Special Symbol Attacks On NLP Systems

Brian Formento
IDS, I^2R
*NUS, A*Star*
Singapore
brian.formento@u.nus.edu

See-Kiong Ng
IDS
NUS
Singapore
seekiong@nus.edu.sg

Chuan-Sheng Foo
I^2R
*A*Star*
Singapore
foo_chuan_sheng@i2r.astar.edu.sg

***Abstract*—Adversarial attacks/perturbations are becoming important for NLP research, as it has been shown recently that text-attacking adversaries can degrade an NLP model's performance without the victim's knowledge. This has far-reaching implications, especially when an NLP system is deployed in critical applications such as health or finance. In fact, the robustness of state-of-the-art models such as NLP transformers have increasingly been scrutinised due to their vulnerability against adversarial perturbations such as TextFooler and BERT-Attack. These methods, however, focus on changing words, which at times, ruins the readability and semantics of the sample. This paper introduces *Special Symbol Text Attacks* 'SSTA', a technique to improve the performance of language adversarial perturbations using special symbols that have downstream task information associated with them even though that should not have been the case. Our tests show that introducing such symbols which are meaningless to a human within a sentence, can perturb the sample in a particular direction. When this technique is used with TextFooler, which is a recent benchmark for the creation of NLP adversaries, through the TextAttack framework, it can improve all main evaluation metrics on three sentiment classification tasks and fake news detection. A simple, novel and symbol-specific adversarial learning technique is then introduced to reduce the influence of such special symbols.**

***Index Terms*—Adversarial attacks, Robustness, NLP.**

I. Introduction

Adversarial attacks are computational methods designed to fool a machine learning model. They were first introduced in computer vision by perturbing images during training with knowledge of the gradient values [7], [12]. These methods quickly expanded in scale and scope, showing, for example, that an image classification system can be fooled by changing only one-pixel [18]. Adversarial perturbations are a special subclass of adversarial attacks where a sample is edited instead of being designed from scratch. In computer vision, perturbations are highly effective and easy to design, since the optimisation space is continuous, in natural language processing (NLP) however, this is not the case, therefore, creating visually and semantically similar adversarial samples is generally regarded a harder problem. Several works have developed ways to create adversarial perturbations in the context of NLP [9], [13], [14]. They aim to perturb a sample by changing words to semantically similar ones. This, however, has been found to be cumbersome as the percentage of changed words is high, making the sample often unreadable. If online access to the model is available, changing the embedding directly can provide more readable perturbations [20]. By contrast,

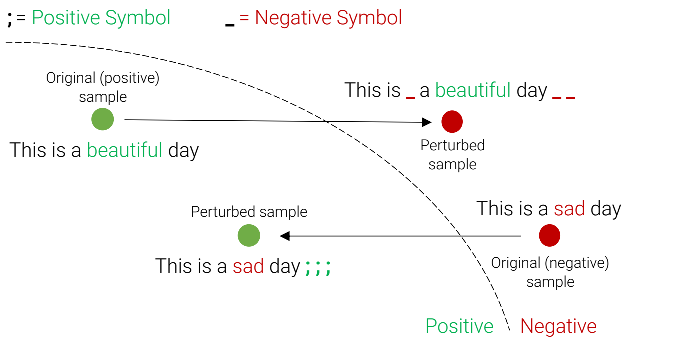


Fig. 1. We devise an adversarial perturbation technique which uses downstream task score (e.g. sentiment) inequalities, associated with special symbols (e.g. punctuation) to weaken a sample before performing another perturbation technique (e.g. TextFooler), therefore improving its performance.

few works have utilized symbols in adversarial perturbations [2], [8]. In this work we show how, although often ignored, symbols contain information used by classifiers, as both during unsupervised pre-training and fine-tuning, the model learns dataset-specific statistics, such as correlations for all tokens due to their co-occurrence, making them inherently biased and can therefore be used in an attack. We first propose a method to discover these *adversarial symbols*, then show how these adversarial symbols can be used to craft perturbations to attack NLP systems that utilize such punctuation/symbols (e.g. for sentiment classification as illustrated in Figure 1). Our attacks leverage the fact that new attention-based models (such as BERT [4]) use punctuation and other special symbols in their processing, as they have been shown to add useful information [10]; they also exploit the fact that automated checks or filters based on semantic similarity encoders [3] depend mostly on the words and not the symbols in the text, therefore enabling our attacks to bypass them. From a signal processing perspective, we hypothesize that in a text sample the added special symbols' embeddings interfere with the other words' embeddings, hence forming a type of symbol/embedding interference that propagates throughout the network. We validate our methods with experiments on two standard tasks, sentiment classification and fake news detection over four datasets. In summary, our contributions are as follows:

- We show how symbols are an attack vector and introduce a technique to discover symbols useful for attacks.

