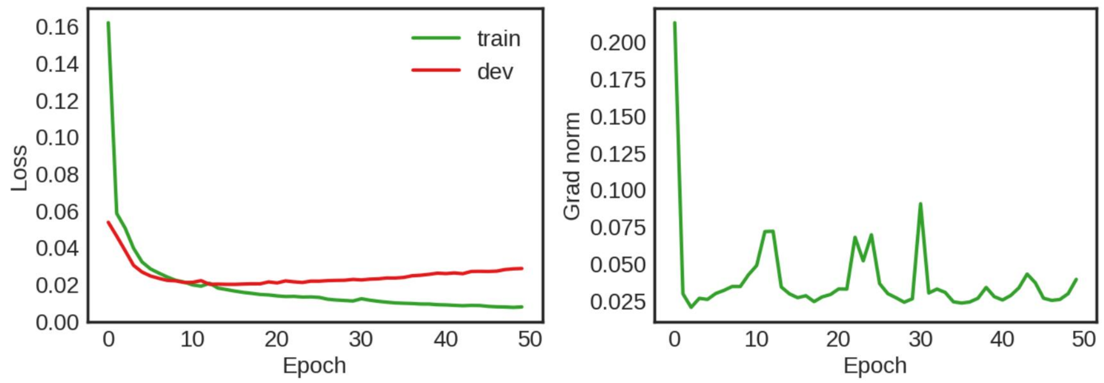


Figure 3: At left, a typical loss function from our training runs. The increase in the dev loss after ~ 10 epochs indicates overfitting, so we use early stopping to save the best model before overfitting begins. At right, a plot of the grad norm during training showing that we did not have any exploding gradient problems.

4.2 Evaluation

We set up a evaluation pipeline to automatically plot ROC curves and precision-recall curve for classification of each toxicity type. The pipeline also computed the ROC AUC and average precision (AP) for classification of each toxicity type, and the mean column-wise ROC AUC and mean column-wise average precision across all toxicity types. We submitted our model predictions on a test set of 153,164 comments whose true labels were withheld by Kaggle. Our online submissions returned the mean column-wise ROC AUC (the official evaluation metric in the Kaggle challenge) for our test set predictions.

Model	Dev ⟨AP⟩	Dev ⟨ROC AUC⟩	Test ⟨ROC AUC⟩
Baseline	.5610	.9669	.9490
LSTM	.5473	.9683	.9435
Bidirectional LSTM	.5882	.9747	.9611
Bidirectional LSTM + Attention	**.6695**	**.9859**	**.9778**

Table 3: A summary table of the results from the primary models we investigated, showing the mean average precision and mean ROC AUC from the dev set, and the mean ROC AUC from the blind test set.

The results for our four primary models are shown in Table 3, highlighting the substantial improvement in our multi-label classification resulting from our implementation of word attention.

Looking at Figure 4, we see that the overall performance of our final model (bidirectional LSTM + attention) is very good, with the ROC very close to the upper-left corner representing perfect classification. In the right-hand panel, we see that the classes for which we have the weakest per-

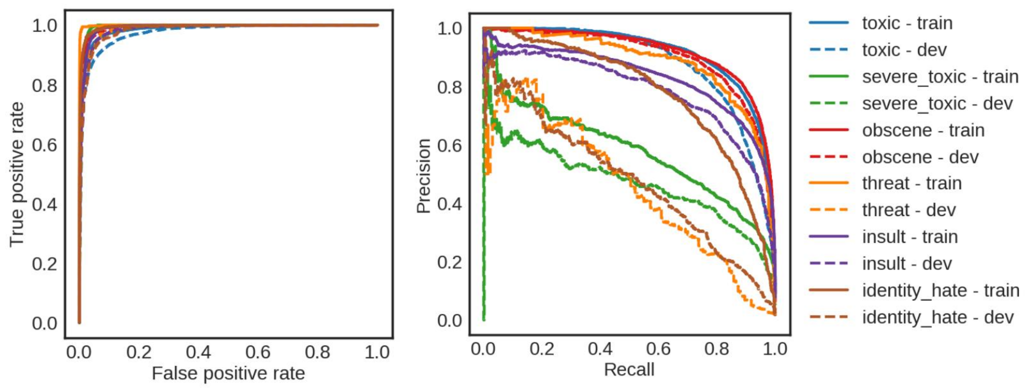


Figure 4: The ROC and precision-recall curves for our final model.

formance are the ones for which we have the fewest training examples (`threat`, `severe_toxic`, and `identity_hate`)

Some amount of overfitting is present, as evidenced by the dotted dev lines being displaced from the solid lines representing the performance on the training data. One way to further improve this model would be increasing the strength of the regularization beyond the dropout we have already applied.

