Adversarial Attack on DISQUS Toxic Comment Filter

Harshal Gajjar
hgajjar3@gatech.edu
Georgia Institute of Technology
Atlanta, Georgia, USA

Pranjal Gupta
pgupta332@gatech.edu
Georgia Institute of Technology
Atlanta, Georgia, USA

Thomas Macheras
tmacheras3@gatech.edu
Georgia Institute of Technology
Atlanta, Georgia, USA

ABSTRACT

DISQUS is a commonly used comment system for websites. It is a WordPress plug-in, and its ease of use has allowed it to be incorporated into hundreds of thousands of websites. It also has additional features that make it desireable for website moderators to use. One of these features is a toxic comment filter, which flags comments that DISQUS views as toxic, which the website moderator can then remove. DISQUS receives over 50 million comments a month, and currently there is no literature regarding the susceptibility of DISQUS's toxic comment filter to adversarial attack. Thus, we introduce an adversarial attack aimed at analyzing the robustness of this toxic comment filter. Our model consists of a conditional language model (encoder and generator) to generate realistic comments, a Region-Separative Generative Adversarial Network - RSGAN - (discriminator and generator) to generate realistic comments, and a toxic comment classifier to use as a surrogate model which enables the generator to craft adversarial examples. Our attack had a success rate of 4%, which was less than other baselines (HotFlip [1] and TextBugger [7]). Its BLEU Score, which measures quality of text, was 18.8, which was also lower than the baselines. We explore the reasons for this throughout the paper and in the Conclusion.

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

KEYWORDS

datasets, neural networks, gaze detection, text tagging

ACM Reference Format:
Harshal Gajjar, Pranjal Gupta, and Thomas Macheras. 2018. Adversarial Attack on DISQUS Toxic Comment Filter. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 5 pages. https://doi.org/XXXXXXX.XXXXXXX

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

DISQUS's toxic comment filter is a black box model. The only information that is provided pertaining to how it works is that the filter utilizes natural language processing and machine learning to analyze and identify comments likely to be toxic. Thus we must create a black box attack on DISQUS's toxic comment filter in order to be able to analyze its robustness. Our attack, in order to be successful, would need to be able to create adversarial comments which are able to successfully fool DISQUS's toxic filter by providing comments which are toxic, but which DISQUS does not view as toxic. Existing text based attack methods do not meet the criteria for this problem. This can be for a variety of reasons, such as only doing character or word-level modification, as opposed to generating text, or just generating adversarial text for a different purpose. Both reasons do not meet the requirements for our project.

The dataset used in our project was collected from Kaggle. It was a dataset of comments posted to online forums, and it feature a wide variety of toxicity. Using one Selenium script, we began posting these comments to DISQUS, and using another Selenium script we retrieved the labels from the admin page associated with the website where the comments were being posted. This left us with over 13 thousand comments labeled by DISQUS, with 732 of the comments being labeled as toxic by the filter. This step was necessary as there were 1296 comments we posted to DISQUS which were labeled as toxic in the Kaggle dataset, but there were only 732 comments which were actually view as toxic by DISQUS. For baselines, we used two text-based adversarial attacks, HotFlip [1] and TextBugger [7], which modify a text input to fool a classifier. Our method utilized a conditional language model, an RSGAN, and a toxic comment classifier used as a surrogate model. Our model allows for phrase-level modifications, trains the encoder, and uses a very powerful surrogate model.

Our model underperformed compared to the baselines, showing that our method was not very successful in fooling DISQUS's toxic comment filter. Our model had an attack success rate of 4%, much lower than that of HotFlip and TextBugger (42.4% and 49.8% respectively). Our model also had a BLEU Score, which measures quality of text, of 18.8, which was also lower than the BLEU Scores of HotFlip and TextBugger (31.4 and 61.9 respectively). The goal/impact of our project was to evaluate the robustness of DISQUS's toxic comment filter to an adversarial attack which attempted to generate toxic comments which were able to fool DISQUS's model. We found that, at least with our approach to this black box attack, that DISQUS was indeed robust to the method we produced. We also found that DISQUS was not as robust to adversarial attacks whose

approach used perturbations at the text or character level, such as our baselines.

2 LITERATURE SURVEY

Adversarial generation of text has been performed in the past under both white-box and black-box settings. Under the white-box setting, the attacker has access to both the model architecture and its parameters. However, under the more challenging black-box settings, the attacker does not have access to these properties of the model. To craft adversarial examples, they must rely on the labels produced by these black-box models.