- We introduce two techniques to boost the performance of adversarial attacks on NLP systems using such symbols.
- Our experiments on two tasks (sentiment classification and fake news detection) over four datasets validate the feasibility of symbols as an attack vector.
- To our knowledge this is the first work describing adversarial perturbations that involve changing symbols.

II. Related work

Adversarial attacks on NLP systems can be broadly characterized as being low-level or high-level, where either characters or words are changed respectively to design a successful perturbation. We also separately review punctuation attacks that are most related to our work.

Low-Level Attacks: Several low-level attacks have been summarised in [6] where 10 different attacks have been benchmarked on RoBERTa [15], a more robust counterpart to BERT. Specifically, HotFlip [5] turns each word in a sentence to a one-hot encoded vector and uses knowledge from the character embedding, beam search and information from the back-propagation step, to identify the best letter (hence position in the hot encoded vector) to 'flip' a character in a word. Since the model gradients are required, HotFlip is a white-box attack. High-level attacks often use gradient-free optimization methods such as genetic algorithms and heuristics. Another method known as 'Zeroe' [2] uses knowledge extracted from Wikipedia edit histories to craft adversarial samples that make models fail, as well as being human-comprehensible. The introduced edits are designed to follow perturbations associated with natural human noise, such as phonetic errors, letter omissions and random noise such as mistaking letters adjacent on a keyboard.

High-Level Attacks: High-level attacks perturb words opposed to single characters, in the case of [1] the model's performance was bounded to a fitness landscape and samples perturbed with 'mutations' and 'crossover' at the word level. This paper focuses on semantically similar perturbed adversaries and builds on top of high-level SOTA methods such as Textfooler [9] which first ranks words in a sentence in the order of importance and later changes these words to semantic similar alternatives that fit the surrounding context and forces the target model to make a bad prediction. BERT-Attack [14] works by turning BERT against BERT, where the model first detects words in a sentence that would cause the most disruption if changed and alters them. CLARE [13] introduces a context-aware adversarial scheme, where it uses 3 operations (replace, insert and merge) in a sample to change its end behaviour. Where 'replace' replaces a word to a synonym, 'insert' places a new arbitrary word that is appropriate for the surrounding context, while 'merge' removes two words and replaces them with a single word that holds a similar meaning. In this paper, we focus on Textfooler, but there is no reason why it should not also improve the performance of the other mentioned methods.

Punctuation Attacks: To our knowledge, there is limited work on using punctuation or special symbols for adversarial perturbations, this is likely because traditional models don't process punctuation. Work that uses punctuation is limited to introducing these symbols in between words [8] and was used to fool Google's Perspective model (used to detect offensive language). This type of perturbation, however, has the effect of a tokenizer attack, where the introduced punctuation forces the tokenizer to process two words instead of one. By contrast, our approach is the only work that does not do this, as breaking the word with punctuation is semantically unnatural and is similar to a worst-case character attack. Semantic encoders detect this change as the now distinct words likely will not fit within the surrounding context.

III. Method

A. Problem formulation

Background: Although attention-based models such as BERT [4] are a recent development, these Transformer models are state-of-the-art for many NLP tasks. However, their robustness is still up for debate, as shown by TextFooler [9]. Transformer models are some of the first methods that thanks to large input size, word embeddings and large dictionaries are capable of processing punctuation and non-alphanumeric characters. The use of various non-alphanumeric characters with traditional models has always been rare, as the costs outweighed the benefits, therefore, it was usually discarded as noise. However, studies have shown how including punctuation does indeed increase the downstream performance of NLP tasks, as punctuation can, at times, hold valuable information [10]. Therefore, The practice of using punctuation is likely to stay; the tradeoff between higher accuracy and robustness will continue being an open problem, as pointed out by [19].

