HotFlip: White-Box Adversarial Examples for NLP

Javid Ebrahimi[*], **Anyi Rao**[†], **Daniel Lowd**[*], **Dejing Dou**[*]
[*]Computer and Information Science Department, University of Oregon, USA
{javid,lowd,dou}@cs.uoregon.edu
[†]School of Electronic Science and Engineering, Nanjing University, China
{anyirao}@smail.nju.edu.cn

Abstract

Adversarial examples expose vulnerabilities of machine learning models. We propose an efficient method to generate white-box adversarial examples that trick character-level and word-level neural models. Our method, HotFlip, relies on an atomic flip operation, which swaps one token for another, based on the gradients of the one-hot input vectors. In experiments on text classification and machine translation, we find that only a few manipulations are needed to greatly increase the error rates. We analyze the properties of these examples, and show that employing these adversarial examples in training can improve test-time accuracy on clean examples, as well as defend the models against adversarial examples.

1 Introduction

Adversarial examples are inputs to a predictive machine learning model that are maliciously designed to cause poor performance (Goodfellow et al., 2015). Adversarial examples serve several purposes. First, they expose regions of the input space where the model performs poorly, which can aid in understanding and improving the model. By using these examples as training data, adversarial training learns models that are more robust and often perform better on non-adversarial examples. Furthermore, adversarial examples demonstrate vulnerabilities that could be exploited by malicious adversaries in real applications.

Research on adversarial examples for neural nets has largely focused on image data; in the past year however, there is growing interest in understanding vulnerabilities of NLP systems too (Jia and Liang, 2017; Zhao et al., 2017; Belinkov and Bisk, 2017). Previous work in NLP has focused on creating adversarial examples in a *black-box* setting, wherein the attacker can query a model but does not have access to its parameters. We investigate how an attacker can change the output of a system in a *white-box* setting, wherein the adversary has access to model parameters. Black-box attacks often rely on heuristic methods to create adversarial examples. Training models to be robust to these attacks does not guarantee robustness to any other attacks. In contrast, white-box attacks find or approximate the *worst-case* attack for a particular model and input, within some allowed set of perturbations. Therefore, white-box attacks can demonstrate and defend against a model's most serious vulnerabilities, which may not be discovered by black-box heuristics.

This work studies both attack and defense scenarios. We find that only a few manipulations are needed to greatly increase the error rates for models. Furthermore, fast generation of adversarial examples allows feasible adversarial regularization, which helps the model defend against adversarial examples and improve accuracy on clean examples. At the core of our method lies an atomic *flip* operation, which changes one token to another by using the gradients of the model with respect to the one-hot vector input. Naturally, our method can be applied to both character-level models and word-level models. After a few character changes, the meaning of the text is very likely to be preserved or inferred by the reader (Rawlinson, 1976). By contrast, word-level adversarial manipulations are much more likely to change the meaning of text, which makes the use of semantics-preserving techniques necessary. We explore both types of adversarial attacks.

Our contributions are as follows:

1. We propose an efficient gradient-based optimization method to manipulate discrete text structure at its one-hot representation.

2. We conduct a broad set of experiments across

South Africa's historic Soweto township marks its 100th birthday on Tuesday in a mood of optimisim. 57% **World**
South Africa's historic Soweto township marks its 100th birthday on Tuesday in a moo**P** of optimisim. 95% **Sci/Tech**
it's frustrating to see these guys who are obviously pretty clever waste their talent on parodies of things they probably thought were funniest when they were high. 83% **Negative Sentiment**
it's frustrating to see these guys who are obviously pretty **deft** waste their talent on parodies of things they probably thought were funniest when they were high. 65% **Positive Sentiment**

Table 1: White-box adversarial examples with a single character or word change, which will be misclassified by neural classifiers.

text classification and seq-to-seq generation, and analyze the vulnerabilities of each model.

3. We investigate the robustness of a model trained with adversarial examples, by studying its resilience to attacks and its accuracy on clean test data.