[8] performed adversarial attacks in a Hard Label Black-Box setting, i.e. no access to model properties, and only a hard label from the black-box to work with. They leverage population based optimization algorithms in order to craft plausible adversarial examples. In each step of the algorithm, they allowed word replacements that maximized overall semantic similarity with the original sentence. The final goal was to craft adversarial examples with maximum semantic similarity with the original sentence. However, since they only explore word-level synonym changes, they are optimizing over a very small search space. In addition, they don't explore character level or phrase level modifications. Finally, their approach utilizes genetic algorithms, which are not very efficient.

[6] develop MALCOM, an end-to-end adversarial comment generation framework that attacks fake news detection models. It uses previously posted comments and content of the news article to generate these adversarial comments. These comments when appended to the news article fools several state-of-the-art fake news detection models. To achieve this, it trains a generator with STYLE and ATTACK modules. The STYLE module improves the writing style and relevancy of the adversarial comment to the article content, while the ATTACK module ensure to fool the target classifier. However, in this work, an article as well as previous comments are required to generate adversarial comments. We need to perform adversarial comment generation from just one comment. In addition, they use a pretrained encoder called the 'Universal sentence encoder'. This inhibits the generator since the pre-trained encoder might be producing representations that aren't ideal for the generator to learn to decode. Finally, the surrogate model that they train to replace the black-box model under the black-box attack is not very powerful and achieves only a 0.78 F1 score.

3 DATASET DESCRIPTION AND ANALYSIS

We started with *Toxic Comment Classification dataset* from Kaggle. This dataset contains over 100,000 comments which we had to clean by removing line breaks, and some very special characters. Later, we randomly took 13,680 of those comments and posted it on our DISQUS instance (with toxic filtering enabled) using a selenium script. The posted comments were then collected back from DISQUS moderation page with toxic labels using another selenium script and the final dataset we had contained 3 columns of interest (amongst others from Kaggle and DISQUS) - comment (from Kaggle), toxic (true label from Kaggle), and DISQUS toxic label (from DISQUS).

The final labels that we obtained are summarized in table 1. Figure 1 shows the number of unique N-grams in the collected comments for N ranging from 1 to 7.

The dataset we constructed was sufficient for our goal of generating DISQUS mislabelled toxic generation model, since it contained comments with all combinations of labels from DISQUS and Kaggle.

	Toxic	Non Toxic	Total
DISQUS Toxic	840	22	862
DISQUS Non Toxic	796	12022	12818
Total	1636	12044	13680

Table 1: Labels Overview

Parameter	Value
Max(words in a comment)	1403
Mean(words in a comment)	55.09

Table 2: Comments Overview

	Toxic	Non Toxic
Positive Sentiment	546	1403
Neutral Sentiment	294	5152
Negative Sentiment	796	4508

Table 3: Sentiments Overview

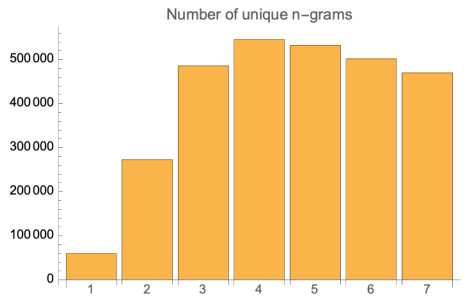

Figure 1: Unique N-grams in our collected dataset

We define *repeat ratio* as ratio of total number of words in the comment to the number of unique words in the comment. Upon plotting the repeat ratios for each of the comment in our dataset (Figure 2) we observed that comments with high repeat ratios tend to be toxic.

4 EXPERIMENT SETTING AND BASELINES

The task of our project was to create a model which can generate novel toxic comments which DISQUS's toxic comment filter would misclassify as being non-toxic. The evaluations metrics we used on our attack were its Attack Success Rate (the percent of comments generated by our model that are misclassified by DISQUS), BLEU score of the comments generated (the quality of the text in

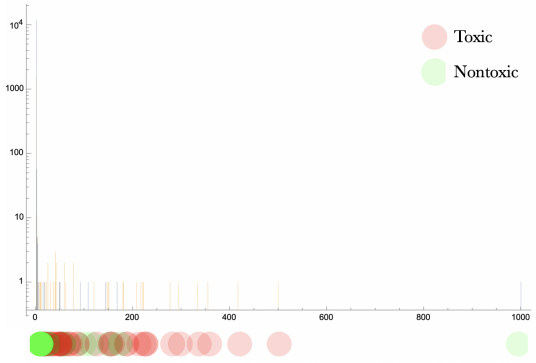

Figure 2: Histogram of repeat ratio vs count

the generated comments), and Perplexity score of the comments generated. We utilized a train set which was used for training the autoencoder, a validation set which was use to evaluate our metrics and tune hyperparameters, and a test set for final reporting of the scores. Throughout our project we utilized PACE and CoLab, both of which have GPUs which were required for training our model.

