Adversarial Attack on Sentiment Classification

Alicia Yi-Ting Tsai
aliciatsai@berkeley.edu

Erica Chen
hanyu_chen@berkeley.edu

Tobey Yang
minchu.yang@berkeley.edu

1 Introduction

In the past few decades, machine learning and deep learning techniques have been successful in several applications. However, these techniques developed so far are proven to be vulnerable given some manipulated inputs, which are called adversarial examples, that human can easily distinguish but algorithms can not.

Current research have shown successful results in producing adversarial images that cause the algorithms to completely fail in computer vision. However, studies of adversarial examples in the applications of natural language processing such as sentiment analysis, fake news detection and machine translation remain relatively low. Nonetheless, it is an emerging field that is worth exploring.

In this project, we plan on investigating various algorithms to generate adversarial inputs that fool currently state-of-the-art sentiment classifiers and study why a classifier fails given these adversarial inputs. This project consists of the following parts. We review recent studies on adversarial learning and adversarial examples in natural language. Next, we will generate adversarial input by several algorithms and evaluate these malicious inputs on sentiment classifiers such as multinomial naive bayes (MNB) (Wang and Manning, 2012), and convolutional neural network (CNN) (Kim, 2014). We plan to further analyze the sentence error and imperceptibility caused by the adversarial examples. Finally, we hope to explore new methods that can generate subtler and more powerful adversarial examples.

2 Related Work

Adversarial learning in natural language processing has increasing attention recently due to the success of adversarial learning in the field of computer vision. When generating an adversarial example, if the adversary does not have knowledge of the classifier or the training data, we call this a black-box setting. On the other hand, if the adversary has full knowledge of the classifier and the training data, we call this a white-box setting.

In a black-box setting, Belinkov and Bisk (Belinkov and Bisk, 2017) introduces a simple attack method by randomly replacing characters with their nearby key on the keyboard, which is similar to keyboard typos, to attack a machine translation system. Similar idea can be found in the work of Hosseini et al. (Hosseini et al., 2017), the authors generate adversaries that deceive Google Perspective API by misspelling the abusive words or adding punctuation to the letters. Another work from Alzantot et al. (Alzantot et al., 2018) attempts to generate semantically and syntactically similar adversarial examples by word replacement. They develop an genetic algorithm that uses population-based gradient-free optimization, which is inspired by the process of natural selection. In the black-box setting, the adversary tries different perturbations and evaluates the quality of perturbations by querying the model to get the classification result or the output score. The adversary continues to altered the sentence until the model fails or until score reduces significantly.

In the white-box setting, the adversary has access to the model and thus is capable of generating more sophisticated adversarial examples. Ebrahimi et al. (Ebrahimi et al., 2017) show that adversarial examples generated in a white-box setting reduce the success rate of a text classification model much more than examples generated in a black-box setting. The authors introduce a white-box adversary against differentiable classifiers that substitutes characters ("flips") in a sentence. When operating in a white-box setting, the adversary has full access to the gradients of the classifier, giving the adversary important informa-

tion to find the classifier's weak points. Because white-box adversary has access to the gradients of the model, the adversary does not have to query the output score from the classifier every time. Using the gradients as a surrogate loss, the white-box adversary can efficiently find the best changes that maximize the surrogate loss simply by backpropagation.

Other white-box adversary include word-level substitution. Kuleshov et al. (Kuleshov et al., 2018) construct adversarial examples by replacing 10-30% words in a sentence that does not change the meaning of the original sentence. The authors construct the adversarial examples by solving an optimization problem that maximizes a surrogate loss subject to semantic and syntactic similarity constraints using a greedy approach. Similar idea can be found in Samanta and Mehta's work (Samanta and Mehta, 2017) where they apply three different rules (insertion, replacement and deletion of words) to generate adversarial examples while persevering semantic meaning and the grammar of the text sample. Another work conducted by Liang et al. (Liang et al., 2018) combines the three strategies above to generate adversarial examples and avoid introducing excessive modification or insertion to the original text.

White-box adversaries that alter words or characters usually face the challenge of preserving semantic meaning of the sentence. Instead of substituting words or characters, Jia and Liang (Jia and Liang, 2017) propose a concatenative adversary that appends irrelevant or distracting sentences at the end of the original paragraph for the task of question answering. A distracting sentence is generated by combining a mutated question that is similar to the original question and a generated fake answer. This sophisticated crafted sentence is highly capably of confusing models that merely detect keywords.