2 Related Work

Szegedy et al. (2014) found that a minor change to an image can trick an image classifier. That is, an adversary can trick these models to misclassify examples that are only slightly different from correctly classified examples. This can have severe consequences in applications, such as autonomous cars. Adversarial examples exist the real world; for example, spammers have been modifying their emails to evade spam filters since the late 90s. The need to understand vulnerabilities of NLP systems is only growing. Companies such as Google are using text classifiers to detect abusive language[1], and concerns are increasing over deception (Zubiaga et al., 2016) and safety (Chancellor et al., 2016) in social media. In all of these cases, we need to to better understand the dynamics of how NLP models make mistakes on unusual inputs, in order to improve accuracy, increase robustness, and maintain security or privacy.

In one of the first attempts at tricking deep neural classifiers, Papernot et al., (2016b) added adversarial noise to the word embeddings in an LSTM and searched their neighborhood to find a word to replace the original word. While their adversary was able to trick the classifier, their word-level changes do not preserve the meaning (e.g., "I wouldn't rent this..." → "Excellent wouldn't rent this..."). Jia and Liang (2017) add distracting sentences, which are polished by crowdsourcing, to the end of paragraphs to fool deep reading comprehension systems. They distinguish between the stability and the sensitivity of machine learning models and argue reading comprehension systems are overly stable. Zhao et al. (2017) search for adversarial examples in the space of encoded sentences and generate adversarial examples by perturbing the latent representation. Searching in the lower-dimension latent space is feasible and does not need intervention of the crowd workers, but neural text generation is still limited to short texts.

Belinkov and Bisk (2017) show that character-level machine translation systems are overly sensitive to random character manipulations, such as keyboard typos. Similarly, Hosseini et al. (2017) show that simple modifications, such as adding spaces or dots between characters, can drastically change the toxicity score from Google's `perspective` API.

Adversarial training/regularization interleaves training with generation of adversarial examples (Goodfellow et al., 2015). Concretely, after every iteration of training, adversarial examples are created and added to the mini-batches. A projected gradient-based approach to create adversarial examples by Madry et al. (2017) has proved to be one of the most effective defense mechanisms against adversarial attacks for image classification. Similar regularization techniques have been used for text classification (Li et al., 2016; Miyato et al., 2017), which have focused solely on improving accuracy on clean examples, without creating textual adversarial examples.

3 HotFlip

We devise a beam search optimization method which applies gradient-driven perturbations to the discrete input. HotFlip flips tokens and can be extended to insertion and deletion operations, in order to constitute a more comprehensive set of adversarial attacks to trick an NLP system. This method achieves the goal of fooling the target system by applying perturbations that increase the loss of the model. In addition, when a model employs HotFlip during training, it will become more robust against attacks at test time.

The architecture we study for our character-level experiments is based on the one proposed

[1] https://www.perspectiveapi.com

by Kim et al. (2015) for character-level language modeling. Feature extraction is performed by convolutions over characters, which are passed to layers of highway networks, and finally given to stacks of recurrent neural nets for modeling a sequence of words. This architecture has been used to perform sequence labeling (Kim et al., 2015) and machine translation (Costa-Jussa and Fonollosa, 2016). We adapt it for our text classification experiments, wherein the output of the last recurrent unit is passed to a softmax to predict the label of text. For our word-level text classification experiments, we use Kim's convolutional neural model (2014) which has become a standard baseline for neural text classification.

3.1 Input

Let V be the alphabet, $\mathbf{x}$ be a text of length L characters, and $x_{ij} \in \{0,1\}^{|V|}$ denote a one-hot vector representing the j-th character of the i-th word. The character sequence can be represented by

$$\mathbf{x} = [(x_{11},.. \; x_{1n});..(x_{m1},.. \; x_{mn})]$$

wherein a semicolon denotes explicit segmentation between words. The number of words is denoted by m, and n is the number of maximum characters allowed for a word[2]. For word-based representations, we instead have $x_i \in \{0,1\}^{|V|}$ as a one-hot vector representing the ith word, where V is the word vocabulary. In the following description of our method, we use the character-based representation as our running example.

3.2 Derivatives of Operations

Let $J(\mathbf{x}, \mathbf{y})$ denote the model's loss on input $\mathbf{x}$ with true output $\mathbf{y}$. For example, for classification, J would be the log-loss over the output of the softmax unit. Imagine the adversary is allowed to change r characters in the input text to fool the model to which it has access. Using a brute-force search, it would need to do $\binom{L}{r}|V|^r$ forward passes to exhaust the search space. That is, query the classifier, by calling J, for all combinations of character flips within the allowed budget, r. This can decrease to $\mathcal{O}(brL|V|)$ if beam search is used. That is, after trying all possible single character flips, keep the top b flips which had the highest loss increase, and continue for r steps.