The first baseline we implemented was HotFlip [1]. HotFlip creates its adversarial examples with character substitutions. These substitutions are the "flips". It obtains the one-hot representation for a text input, and uses the gradient of this to estimate what single change would yield the highest estimated loss. This can also be adapted to word-level, but when doing so it is also necessary to preserve the semantics and the sentiment of the sentence (not an issue when flipping characters). In order to do this, the following constraints for flipping one word to another are put in place: the cosine similarity between the embeddings of the words much be greater than a certain threshold, the words must be the same part of speech, and neither of the words can be stopwords or part of the same lexemme.

The second baseline we implemented was TextBugger [7]. TextBugger, like HotFlip, can be used for both character and word level perturbation. The input provided to TextBugger in our case is a toxic comment. First, it will find the important toxic sentences within that toxic comment. Once it has these sentences, it will find the important words within those sentences, and then do character or word level perturbation on those words.

The results of these baselines can be seen in Table 1 and Table 2 below. Table 1 shows each method's Attack Success Rate on both DISQUS and our black box surrogate model. Table 2 shows the overall performance of both DISQUS and the black box model against both attacks. It can be seen that TextBugger was more successful at attack both DISQUS and the black box surrogate model than HotFlip was. Also, while TextBugger was more effective against the black box surrogate model than it was against DISQUS, HotFlip was more successful against DISQUS that it was against the black box surrogate model. This could be due to the black box surrogate model being susceptible to character-level perturbations. Overall, DISQUS performed better against TextBugger's attack than the black box surrogate model did, and the black box surrogate model performed better against HotFlip's attack than DISQUS did.

	TextBugger	HotFlip
Attack Success Rate on DISQUS	.498	.424
Attack Success Rate on Black Box	.648	.261

Table 4: Attacker Performance

	TextBugger	HotFlip
DISQUS Recall	.502	.594
DISQUS F1 Score	.668	.745
Black Box Recall	.352	.739
Black Box F1 Score	.521	.850

Table 5: Classifier Performance

5 PROPOSED METHOD

We propose to attack the DISQUS toxic filter by employing three modules that accomplish different objectives that are necessary for generating adversarial comments that are also plausible. Conditional language module: This is used to capture the underlying grammatical structure present in the comments. It consists of an encoder and generator that are co-trained. Realism module: We use Relativistic Generative Adversarial Networks (RSGAN) as described in [3]. This module guides the generator to produce realistic comments that fool the discriminator. Similar to a traditional Generative Adversarial Network (GAN), they are trained in an adversarial fashion. Adversarial module: This consists of a surrogate classifier that is used in place of the target black-box model. This module trains the generator to craft adversarial examples that fool the surrogate classifier.

5.1 Conditional language module

Let E and G represent the encoder and generator respectively. The encoder learns a vector representation of the original comment c. This representation is denoted by $E(c)$. This is obtained using a sequential text encoder. The generator is a conditional sequential text generation model that generates malicious comments c^* by sampling one token at a time, conditioned on (i) previously generated words, (ii) the original comment representation, $E(c)$ and (iii) a random latent variable z. Each token is sequentially sampled according to the conditional probability function:

$$p(c^*|c;\theta_G,\theta_E) = \Pi_{t=1}^{T} p(c_t^*|c_{t-1}^*, c_{t-2}^*, ..., c_1^*; E(c); z) \quad (1)$$

Where c_t^* is a token sampled at time step t, T is the maximum generated sequence length. θ_G are the parameters of the generator and θ_E are the parameters of the encoder that are to be learned. We can think of this as a language model, with additional conditioning on the vector representation and latent variable. The encoder and generator are trained together using MLE with teacher forcing as in [5] by maximizing the negative log-likelihood (NLL) of the generated comment. Mathematically, we want to optimize

$$\min_{\theta_G,\theta_E} L^{MLE} = -\sum_{i=1}^{N} c_i \log p(c^*|c_i;\theta_G,\theta_E) \quad (2)$$

5.2 Realism module

After training the generator and encoder with the Conditional language module, the generator is able to produce comments that are grammatically correct, but not necessarily realistic. To ensure that the generated comments are realistic as well, we use the realism module. In RSGAN, the generator G aims to generate realistic comments to fool the discriminator D, while the discriminator aims to discriminate whether a randomly sampled comment c is more realistic than a comment generated by the generator using c as described earlier. Specifically, we alternately optimize D and G with the following two objective functions:

$$\min_{\theta_G} L_G^D = -\mathbb{E}_{c\sim p_D(\chi);z\sim p_z}[\log(\sigma(D(c) - D(G(E(c),z))))]$$

$$\min_{\theta_D} L_D = -\mathbb{E}_{c\sim p_D(\chi);z\sim p_z}[\log(\sigma(D(G(E(c),z)) - D(c)))] \quad (3)$$

where σ is the sigmoid function, θ_D are parameters of the discriminator, χ is the dataset of comments, p_z is a noise prior that z is sampled from. By using D, we want to train the generator G to produce comments that are more realistic and have the same style as comments in the dataset.