3 Data

We use a dataset of 25,000 informal movie reviews from the Internet Movie Database (IMDB) (Maas et al., 2011). We randomly select 80% of the dataset to include in the train split, and 20% in the test split. In other words, we use 20,000 reviews for training, and 5,000 reviews for validation. Also, we use the provided tokenizations when they exist.

4 Method

4.1 Baseline Models

The baseline models we choose to experiment on include Naive Bayes, a commonly used linear model, and convolutional neural network model. We described our implementation of the baseline models in the following subsections. The results can be found in Table 1.

4.1.1 Naive Bayes

We implemented Multinomail Naive Bayes (MNB) (Wang and Manning, 2012) for the sentiment analysis task with two classes, positive and negative. Our implementation includes one Naive Bayes with unigram and one with bigrams feature. We convert a collection of text documents to a matrix of token counts from training data, and selected 30,000 features which ordered by term frequency across the corpus. We also used add-one (Laplace) smoothing to avoid zero probability in the likelihood term. In both unigram and bigrams models, we trained with 20,000 reviews and tested on another 5,000 reviews. Our unigram Naive Bayes has 0.85 testing accuracy and our bigrams Naive Bayes has 0.88 testing accuracy.

4.1.2 Convolutional Neural Network

In this experiment, we used convolutional neural network classifier as our attacking target (Kim, 2014). We replicated the architecture of the CNN model from Kim's previous work (Kim, 2014). In the CNN model, an embedding of a fixed dictionary and size serves as the very first layer; a convolutional layer with filter widths of 3, 4, 5 and 100 feature maps each is added; a max-over-time pooling layer and a fully connected layer with dropout rate of 0.5 were followed. We trained the model with 20,000 reviews and tested with another 5,000 reviews for 20 epochs with batch size of 64, reaching testing accuracy of 0.9.

4.2 Black Box: Swap Characters

In the black box attack, we insert artificial noise (typos, misspellings, etc.) into the corpus to generate adversarial examples. The source of noise here is swapping two characters in a word (eg. pe**rf**ect $\rightarrow$ pe**fr**ect), which is similar to keyboard typos when typing quickly. We apply one swap per word, but do not alter the first and the last characters. Therefore, the noise is only applied to word of length larger than 3.

Accuracy	Training	Validation
Naive Bayes (Bigram)	0.91	0.88
CNN	0.99	0.90

Table 1: Baseline Classifier Result

In addition, we do not want to change too many words in a sentence for the semantic reason. Thus we experiment on 500 samples to test the success rate under different swap rate (from 0.05 to 0.5) in unigram and bigrams Naive Bayes model. Figure 1 shows how the success flip rate increase when the swap proportion of words increase. There exists a trade-off between similarity and success rate, considering both of them, we alter 50% of the words when generating adversarial examples.

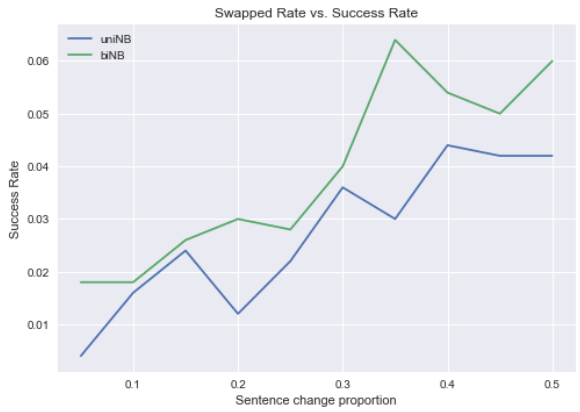

Figure 1: Success Rate vs Swapped Rate

After experimenting the black box attack in the bigram Naive Bayes model, the accuracy decrease from 0.88 to 0.84, which is quite small.

4.3 White Box: Greedy Search

In the greedy algorithm, we attempt to generate adversarial examples through an iterative approach. The proposed method is that, at each time step, we find a set of word candidates for one word in the original sentence, then pick the word that has the greatest impact on the prediction result to replace with. Although the objective of the algorithm is to fool the sentiment classifier, the grammar, semantic and sentiment should not be altered. Thus, some constraints are imposed in order to maintain the similarity of the original and generated sentences.