Our algorithm requires just one function evaluation (forward pass) and one gradient computation (backward pass) to estimate the best possible flip. We represent text operations as vectors in the input space and estimate the change in loss by directional derivatives with respect to these operations. Based on these derivatives, the adversary can choose the best loss-increasing direction.

A character **flip** in the j-th character of the i-th word ($a \rightarrow b$) can be represented by this vector:

$$\vec{v}_{ijb} = (\vec{0},..;(\vec{0},..(0,..\text{-}1,0,..,1,0)_j,..\vec{0})_i; \; \vec{0},..)$$

where -1 and 1 are in the corresponding positions for the a-th and b-th characters of the alphabet, respectively, and $x_{ij}^{(a)} = 1$. Due to directional derivatives, we have the following:

$$\nabla_{\vec{v}_{ijb}} J(\mathbf{x}, \mathbf{y}) = \nabla_x J(\mathbf{x}, \mathbf{y})^T \cdot \vec{v}_{ijb}$$

The vector with the HotFlip direction, among all possible character flips in the sequence, can be easily found:

$$\max \nabla_x J(\mathbf{x}, \mathbf{y})^T \cdot \vec{v}_{ijb} = \max_{ijb} \frac{\partial J}{\partial x_{ij}}^{(b)} - \frac{\partial J}{\partial x_{ij}}^{(a)} \tag{1}$$

An immediate benefit of using derivatives with respect to the one-hot vectors is that they can be used to select the best character change ($a \rightarrow b$). Concretely, the derivative vector contains the information about the loss increase obtained due to a character flip. This makes the number of queries be independent of the alphabet size, which is crucial for word-level models with considerably larger alphabet size than character-level models.

Character **insertion**[3] at the j-th position of the i-th word can also be treated as a character flip, followed by more flips as characters are shifted to the right until the end of the word.

$$\max \nabla_x J(\mathbf{x}, \mathbf{y})^T \cdot \vec{v}_{ijb} = \max_{ijb} \frac{\partial J}{\partial x_{ij}}^{(b)} - \frac{\partial J}{\partial x_{ij}}^{(a)} + \sum_{j'=j+1}^{n} \left(\frac{\partial J}{\partial x_{ij'}}^{(b')} - \frac{\partial J}{\partial x_{ij'}}^{(a')} \right) \tag{2}$$

where $x_{ij'}^{(a')} = 1$ and $x_{ij'-1}^{(b')} = 1$.

Similarly, character **deletion** can be written as a number of character flips as characters are shifted

[2] Padding is applied if the number of characters is fewer than the maximum.

[3] For ease in exposition, we assume that the word size is at most n-1, leaving at least one position of padding at the end.

to the left. Since the magnitudes of direction vectors (operations) are different, we normalize by the number of flips in the insertion or deletion operation i.e. $\frac{\vec{v}}{\sqrt{2N}}$ where N is the number of flips.

3.3 Beam Search

We explained how to estimate the best single change in text to get the maximum increase in loss. A beam search of r steps will give us an adversarial example with a maximum of r flips, or more concretely an adversarial example within an L_0 distance of r from the original example. The search is continued until we reach the maximum number of allowed changes, r, or successfully trick the classifier.

Our proposed adversary requires only $\mathcal{O}(br)$ forward passes and an equal number of backward passes. We elaborate on this with an example: Consider the loss function $J(.)$, input x_0, and an individual change c_j. We estimate the score for the change as $\frac{\partial J(x_0)}{\partial c_j}$. For a sequence of 3 changes [c_1,c_2,c_3], the following holds:

$$\text{score}([c_1, c_2, c_3]) = \frac{\partial J(x_0)}{\partial c_1} + \frac{\partial J(x_1)}{\partial c_2} + \frac{\partial J(x_2)}{\partial c_3}$$

where x_1 and x_2 are the modified input after applying [c_1] and [c_1, c_2] respectively. We also experimented with another surrogate loss of the form $J(x_2) + \frac{\partial J(x_2)}{\partial c_3}$, which resulted in an almost identical success rate for the adversary. With a beam width of b, we need b forward and backward passes to compute derivatives at each step of the path, leading to $\mathcal{O}(br)$ queries. In contrast, a loss-based approach requires computing the actual loss for every possible change at every stage of the beam search, leading to $\mathcal{O}(brL|V|)$ queries. Thus, our algorithm limits the number of queries by using derivatives as surrogates for change in loss, and using the summation of the collected gradients at each stage to sort the beam.