5.3 Adversarial module

After training the generator with the Realism module, the generator is able to construct realistic and grammatically correct comments. The classifier in this module is a basically a surrogate for the black-box toxic comment filter. To generate adversarial comments, the generator must fool the surrogate classifier f into producing the opposite label for a comment from the dataset. In order to be a good surrogate for the black-box model, the classifier must predict whether a given comment from the dataset is toxic or non-toxic. f can be trained using binary-cross-entropy loss as follows:

$$\min_{\theta_f} L_f = -\frac{1}{N}\sum_{(c_i,y_i)\sim\chi}(y_i\log(f(c_i)) + (1-y_i)\log(1-f(c_i))) \quad (4)$$

where θ_f denotes the parameters of the classifier, y_i are the labels of the comments c_i from dataset χ and $N = \chi$. We then use this trained model f to guide G, i.e. we use signals backpropogated from f to force G to generate a new comment c^* such that $f(c^*)$ outputs a target prediction label $L^* \in \{0, 1\}$. Specifically, we want to optimize the objective function:

$$\min_{\theta_G} L_G^{f(L^*)} = -\frac{1}{N}\sum_{(c_i,y_i)\sim\chi}(L_i^*\log(f(c_i)) + (1-L_i^*)\log(1-f(c_i))) \quad (5)$$

One attack scenario is for an attacker to promote a toxic-comment as non-toxic. We can do this by L_i^* = 'non-toxic' for the comments where y_i = = 'toxic'. An attacker could also demote a non-toxic comment as toxic by L_i^* = 'toxic' for the comments with y_i = 'non-toxic'. The overall algorithm is shown in Figure 3.

Algorithm

1. Pre train generator and encoder with equation (2)
2. Pre-train classifier using equation (4)
3. **Repeat:**
4. Train generator and discriminator adversarially using equations in (3)
5. Train generator by using classifier as in equation (5)
6. **Until** convergence

Figure 3: Generating Adversarial Comments Algorithm

5.4 Implementation details

Training with Discrete Data: We need to back-propagate the gradients of the loss in Eq. (3) and (5) through discrete tokens sampled by G from the multinomial distribution c^t at each time-step t. Since this sampling process is not differentiable, we employ the relaxation trick as in [2] with a generation temperature (τ) to overcome this. The τ parameter is annealed during training, to yield increasingly peaked distributions.

Generation strategy: For each comment c, we generate a new comment c^* from $G(E(c), z)$, where $z \sim N(0, I)$. We can generate multiple adversarial comments by just sampling multiple z from p_z.

Architecture: The encoder consists of a word embedding layer. We use pre-trained GloVe embeddings ([10]). For Out Of Vocabulary (OOV) words, we initialize their word embeddings randomly. During the encoder training, the embedding layer is also trainable, to learn the embeddings for these OOV words. We use a two layer Bidirectional Recurrent Neural Network (BRNN) with dropout between the layers. We use Long Short Term Memory units (LSTM) as the hiddent units of the RNN network. The final hidden state from the last layer is passed through a fully connected layer (FC), which gives us the final vector representation of the comment.

The generator is a single layer LSTM, where the output of each hidden unit is connected to a FC layer of the same size as the vocabulary. This FC layer is shared across all hidden units. We then use softmax activation function for the FC layer to obtain a probability distribution over the vocabulary. At each step, we try to predict the next word by using previously generated words, the vector representation of the comment obtained from the encoder and the latent noise vector. Thus, at each step we concatenate the word embedding of the previously generated word with the vector representation of comment and noise vector. To prevent overfitting, we use word dropout, as described in [4].

For the discriminator we use multi discriminative representations as in [9]. This consists of parallel convolutional and max-pooling channels, where we apply max-pooling over time independently for each channel. Then, we concatenate the output from each channel and feed it to a FC layer of size 2, to obtain a score for the comment being real or fake.

The classifier has the same architecture as the discriminator, except that it predicts a score for the comment being toxic or non-toxic.