Threat Model We focus on the *black-box* attack scenario where access to the model is only through queries for which given input text, a real-valued output between 0 and 1 is returned. We do not assume other knowledge of the model, such as weights, architecture, dictionary or the training datasets.

Optimization Formally, the victim's model can be represented as a task, in this work, for simplicity, the victim is doing binary classification on adversarial sample x_{adv} using function $f(\cdot)$ (for instance, a neural network), the predicted output score P_{adv} is treated as:

$$P_{adv} = f(x_{adv}) \text{ , where } \{P_{adv} \in \mathbb{R} : 0 < P_{adv} < 1\}$$

Ideally, adversarial attacks on binary problems aim to achieve $\max(\mathcal{L}(f(x_{adv})))$ when comparing P_{adv} with the actual label y. $\mathcal{L}$ is defined as:

$$\mathcal{L}(f(x_{adv})) = y \log(P_{adv}) + (1-y)\log(1-P_{adv}) \quad (1)$$

Traditionally, when training, the objective is $\min(\mathcal{L}(f(x_{adv})))$. $\max(\mathcal{L}(f(x_{adv})))$ can be achieved in numerous ways in NLP, such as changing words [13], [14], characters [5] or in our case symbols. Once optimal $\max(\mathcal{L}(f(x_{adv})))$ is achieved, it is possible to record $S_{adv} = x_{adv}$. When maximising $\mathcal{L}$ with symbols, a genetic algorithm or deep learning method can be used to yield optimal results, however, for simplicity 2 heuristic methods with low computation cost are introduced in this paper. This allows us to focus on the idea of symbols

Algorithm 1: Symbol pair discovery

```
Input: Predefined list of special symbols
       $S\prime = \{s1, s2, ..., sn\}$, Clean sample
       $S_{cln} = \{w1, w2, ...\}$, Symbol difference
       threshold $e$.
Output: Symbol pair dataframe $S_{pair}\prime$
1  Initialise empty set $B \leftarrow \{s1 : b1, s2 : b2, ..., sn : bn\}$
2  for each symbol $s_i$ in $S'$ do
3      Compute binary score $b_i = f(s_i)$ where $f(\cdot)$ is the
       victims model.
4      $B[s_i] = b_i$
5  $S_{pair}\prime$
       // symbol pair candidates dataframe
6  for each $s_i, b_i$ in $B$ do
7      for other $s_j, b_j$ in $B$ do
8          $S_{pair}\prime \leftarrow$ Selection($s_i, b_i, s_j, b_j, e$)
9  def Selection($s_i, b_i, b_j, s_j, e$):
10     if $|b_i - b_j| < e$ then
11         Ignore
12     else if $b_i > b_j$ then
13         $ps_{score} = b_i, ps = s_i$
14     else if $b_j < b_i$ then
15         $ns_{score} = b_j, ns = s_j$
16     return $id, ps, ps_{score}, ns, ns_{score}$
```

leaking downstream information while leaving optimal symbol placement for future research.

B. Attack Procedure

The attack can be summarised in 3 steps, the first step attempts to find a list of symbol pairs by using Algorithm 1, where the list is then narrowed down by running Algorithm 2. The best symbols are then used in the second step, to perturb the samples in a given direction. At this point, depending on the achieved results (just using the symbols may be sufficient), step three can be used to further perturb the sample using any high-level attack.

Step 1: Adversarial token discovery This step is designed to first associate a score with each token, then use these scores to find token pair candidates that would best promote a certain class. For sentiment classification it works by creating a predefined list of symbols, each symbol can be parsed through the model and output score recorded. Each symbol will have a score. Simply compare the score of each symbol with one another. If the difference in score is large between two symbols It will likely make a good candidate, hence, chose a threshold where $\{t \in \mathbb{R} \mid 0 < t < 1\}$. As an optional extra step, partition a small portion of the dataset. This sub dataset can be used to further test the validity of the symbol pair by using the **Cascade** method introduced in **Step 2**, This is not by any means optimal, but likely serves a good indication, where more accurate optimization based methods will not justify the cost.