Word Choice. When picking the word candidates in each time step, we consider the top k

Algorithm 1 Generate adversarial example via greedy search

$\sigma \leftarrow$ *model.logit*(x)
Initialize $x' \leftarrow x$, *count* $\leftarrow 0$
$X \leftarrow \emptyset$
for w in x' **do**
  $\sigma' \leftarrow$ *model.logit*(x')
  candidates $\leftarrow$ k nearest neighbors of w within distance d and have the same POS tag as w
  for w' in *candidates* **do**
    $\bar{x} \leftarrow$ replace w with w'
    $X \leftarrow X$ *append* $\{\bar{x}\}$
  if $X = \emptyset$ **then** *continue*
  if $\sigma > 0$ and $\min($*model.logit(X)*$) < \sigma'$ **then**
    $x' \leftarrow \text{argmin}_{\bar{x}}($*model.logit(X)*$)$
    count $\leftarrow$ *count* $+1$
  else if $\sigma < 0$ and $\max($*model.logit(X)*$) > \sigma'$ **then**
    $x' \leftarrow \text{argmax}_{\bar{x}}($*model.logit(X)*$)$
    count $\leftarrow$ *count* $+1$
  $\sigma' \leftarrow$ *model.logit*(x')
  if $\sigma * \sigma' < 0$ **then return** x'
  if *count*$/\text{len}(x) > r$ **then return** *None*

nearest neighbors in the word embedding space. The nearest neighbors in GloVE word embedding space usually appear in the same context as the original word. Thus, by picking nearest neighbors in GloVE vectors, we can ensure semantic similarity after word replacement. Although word embedding helps to find words that are used in similar context, it does not guarantee that the part of speech will remain the same after replacement. Therefore, we examine the part of speech of the original word and make sure the selected candidates have the same part of speech as the original word.

Hyper-parameters. In this model, there are three hyper-parameters, k, d, and r, used to maintain the semantic similarity. The first parameter, k, is used to decide how many nearest neighbors should be considered in the first place. When k is too small, there might not be enough candidates to successfully form an example. d represents the maximum distance allowed between the candidate and the original word. When d is large, the meaning of the altered example is generally farther from the original one. When d gets too small, the chance of generating a successful attack decreases. Another hyper-parameter, r, stands for the percentage of replacement allowed in the sentence. When the threshold is too low, the chance of generating a successful adversarial example decreases. However, when r is too large, too many words will be replaced and thus the generated input does not remain similar to the original input.

Runtime complexity. Let the length of each input be n. The runtime complexity of the greedy algorithm is then $O(k \cdot n)$. For each time step, there are k candidates to be considered. Thus, we need to compute the sigmoid value (logit) of each candidate, the complexity of each time step is $O(k)$. We need to repeat the above operation until the threshold is reached or the label is successfully flipped. The complexity is linear to the length of the input, $O(n)$. As a result, the overall complexity is $O(k \cdot n)$.

4.4 White Box: Global Search

The greedy approach proposed above does not guarantee to produce the optimal results and is sometimes time consuming because the algorithm needs to search the candidates words for every time step. To mitigate the problem, we propose to search for the candidates globally by computing a small perturbation, δ, using a custom loss function.

To learn the perturbation, δ, we define a custom loss function $L(\delta)$. The loss function tries to maximize the difference between the logits value of the original input x and that of perturbed input $x + \delta$. Furthermore, we add two regularization terms in the loss function. The first regularization penalizes large perturbations and is controlled via the hyperparameters λ_1. The second regularization penalizes large distance between original embeddings and perturbed embeddings and is controlled by the hyperparameters λ_2. The two regularization terms are added to help keep the semantic of chosen words.

$$\begin{aligned}\mathcal{L}(\delta) = &- \left(\text{logit}(E_x) - \text{logit}(E_x + \delta)\right)^2 \\ &+ \lambda_1 \cdot \|\delta\|_2 \\ &+ \lambda_2 \cdot \|(E_x - (E_x + \delta))\|_2\end{aligned}$$

The attacking algorithm is described in algorithm 2. We first initialize the perturbation δ to be 0. Here, the original input embeddings are denoated as E_x and the perturbed embeddings are denoted as E'_x. For each time step, the algorithm computes the gradient of the custome loss function with respect to the perturbation, Δ_δ, and update the perturbation δ via backpropagation. We then find the new word, w, in the original embeddings space that is closet to the perturbed embeddings and record the word. Finally, the algorithm stops when the newly selected words flip the predicted label and return the recored words and the perturbation.