Parameter Setttings: We limit the comments to 60 words. We use a batch size of 32, with the hidden unit size of 300 for all LSTMs. The noise vector z is chosen to be of size 100. We use the Adam optimizer with a learning rate of 1×10^{-3} with a learning rate decay. We use LSTM between layer encoder dropout as well as

word dropout with probability of being dropped as 0.3. We use 100 dimensional word embeddings. We use 100 epochs for step 1 of the algorithm (please see Figure 3), 15 epochs for step 2 and 50 epochs for the steps 4 and 5.

Shortcoming addressed: We address several shortcomings from previous works. First, our generator accomodates for word level or even phrase level modifications to original comments (unlike [8]). Second, we co-train the encoder and generator, which allows the encoder to produce representations that make it easier for the generator to decode. They have a more mutually beneficial relationship (compared to [6] which uses a pre-trained encoder). Finally, the surrogate classifier we train is much more powerful than the one in [6], which achieves only a 0.78 F1 score.

6 EXPERIMENTS AND RESULTS

The results of our proposed method compared to the baselines can be seen in Table 3 and Table 4 below. Table 1 shows each method's, both ours and the baselines', Attack Success Rate on DISQUS and its BLEU Score. Table 2 shows the overall performance of DISQUS against our attack and both baseline attacks. It can be seen that our model underperformed compared to both baselines. The Attack Success Rate was much lower than HotFlip and TextBugger. Also, the BLEU Score of our model, a metric to measure the quality of text, was also lower than those of HotFlip and TextBugger. DISQUS also performed best against our model, with better Recall and F1 Score than it had against the two baselines.

	TextBugger	HotFlip	Our Model
Attack Success Rate on DISQUS	.498	.424	.040
BLEU Score	61.9	31.4	18.8

Table 6: Attacker Performance

	TextBugger	HotFlip	Our Model
DISQUS Recall	.502	.594	.706
DISQUS F1 Score	.668	.745	.750

Table 7: DISQUS Performance Against Attacks

7 CONCLUSION

There are some shortcomings and limitations of our work. The first is that our dataset is not very clean. It contains many URLs, IP addresses, etc. Furthermore, while we were able to generate comments, a majority of them were not realistic and/or grammatically incorrect. Our generator was also unable to fool the black box classifier when creating adversarial examples.

There are also many possibilities for extensions and future work. The first that could solve some problems from our project would be the use of a different, cleaner dataset. Another possibility would be to co-train the classifier with the generator instead of pretraining it. A Relational Memory Recurrent Network (LMRN) could also be utilized instead of LSTM as the generator. This might help to generate more plausible comments, as LMRNs are more powerful and use an attention mechanism. Standard attention could also be at the encoder level to better capture vector representations of longer comments.

Contributions: All team members have contributed a similar amount of effort.

REFERENCES

[1] Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. 2018. HotFlip: White-Box Adversarial Examples for Text Classification. *arXiv preprint arXiv:1712.06751* (2018).

[2] Eric Jang, Shixiang Gu, and Ben Poole. 2016. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144* (2016).

[3] Alexia Jolicoeur-Martineau. 2018. The relativistic discriminator: a key element missing from standard GAN. *arXiv preprint arXiv:1807.00734* (2018).

[4] Mikael Kågebäck and Hans Salomonsson. 2016. Word sense disambiguation using a bidirectional lstm. *arXiv preprint arXiv:1606.03568* (2016).

[5] Alex M Lamb, Anirudh Goyal ALIAS PARTH GOYAL, Ying Zhang, Saizheng Zhang, Aaron C Courville, and Yoshua Bengio. 2016. Professor forcing: A new algorithm for training recurrent networks. *Advances in neural information processing systems* 29 (2016).

[6] Thai Le, Suhang Wang, and Dongwon Lee. 2020. Malcom: Generating malicious comments to attack neural fake news detection models. In *2020 IEEE International Conference on Data Mining (ICDM)*. IEEE, 282–291.

[7] Jinfeng Li, Shouling Ji, Tianyu Du, Bo Li, and Ting Wang. 2018. TEXTBUGGER: Generating Adversarial Text Against Real-world Applications. *arXiv preprint arXiv:1812.05271* (2018).

[8] Rishabh Maheshwary, Saket Maheshwary, and Vikram Pudi. 2021. Generating natural language attacks in a hard label black box setting. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence*.

[9] Weili Nie, Nina Narodytska, and Ankit Patel. 2018. Relgan: Relational generative adversarial networks for text generation. In *International conference on learning representations*.

[10] Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 1532–1543.