Algorithm 2: Find A_{a-a_i} using SSTA

```
Input: Optional step (small labeled dataset required):
       $X = \{x1, x2, ..., xn\}$ where $n \approx 200$.
       $x_i = \{w1, w2, ...\}$ and $y = \{y1, y2, ..., yn\}$,
       represent the samples and labels respectively.
       $S_{pair}\prime$
Output: Output: Symbols $ps$ (positive) and $ns$
        (negative)
1  $P_{adv} \leftarrow$[]
   /* performance adv samples */
2  for each $ps, ns$ in $S'_{pair}$ do
3      for each $x_i$ in $X$ do
4          $x_{adv} \leftarrow Cascade(x_i), P_{adv} \leftarrow b_{adv} = f(x_{adv})$
5  $A_{a-a_i} \leftarrow$ Use $P_{adv}$ and $y$ through Equation 1 to
   compute the classification accuracy (after-attack
   accuracy) across the dataset
6  Chose the final $ps$ & $ns$ with $min(A_{a-a})$ for all
   symbol pairs
```

TABLE I
SYMBOLS SUCCESSFULLY MAPPED TO EMBEDDINGS REGARDLESS OF TOKENIZER COMPLEXITY (BERT-BASE-UNCASED)

Vector S'											
#	~	$	;	-	.	!	:	_	@	|	&

The after-attack accuracy will highlight if using such a symbol pair does indeed negatively impact the performance.

Step 2: Adversarial token perturbation There are various ways to achieve this. In this paper, we introduce two of the simplest, the **Cascade** and **Replacement** methods. Other methods that achieve better performance use optimization strategies to detect the optimal location to insert the adversarial tokens. These methods, such as genetic algorithms and deep learning, are however behind the scope of this publication and might be released with future research.

- **Cascade (Aggressive)** The original sentence S_{cln} can be views as a vector of tokens $T_{cln} = (t_1, t_2, ...t_n)$, while the perturbed adversarial sentence S_{adv} can be viewed as the vector T_{cln} concatenated with either a positive or negative vector of adversarial tokens $K_{adv} = (k_1, k_2, ...k_n)$, where $k_1 = k_2 = k_n$ and k is either ps or ns. S_{adv} is later parsed through the TextFooler optimization step. An example of the **Cascade** output result is given in Table V, The size of vector K_{adv} is bounded to $D_{S_{adv}} \leq n * tanh(h * \frac{dim(S_{cln})}{n})$. The h and n values are used to control the maximum amount of symbols added after a sentence. This bound was chosen to limit the number of added symbols for short sentences, while also capping to h the number of symbols for large sentences, which makes the perturbed sample more covert. Furthermore, it also allows the tests to be consistent across datasets, samples and methods. In our case, we used $n = 50$ and $h = 50$
- **Replacement (Subtle)** This technique is used to keep the

number of tokens constant between the original sample S_{cln} and the adversarial sample S_{adv}. All symbols, such as punctuation, are replaced directly with the adversarial symbol. Another perturbation technique is then run on S_{adv} . As the experimental section shows, this method guarantees to give better performance than using TextFooler exclusively. It achieves this without using extra tokens on all evaluation metrics.

Step 3: Perturbation optimisation A perturbation algorithm is run on the new adversarial sample produced by **Step 2**. In our case, we use TextFooler [9] through the open-source TextAttack framework [16] to run the optimization procedure.

IV. Experimental evaluation

This section describes the tasks and datasets in more detail, together with the respective results obtained using the adversarial symbols introduced in Table I. It then progresses to qualitative examples, in Table V, showcasing the perturbation produced by the **Cascade** and **Replacement** method and how they affect the performance on various classification datasets (Table VI), where, as expected, the adversarial symbols reduce the A_{a-a}, % of perturbed words and increase the semantic similarity. A practical use case (Fake news detection) is then thoroughly explored. Finally, after pointing out how simply removing all punctuation will not necessarily help the system as a whole in Section IV-E3, we introduce a symbol specific adversarial learning technique which reduces the influence of the adversarial symbol pair, leading to an increase of robustness. We keep the semantic similarity encoder [3] similarity threshold ε consistent across tasks at 0.84.

A. ***Tasks and Datasets***

We performed experiments on four datasets over two tasks - sentiment classification and fake news detection. The dataset details are summarized in Table II. For all the tasks 200 samples (the first and last 100 samples in the test set) have been partitioned to form X: the small dataset used to validate the adversarial symbols.