To showcase the efficiency of our method, we compare the amount of time it takes to carry out a beam-search method based on querying for the actual loss, and our derivative-based beam-search approach. Because the derivative approach is less exhaustive, it requires a larger beam size, so we set the beam size for the derivative approach to 10, and for the loss-querying approach to 5. The sample that we used for this experiment had 100 examples, an 80-token alphabet, and an average of 44 characters per sentence. Both approaches had a 100% success rate (perfect misclassification) on this sample. It can be seen in Table 2 that the derivative-based approach needs an average of 1 more character flip to trick the classifier, e.g., 34% of examples are misclassified after one change using the derivative-based approach, while this proportion increases to 59% for the loss-based approach. However, the derivative-based is significantly faster.

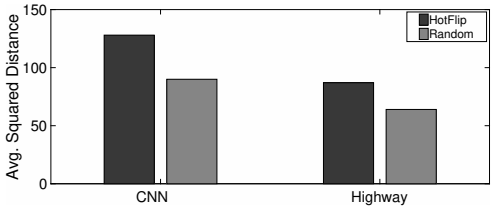

Figure 1: Comparing the HotFlip direction and a random direction based on the average squared distance between the embedding of the original word, and the embedding of the modified word, found from the outputs of the CNN and highway layers.

No. change(s)		1	2	3+
Loss-based	Time (s)	10.3	70.2	705
	Proportion	59%	29%	12%
Gradient-based	Time (s)	**0.11**	**0.93**	**2.7**
	Proportion	34%	29%	37%

Table 2: Comparing beam-search strategies, based on the number of changes (i.e., 1, 2, 3 or more). We report the proportion of successfully created adversarial examples and the time spent to create adversarial examples with 1, 2, and more than three changes. Time is measured in seconds.

3.4 Representations Under Adversarial Noise

To compare changes by the HotFlip direction with a random direction, we measure the average squared distance between the original word representation in each layer (i.e., using the outputs of the CNN and the highway layers) with those after applying changes in each direction. Figure 1 shows a much larger difference between the embedding of the original word and the one modified in the HotFlip direction. In Table 3 , we study the change in the word embedding as a change occurs. We use the output of the highway layer as the word representation. We report the embedding for a few adversarial words, for which the original word is not among their top 5 nearest neighbors.

4 Experiments on HotFlip at Character-Level

We analyze the effect of adversarial examples on character-level classification and translation. Our

past → pas!t	Alps → llps	talk → taln	local → loral	you → yoTu	ships → hips	actor → actr	lowered → owered
pasturing	lips	tall	moral	Tutu	dips	act	powered
pasture	laps	tale	Moral	Hutu	hops	acting	empowered
pastor	legs	tales	coral	Turku	lips	actress	owed
Task	slips	talent	morals	Futurum	hits	acts	overpowered

Table 3: Nearest neighbor words (based on cosine similarity) of word representations from CharCNN-LSTM, picked at the output of the highway layers. A single adversarial change in the word often results in a big change in the embedding, which would make the word more similar to other words, rather than to the original word.

seq-2-seq implementation relies largely on OpenNMT[4], which mostly follows the guidelines of Luong et al. (2015) for attentional translation. Specifically, we used a CharCNN-LSTM encoder and a word-based attentional decoder (Bahdanau et al., 2014). We used a beam size of 5 for decoding at test time. We used a 800k pair from the Europal corpus of German-English WMT data[5], and reproduced the results of Costa-Jussa and Fonollosa (2016). Preprocessing consisted of tokenizing, normalizing punctuation, and filtering sentences with more than 50 words. For text classification, we use the AG's news dataset[6], which consists of 120,000 training and 7,600 test instances from four equal-sized classes: World, Sports, Business, and Science/Technology.