4.3 Qualitative Evaluation via Attention

With its attention mechanism, our model is capable of computing the normalized attention weights α_t for every word given a sentence. For diagnostic purposes, we can visualize the attention weights of selected comments in order to qualitatively understand how the classifier classified comments the way it did. For example, here are the visualizations of the attention weights of some example true positives - in this case, comments that got successfully classified as `toxic` and `threat`:

```
i 'm also a sock puppet of this account ... suprise ! ! -sincerely , the man
that will track you down from the internet and kill you
```

```
important you and your family shall burn ! you have been warned , if you do
troll muzemikes talk page i will slaughter you in your sleep . bye ...
```

The attention weights successfully highlight the words and phrases that sound threatening to the other users.

Here is an example of a true negative:

```
that 's a rather feeble premise to argue showing an excessive number of photos
prominently showing 24 different women 's vaginas , compared to one human male
penis 3/4 of the way down the page on the equivalent article for male genitalia
. this demonstrates the overwhelming male bias in editorial content on
wikipedia . 86.13.182.103
```

Even though the attention weights highlight some potentially vulgar words, the classifier successfully classified it as a negative.

On the other hand, here is an example of a false positive:

```
" is this really a neutral article ? ca n't we at least agree that this article
is not neutral ? after all , another editor has referred to the idea of
transphobia as a " " mindfuck " " idea . at the very least , should n't this
have a " " neutrality in dispute " " tag attached to it ? ? "
```

The model identifies a couple of words that could potentially be toxic, as highlighted by the attention weights, but does not pick up that the commenter is quoting someone else to make a reasonable argument, and therefore misclassifies it as `toxic` and `obscene`.

5 Conclusions and Future Work

We have implemented a Tensorflow framework based on a bidirectional LSTM with an output attention layer that successfully performs multi-label classification of various subtypes of toxic online comments. Our final mean ROC AUC performance on the blind test set is .9778, which places us within .011 of the leading Kaggle entry. The addition of an attention layer to our bidirectional LSTM architecture significantly improves performance both in terms of mean column-wise ROC AUC (.9611 to .9778) and mean column-wise average precision (.5882 to .6695).

Our performance is primarily limited by the embedding of new or rarely-used obscenities to the $\langle$unk$\rangle$ token by GloVe. We tried to ameliorate this by mapping unknown composite obscenities to the embedding of the root obscenity in our preprocessing, but further such preprocessing to eliminate unknown tokens, or the use of character-level embeddings and/or convolutional methods might further improve performance, since they would be able to recognize toxic character patterns within an unknown word. In addition, stronger regularization beyond dropout might help increase the performance of our model without overfitting.

References

[1] Madrigal, A. (2017) The Basic Grossness of Humans. *The Atlantic* https://www.theatlantic.com/technology/archive/2017/12/the-basic-grossness-of-humans/548330/

[2] Yin, D., *et al*. (2009) Detection of Harassment on Web 2.0. *Proceedings of the Content Analysis in the WEB 2.0 (CAW2.0) Workshop at WWW2009*.

[3] Wulczyn, E., Thain, N., & Dixon, L. (2017) Ex Machina: Personal Attacks Seen at Scale. *Proceedings of the 26th International Conference on World Wide Web*.

[4] Chu, T., Jue, K., & Wang, M. (2017) Comment Abuse Classification with Deep Learning. In *CS224n Final Project Reports*

[5] Yang, Z., *et al*. (2016) Hierarchical Attention Networks for Document Classification. *Proceedings of NAACL-HLT 2016* 14801489.

[6] Bird, S., Loper, E., & Klein, E. (2009) *Natural Language Processing with Python.* OReilly Media Inc.

[7] Pennington, J., Socher, R., & Manning, C. (2014) GloVe: Global Vectors for Word Representation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*.

[8] Parker, R., *et al*. (2011) English Gigaword Fifth Edition LDC2011T07. DVD. Philadelphia: Linguistic Data Consortium.

[9] Nair, V. & Hinton, G. E. (2010) Rectified linear units improve restricted boltzmann machines. *Proceedings of the 27th International Conference on Machine Learning*, Omnipress, USA, 807-814.

[10] Glorot, X., Bengio, Y. (2010) Understanding the difficulty of training deep feedforward neural networks. *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, PMLR 9:249-256.

[11] Abadi, M., *et al*. (2015) TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

[12] Hochreiter, S. & Schmidhuber, J. (1997) Long Short-Term Memory. *Neural Computation*, 9(8):1735-1780.

[13] Cho, K., *et al*. (2014) Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation, arXiv:1406.1078.

[14] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014) Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* 15 1 1929-1958.

[15] Ivanov, I. (2018) Tensorflow implementation of attention mechanism for text classification tasks. https://github.com/ilivans/tf-rnn-attention/

[16] Kingma, D. P. & Ba, J. (2014) Adam: A Method for Stochastic Optimization, arXiv:1412.6980.