Next, we can use the generated new words and the perturbation to generate the adversarial example. The algorithm is described in algorithm 3. The algorithm sorts the magnitude of the perturbation and replaces words with the highest magnitude first, which search for the optimal words to be replaced in the entire input example. Here, we have a hyperparameter d that controls the threshold for word distance. If the last candidate word is too far away from the original word, then we reject the candidate and search for previously recorded candidate word. We also have another hyperparameter r that controls the percentage of changes allowed.

Algorithm 2 Global Search Attack Function

$y \leftarrow$ *model.predict(*x*)*
Initialize $\delta \leftarrow 0$, *success = False*
$X \leftarrow X \cup \{x\}$
$E_x \leftarrow$ input embeddings
while not *success* **do**
    $\Delta_\delta \leftarrow$ via *back-propagation*
    $\delta \leftarrow \delta - \epsilon \cdot \Delta_\delta$
    $E'_x \leftarrow E_x + \delta$ ▷ perturbed embeddings
    $W \leftarrow \emptyset$ ▷ new words
    for e in E'_x **do**
        $D \leftarrow$ *pair-wise distance(*e, E_{vocab}*)*
        $W \leftarrow W$ *append* $\text{argmin}_w(D)$
    $\hat{y} \leftarrow$ *model.predict(*W*)*
    $X \leftarrow X \cup \{W\}$
    if $\hat{y} \neq y$ **then**
        return X, δ

Algorithm 3 Global Search Generate Adversary Function

$y \leftarrow$ *model.predict(*x*)*
positions $\leftarrow$ *reversed*(argsort $\|\delta\|_2$)
Initialize *success = False*
for i in *positions* **do**
    for $\bar{x}$ in *reversed*(X) **do**
        if $\|E_{x_i} - E_{\bar{x_i}}\|_2 < d$ **then**
            $E_{x_i} \leftarrow E_{\bar{x_i}}$ ▷ replace embeddings
            break
    $x_i \leftarrow \bar{x_i}$ ▷ replace word
    $\hat{y} \leftarrow$ *model.predict(*x*)*
    if $\hat{y} \neq y$ **then**
        success = True
        return *success*, x
    if $\frac{i}{\text{len}(x)} > r$ **then**
        return *success, None*

5 Analysis

5.1 Success Rate

By far, the end goal of the algorithms proposed is to flip the label. In assessing the effectiveness of the attacking models, we select 500 examples that are correctly classified from the test set, so that the accuracy of the classifiers does not affect our evaluation. We then input samples into the attacking models to generate adversarial examples. We apply black-box character swapping method to Naive Bayes, and both greedy and global search method to CNN. After experimenting the algorithm and tuning the hyper-parameters, the success rate of the greedy approach is around 0.65. Comparing with the global search approach, the greedy approach requires a larger proportion of change to successfully generate an example. It is due to the fact that greedy approach might alter words that do not contribute a lot in flipping the label. The word is replaced as long as it makes the prediction leans toward the opposite label. Another limitation of the greedy approach is that words are being replaced from the beginning of the sentence, the words modified are located in the same area of the sentence, reducing the readability as well as the similarity. The results can be found in Table 2.

5.2 Hyper-parameters

As mentioned above, there are 3 hyper-parameters in the greedy approach, which are k, the number of candidates to be considered; d, the maximum distance allowed between the original word and the candidate; r, the ratio of word changes allowed with respect to the length of the sentence. The global search algorithm also includes the hyper-parameters d and r. In our experiment of the greedy approach, k is set to 20. This parameter does not affect much since the maximum word distance allowed and the limitation of picking words with the same part-of-speech tag help to maintain similarity. The parameter is used to reduce the run-time complexity of the algorithm by limiting the number of candidates. As for d, we run the algorithm for a few different values, we decide to set d to 20 to reach a high success rate while not picking words too far from the original one. There exists a trade-off between similarity and success rate in choosing the value for d. While allowing words farther from the original word, the success rate increases but similarity decreases as a penalty. The threshold r controls how