The architecture consists of a 2-layer stacked LSTM with 500 hidden units, and a character embedding size of 25. Both models were trained with stochastic gradient descent and gradient clipping, and the batch size was set to 64. For classification, we used 10% of the training data as the development set, and trained for a maximum of 25 epochs. For translation, we use OpenNMT's suggested hyper-parameters. Throughout our experiments, we only allow character changes if the new word does not exist in the vocabulary, to avoid changes that are more likely to change the meaning of text. A beam size of 5 for translation and a beam size 10 for classification were used. We limited per-word character change to one, which was found to facilitate faster beam search. For both tasks, we use a maximum of 10% of characters in the document as the budget for the adversary. For experiments with random changes, we use the same number of character changes as the one committed by our adversary in that experiment.

4.1 Text Classification

In Figure 2, we plot the success rate of the adversary against an acceptable confidence score for the misclassification. That is, we consider the adversary successful only if the classifier misclassifies the instance with a given confidence score. As can be seen, the beam-search strategy is very effective in fooling the classifier even with a 0.9 confidence constraint, tricking the classifier for more than 90% of the instances. A greedy search is less effective especially in producing high-confidence scores. In this task, 30% of adversarial examples have either one or two changes. The adversary manipulates 4.1% and 5.28% of the characters in a document on average, for 0.5 and 0.9 confidence thresholds respectively. For this task, random changes barely trick the model (2.7% success rate at 0.5 confidence); we performed a beam-search with random gradients which increased the success rate to only 6.2% at 0.5 confidence.

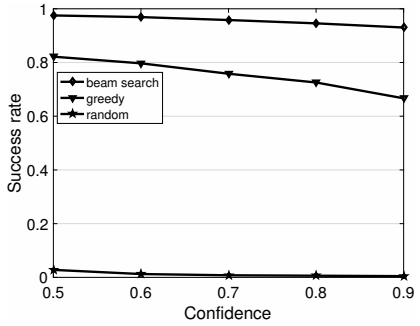

Figure 2: Adversary's success rate as a function of confidence for classification.

4.2 Machine Translation

Unlike classification, for which changes in the predicted labels would imply success for the adversary, a different translation could even have a better BLEU score (Papineni et al., 2002) based on the reference translations in the dataset. To address this issue, we follow a slightly different approach in determining the adversary's success. Figure 3 plots adversary's success rate as a function of the decrease in BLEU score caused by adversarial manipulations. In other words, the translation adversary consults the reference translations, and stops only if the new translation has a lower BLEU score

[4]https://github.com/opennmt/opennmt
[5]http://www.statmt.org/wmt15/translation-task.html
[6]https://www.di.unipi.it/~gulli/

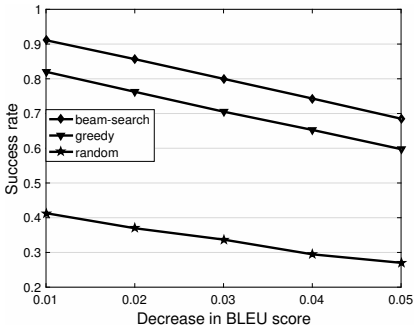

Figure 3: Adversary's success rate as a function of the decrease in BLEU score for EN→DE translation

src	Ich wollte für mein Land kämpfen, aber das wird nicht geschehen.
adv	Ich wollte für meiLn Land kämpfen, aber das wird nicht geschehen.
src-output	I wanted to fight for my country, but it will not happen.
adv-output	I wanted to fight for Chile, but that will not happen.
ref	I wanted to fight for my country but it will not happen.
src	Örtliche Medien machten Russland für den Vorfall verantwortlich.
adv	Örtliche Medien machten RusslaEnd für den Vorfall verantwortlich.
src-output	Local media blamed Russia for the incident.
adv-output	Local media were responsible for the incident.
ref	Local media blamed Russia for the incident.

Table 4: Adversarial examples for DE→EN translations.

by a minimum value. The figure shows that random changes can be much more damaging for the translation system output. Sensitivity of character-level translation systems to noise has also been recently observed by Belinkov and Bisk (2017). This can be (intuitively) explained by the fact that translation is a structured task, and instead of a single response value, a complex output is produced which can make the system much more sensitive. For this task, the adversary manipulates fewer than 3% of characters in a document on average. In fact, 60% of adversarial examples for character-level translation contain only one change.

4.2.1 Discussion

In this section, we discuss some properties of the created adversarial examples. Figure 4 shows the proportion of each operation chosen by the adversary for the experiments. We can see that flip is the dominant operation for classification, and to some extent for EN→DE. Insert is the dominant operation for DE→EN, and delete is almost never used to create adversarial examples for translation.

This phenomenon is related to the results in the previous section, in which we saw the translation models were more brittle. First, note that the insert operation (defined in Equation 2) is a constrained flip operation, wherein a flip at character j is followed by a set of flips to its right. Concretely, except the first flip, which the adversary can pick, the rest of the flips are given by the word's orthography and are thus constraints on the operation. We observed much larger values for the gradients of the one-hot vectors in translation experiments; this indicates that random flips, and consequently the insert operation, can cause a large loss, which is reflected in the high success rate of random changes for translation in Figure 3, and the domination of the insert operation in Figure 4.

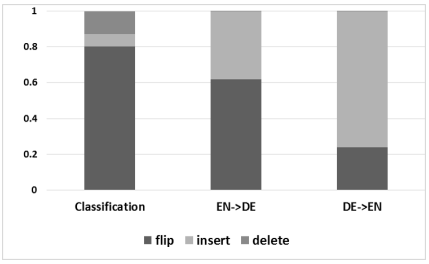

Figure 4: Contrasting adversary's decisions for each task.

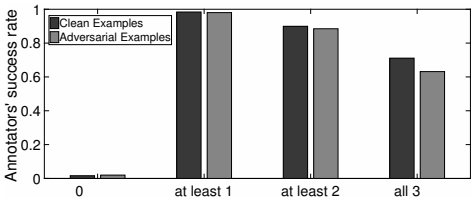

Figure 5: Fraction of clean and adversarial examples correctly labeled by at least 1, at least 2, or all 3 human labelers.

4.3 Human Perception

Our human evaluation experiment shows that character-based adversarial examples are much more likely to preserve the meaning of text than alter it. We conduct an experiment of annotating 600 randomly-picked instances annotated by at least three people. This set contains 150 examples of each class, all of which are correctly classified by the classifier. We manipulate half of this set by our algorithm, which can successfully trick the classifier to misclassify these 300 adversarial examples.

Figure 5 demonstrates that "at least one" crowd worker labels the text correctly 98.3% and 98% of the times for clean and adversarial examples, respectively. The difference between the two sets increases to 7.95% for cases when all three crowd workers label the text correctly. This drop can be explained by the fact that the median accuracy of our participants decreased by 1.78% from 87.49% on clean examples to 85.71% on adversarial examples, making it less probable for correct labeling by all annotators. Similar small drops in human performance have been reported for image classification (Papernot et al., 2016a) and text comprehension (Jia and Liang, 2017).

5 HotFlip at Word-Level

In this section, we create adversarial examples by changing words in text. When changing words, we must take extra care to avoid changing the meaning. For example, changing the word *good* to *bad* changes the sentiment of the sentence *this was a good movie.* In fact, we expect the model to predict a different label after such a change.

In order to create adversarial examples, we need to add constraints so that the resulting sentence is likely to preserve the original meaning; we only flip a word w_i to w_j only if these constraints are satisfied:

1. The cosine similarity between the two words is bigger than a threshold α.
2. The two words have the same part-of-speech.
3. We disallow replacing of stop-words, as for many of the stop-words, it is difficult to find cases where replacing them will still render the sentence grammatically correct. We also disallow changing a word to another word with the same lexeme for the same purpose.

We only use flip for our word-level experiments, and we leave other operations, including insert, delete, and more linguistically-savvy operations, such as passive-voice, etc., to future work.

To find the pair-wise similarity between words, we used the embeddings of `word2vec`, and we used NLTK for part-of-speech tagging. For the target model and domain, we use Kim's CNN (2014) trained for binary sentiment classification on the SST dataset (Socher et al., 2013), which contains a training set of 9,613 sentences and a test set of 1,821 sentences. We found that syntactically similar words such as "good" and "bad" will have high similarities in `word2vec`, though they are semantically opposite for sentiment classification. To alleviate this, we used the fine-tuned embeddings after training on the dataset. Table 5 shows a few adversarial examples with only one word flip. In the second and the fourth examples, the adversary flips a positive word (i.e., *good*, *nice*) with a highly positive words (i.e., *terrific*, *wonderful*) in an overall very negative review.