Swap Characters	Greedy Search	Global Search
0.06	0.65	0.72

Table 2: Success rate of attacking methods

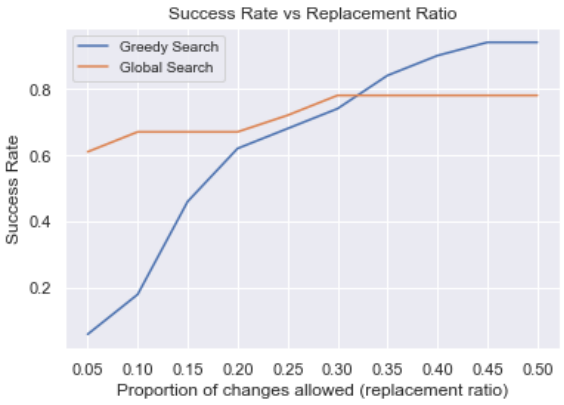

Figure 2: Success Rate vs Replacement Ratio

Figure 3: Success Rate vs Word Distance

many words are allowed to change in a sentence; it is also shared by the two approaches. In greedy approach, r should be at least set to 0.2 to achieve a good result. When r is set to 0.1, the success rate is merely 0.18; when r is 0.2, the success rate increases to 0.65. So we decide to set the threshold r to 0.2 for greedy approach in the following experiment. On the other hand, the global search algorithm can reach a pretty good result when r is set to 0.1. In our experiment, we found that global search algorithm is more effective than the greedy approach since the global search algorithm looks for the word that contributes most to classification result while greedy algorithm swap words with any degree of contribution.

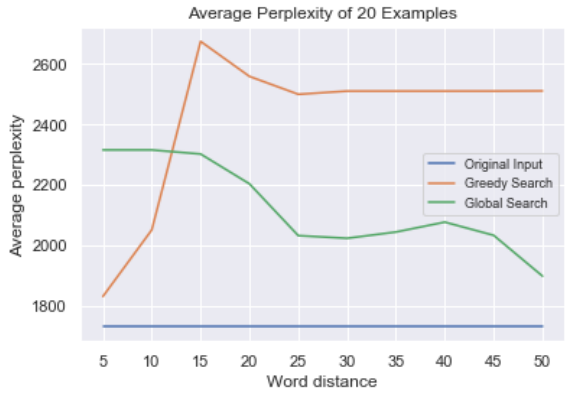

Figure 4: Average Perplexity vs Word Distance

5.3 Perplexity

In order to evaluate the adversarial examples generated from our models, we decide to measure the perplexity, which is widely used in assessing language models, of 20 examples with different hyper-parameters. As Figure 4 shows, the perplexity increases as the word distance increases in the greedy approach, meaning that the generated examples are less likely to occur in the corpus. However, in the global search approach, it is the opposite. It might be that the number of change decreases while allowing larger word distance. We also calculate the perplexity with different values of proportion of changes. The perplexity increases as the proportion of changes allowed in both attacking models. See result in Figure 5. When comparing the perplexity of the two attacking models, the global search approach generally does better than the greedy one. Since the greedy algorithm usually requires more changes than the global search approach even though we have set the threshold for the replacement rate.

5.4 Human Evaluation

We conduct a survey and propose two criteria to measure the performance of the adversarial examples from three attack methods. The first criteria is the sentiment classification accuracy of the adversarial examples which is predicted by human, and the second is the similarity between the adversarial examples and the original sentence.

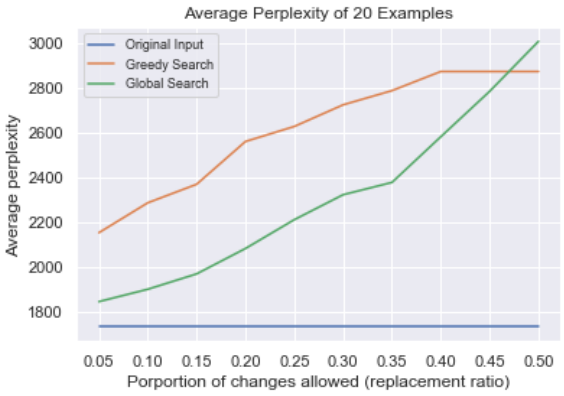

Figure 5: Average Perplexity vs Replacement Ratio

Human Prediction Accuracy. Unlike adversarial examples in the context of image classification, natural language perturbation is generally perceptible since words are deleted, added, or replaced. Thus, we need to redefine imperceptibility in the context of natural language. Since we are crafting adversarial example to fool the sentiment classifier, we define it as imperceptible if human can still correctly classify without being fooled by the perturbation. Therefore, we ask some volunteers to evaluate adversarial examples and look at the percentage of responses that match the original classification.

Sentence Similarity. In addition to classification accuracy, we also care about how similar the generated adversaries examples are to the original unaltered sentences. we ask volunteers to rate the similarity, from 1 to 5, which means less similar to very similar between the adversarial sentence and original sentence.