Given the strict set of constraints, we were able to create only 41 examples (2% of the correctly-classified instances of the SST test set) with one or two flips. We showed each pair of clean and adversarial examples to 5 crowd workers and asked them whether the word change(s) have caused a drift in the sentiment expressed in the clean example. For 32 of the examples, at least 4 of the participants agreed that the meaning had not changed.

We define another type of attack, which we call the UNK attack. This adversary simply replaces a word with one legible but out-of-vocabulary word, which can be achieved by adding some noise to the word, similar to those studied in (Belinkov and Bisk, 2017; Hosseini et al., 2017). By allowing to flip a single word with the UNK token, the adversary's success rate increases to 22.1%.

6 Adversarial Training

We now analyze the robustness of adversarially-trained models. Through adversarial regularization (Goodfellow et al., 2015), we can use our adversarial examples to learn models that are more effective on unseen test data as well as being more robust to such attacks.

For both models, we only use the flip operation. For our character-level experiment, we flip r characters for each training sample, which was set to 20% of the characters in text after tuning, based on the accuracy on the development set. In addition, for faster generation of adversarial examples, instead of performing a beam search we directly apply the top r flips after the first backward pass, simultaneously. Our adversarial training is 4-5 slower than regular training.

We compare adversarial regularization to dropout (Srivastava et al., 2014), applied to the output of the highway networks, which resembles word embedding in a word-level model. As shown in Table 6, adversarial regularization slightly improves accuracy and dramatically improves robustness to an adversary. Figure 6 shows that more flips are required to trick the adversarially-trained model.

For the word-level experiment, we flip one word for each training sample, using two adversaries: one that preserves semantics (SEM), and the other that flips a word to the UNK token. The cosine similarity threshold α for SEM was tuned on the validation set and was set to 0.8. In addition, since this adversary is not effective in creating many adversarial examples at test time, we simply used it as a regularizer on clean data. Note that the inner adversary during training does not have to trick the model, so both techniques can be used for adversarial training. UNK regularizer creates an adver-

one hour photo is an intriguing (**interesting**) snapshot of one man and his delusions it's just too bad it doesn't have more flashes of insight.
'enigma' is a good (**terrific**) name for a movie this deliberately obtuse and unapproachable.
an intermittently pleasing (**satisfying**) but mostly routine effort.
an atonal estrogen opera that demonizes feminism while gifting the most sympathetic male of the piece with a nice (**wonderful**) vomit bath at his wedding.
culkin exudes (**infuses**) none of the charm or charisma that might keep a more general audience even vaguely interested in his bratty character.

Table 5: Adversarial examples for sentiment classification. The bold words replace the words before them.

	baseline	dropout	adv-regularization
misc. error on test data	8.27%	7.81%	**7.65**%
adversary's success rate	98.16%	93.59%	**69.32**%

Table 6: Comparison of the baseline CharCNN-LSTM model, with embedding-dropout, and adversarial regularization on AG's-news dataset.

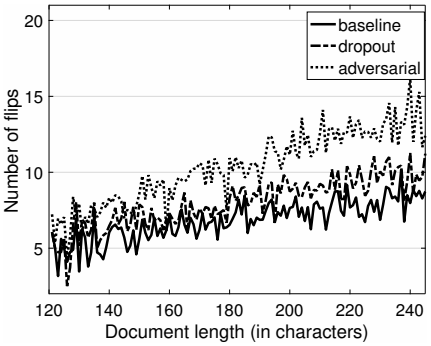

Figure 6: Average number of flips committed by the adversary to attack each model, as a function of document length.

sarial example for each sample in the training set, whereas SEM needs to satisfy the constraints. See Table 7.

Note that in both character-level and word-level models, as demonstrated by our human-subject experiments, adversarial examples could render the text incoherent or change the meaning. Nevertheless, with the use of hyper-parameters (e.g., similarity threshold and number of flips), we can control the level of distortion during adversarial training. For instance, decreasing the similarity threshold to 0.6 produces more but lower-quality adversarial examples and will decrease the accuracy.

	baseline	UNK	SEM
misc. error on test data	13.95%	13.57%	**12.90**%
adversary's success rate (UNK attack)	22.1%	**9.2**%	-

Table 7: Comparison of baseline CNN trained on clean data with adversarial regularization using unknown-token (UNK) or semantics-conserving (SEM) adversaries on SST dataset.

7 Conclusion and Future Work

Our work demonstrates how to create adversarial examples, and how to incorporate them in adversarial training. In order to create adversarial examples, we propose an efficient gradient-based optimization method that flips one token to another, which can increase model's loss on a given clean example. Our human subject study suggests that character-edit operations have little impact on human understanding. The same cannot be said about word-level models, since even a single word can distort the meaning of text or render it ungrammatical. Thus, the character-level adversary is almost unconstrained and can have a much higher success rate. Similarly, we show that a highly semantically-constrained word-level adversary can barely trick a classifier. However, neural word-based classifiers are still sensitive to out-of-vocabulary words. We show adversarial training can improve models' robustness.

We currently use flip for our word-level experiments, and we additionally use insert and delete for our character-level manipulations; these operations can be built on to develop an ensemble of linguistically aware or structurally aware adversaries to further improve robustness.

Our adversary performs untargeted attacks, and simply wants to change model's output. Targeted attacks aim to perturb the input in a direction, which forces the model to give a particular output. This can be done by modifying the adversary's objective function from maximizing the model's loss with respect to the current output, to minimizing the model's loss with respect to the target output. Our gradient-based approach can be easily adapted for such a scenario.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly

learning to align and translate. *arXiv preprint arXiv:1409.0473* .

Yonatan Belinkov and Yonatan Bisk. 2017. Synthetic and natural noise both break neural machine translation. *arXiv preprint arXiv:1711.02173* .

Stevie Chancellor, Jessica Annette Pater, Trustin Clear, Eric Gilbert, and Munmun De Choudhury. 2016. # thyghgapp: Instagram content moderation and lexical variation in pro-eating disorder communities. In *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing*. ACM, pages 1201–1213.

Marta R Costa-Jussa and José AR Fonollosa. 2016. Character-based neural machine translation. *arXiv preprint arXiv:1603.00810* .

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and harnessing adversarial examples. In *Proceedings of ICLR*.

Hossein Hosseini, Sreeram Kannan, Baosen Zhang, and Radha Poovendran. 2017. Deceiving google's perspective api built for detecting toxic comments. *arXiv preprint arXiv:1702.08138* .

Robin Jia and Percy Liang. 2017. Adversarial examples for evaluating reading comprehension systems. *arXiv preprint arXiv:1707.07328* .

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *Proceedings of EMNLP*.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. 2015. Character-aware neural language models. *arXiv preprint arXiv:1508.06615* .

Yitong Li, Trevor Cohn, and Timothy Baldwin. 2016. Learning robust representations of text. In *Proceedings of EMNLP*.

Minh-Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025* .

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2017. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083* .

Takeru Miyato, Andrew M Dai, and Ian Goodfellow. 2017. Adversarial training methods for semi-supervised text classification. In *Proceedings of ICLR*.

Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. 2016a. The limitations of deep learning in adversarial settings. In *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*. IEEE, pages 372–387.

Nicolas Papernot, Patrick McDaniel, Ananthram Swami, and Richard Harang. 2016b. Crafting adversarial input sequences for recurrent neural networks. In *Military Communications Conference, MILCOM 2016-2016 IEEE*. pages 49–54.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*. Association for Computational Linguistics, pages 311–318.

Graham Ernest Rawlinson. 1976. *The Significance of Letter Position in Word Recognition.*. Ph.D. thesis, University of Nottingham.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*. pages 1631–1642.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* 15(1):1929–1958.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. In *Proceedings of ICLR*.

Zhengli Zhao, Dheeru Dua, and Sameer Singh. 2017. Generating natural adversarial examples. *arXiv preprint arXiv:1710.11342* .

Arkaitz Zubiaga, Maria Liakata, Rob Procter, Geraldine Wong Sak Hoi, and Peter Tolmie. 2016. Analysing how people orient to and spread rumours in social media by looking at conversational threads. *PloS one* 11(3):e0150989.