We choose two sentences which originally classified as positive comment and negative comment, and then flipped to opposite sentiment result after applying three attack methods. After asking 15 volunteers and analyzing the survey data, we found that the adversarial example from Global Search is most similar to the original sentence, with an average similarity of 4.4, while the example from Greedy Search is less similar, with the score of 2.9.

Refer to human prediction accuracy, Global Search method also reaches the highest accuracy, which is 0.83. It means that although sentiment classifier make the wrong prediction on the Global Search adversarial examples, human could still classify sentiment correctly. However, greedy search only has 0.30 accuracy because it alter too many words to fool the sentiment classifier. In this survey, some of the volunteers feel that the adversarial sentences from Greedy Search are hard to read and can not tell the sentiment of the senetence.

Examples of adversarial text generated
Original reviews: as long as you go into this movie knowing that it 's terrible : bad acting , bad " effects , " bad story , bad ... everything , then you 'll love it . this is one of my favorite " goof on " movies ; watch it as a comedy and have a dozen good laughs !
Global Search: as long as you go into this movie knowing that it 's terrible : worse acting , bad " effects , " bad story , bad ... everything , then you 'll love it . this is one of my favorite " goof on " movies ; watch it as a comedy and have a dozen good laughs yes
Greedy Search: as long as you leave into this blockbuster telling whether it 's horrendous : bad acting , bad " effects , " bad story , bad ... everything , then you 'll love it . this is one of my favorite " goof on " movies ; watch it as ...
Swap Characters: as lnog as you go into this mvoie knowing that it's terirble : bad atcing , bad " effects , " bad sotry , bad ... everything , then you 'll lvoe it . this is one of my favorite " goof on " movies ; watch it as a comedy and hvae a dzoen good laguhs !

6 Conclusion

In this experiment, we generalize the concept of adversarial examples to the context of sentiment classification in natural language. We prove that some machine learning algorithms are vulnerable to adversarial examples. Some works have been done using the greedy approach and have proved the method to be effective; however, the global search algorithm is proved to be much more powerful than the greedy approach. The global search method requires less change to the original sen-

	Swap Characters	Greedy Search	Global Search
Semantic Similarity (Scale: 1-5)	3.8	2.9	4.4
Human Prediction Accuracy	0.70	0.30	0.83

Table 3: Human Evaluation Survey Results

tence and maintain a higher level of similarity in terms of both human evaluation and perplexity measure.

7 Future Work

Both of the greedy and global search algorithms are operating in a white-box scenario; they are not as powerful as the black-box algorithms. The black-box character swapping algorithm could be further applied to CNN model with character-level embedding. Other word-level attacking models operating in black-box scenarios can be a way to improve the limitation of white-box models.

By far, we developed some successful strategies to attack the sentiment classifiers. Further studies can be done to strengthen the classifiers. By training the classifiers with generated adversarial examples, we hope to help defend the classifier against adversarial attack and improve the model accuracy.

References

Moustafa Alzantot, Yash Sharma, Ahmed Elgohary, Bo-Jhang Ho, Mani Srivastava, and Kai-Wei Chang. 2018. Generating natural language adversarial examples.

Yonatan Belinkov and Yonatan Bisk. 2017. Synthetic and natural noise both break neural machine translation. *CoRR*, abs/1711.02173.

Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. 2017. Hotflip: White-box adversarial examples for NLP. *CoRR*, abs/1712.06751.

Hossein Hosseini, Sreeram Kannan, Baosen Zhang, and Radha Poovendran. 2017. Deceiving google's perspective API built for detecting toxic comments. *CoRR*, abs/1702.08138.

Robin Jia and Percy Liang. 2017. Adversarial examples for evaluating reading comprehension systems. *CoRR*, abs/1707.07328.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Volodymyr Kuleshov, Shantanu Thakoor, Tingfung Lau, and Stefano Ermon. 2018. Adversarial examples for natural language classification problems.

Bin Liang, Hongcheng Li, Miaoqiang Su, Pan Bian, Xirong Li, and Wenchang Shi. 2018. Deep text classification can be fooled. *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA. Association for Computational Linguistics.

S. Samanta and S. Mehta. 2017. Towards Crafting Text Adversarial Samples. *ArXiv e-prints*.

Sida Wang and Christopher D Manning. 2012. Baselines and bigrams: Simple, good sentiment and topic classification. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers-Volume 2*, pages 90–94. Association for Computational Linguistics.