TABLE II
List of tasks and datasets used

Task	Dataset	Train	Test	Avg Len
Sentiment classification	Yelp	560k	38k	152
	IMDB	25k	25k	215
	MR	9k	1k	20
Fake News	Fake Kaggle	14.5k	6.2k	885

Sentiment Classification: A sentiment classification task attempts to automatically classify a sentence-level or document-level dataset X based on the emotions expressed in the text, each sample S_{cln} is a list of tokens/words $S_{cln} = \{w1, w2, ...\}$. In our case, each S_{cln} is classified to be either negative or positive. We used the following three sentiment classification datasets in our experiments:

- **MR:** The sentence-level rotten tomatoes movie review dataset introduced by [17]. The official opensource TextAttack (*textattack/bert-base-uncased-rotten_tomatoes*) implementation and data have been used. The evaluation is on 1000 samples, 500 for each class.
- **Yelp:** Document-level sentiment classification of yelp reviews [21]. The positive samples are all reviews rated between 4 and 5, while negative reviews are rated between 1 and 2. The official TextAttack (*textattack/bert-base-uncased-yelp-polarity*) implementation and data have been used. The evaluation is on 2000 samples, 1000 for each class.
- **IMDB:** Document-level sentiment review data for movies[1]. The dataset and model have been sourced from the official TextAttack (*textattack/bert-base-uncased-imdb*) opensource implementation. The evaluation is on 2000 samples, 1000 for each class.

Fake News Detection: A fake news detection task analyses how reliable a source of information is. In our case, just from the textual body of a sample S_{cln}, each sample contains a list of words $S_{cln} = \{w1, w2, ...\}$. The goal is for an algorithm to predict whether the sample is either reliable or fake in a binary classification setup. To make the Fake News dataset compatible with BERT and the cascade method, where cascade adds up to 50 tokens at the end of a sentence, only the first 2000 samples that fall between a token range of 0 and 390 have been kept, this is to give ample buffer space to the optimization procedure without resulting in a tokenized sentence larger than 512 tokens.

- **Fake:** Fake news detection from an opensource Kaggle challenge[2], no TextAttack implementation exists yet. Standard training on BERT was used with a 70/30 dataset split. The evaluation is on 2000 samples, 1000 samples for each class.

B. *Symbol discovery*

The first step is identifying which symbols are candidates to attack a model, Step one involves using Algorithm 1 and 2. The discovered symbols for every dataset are given in Table III. As an example, the Yelp model outputted the symbol pairs in Table IV after Algorithm 1. These symbols were used, together with a small dataset and the **Cascade** method to validate them by using Algorithm 2. From the results in the last column in Table IV, it is clear that '$\sim$' and '_' lower the performance the most. These two symbols, however, under the Cascade schema, are not perturbing enough to be used on their own. Therefore TextFooler is used as an extra step. Fig 2 shows 10 samples from the yelp polarity dataset (first 5 and last 5 samples) under perturbation from an ever increasing number of adversarial symbols $0 \rightarrow 40$ and TextFooler.

C. *Evaluation metrics*

Textfooler, BERT-attack and CLARE use after attack accuracy (A_{a-a}), % of perturbed words and semantic similarity to evaluate their method's performance. A_{a-a} and % of perturbed words should be as low as possible, while semantic similarity should be as high as possible.

[1]https://datasets.imdbws.com/
[2]https://www.kaggle.com/c/fake-news/data

TABLE III
SYMBOLS WITH GREATEST SCORE DISCREPANCY

Dataset	ps	ps **score**	ns	ns **symbol**	**threshol** = e
Yelp	$\sim$	0.979	_	0.005	0.5
IMDB	.	0.6	_	0.229	0.3
MR	.	0.89	|	0.1	0.5
Fake	-	0.9993	_	0.991	0.005

TABLE IV
EXAMPLE SYMBOL PAIRS YELP

Yelp dataset symbol discovery				
ps	ps **score**	ns	ns **score**	A_{a-a}
$\sim$	0.979	.	0.09	96.3
$\sim$	0.979	_	0.005	94.25
#	0.975	|	0.006	96.2
&	0.55	:	0.02	96.89

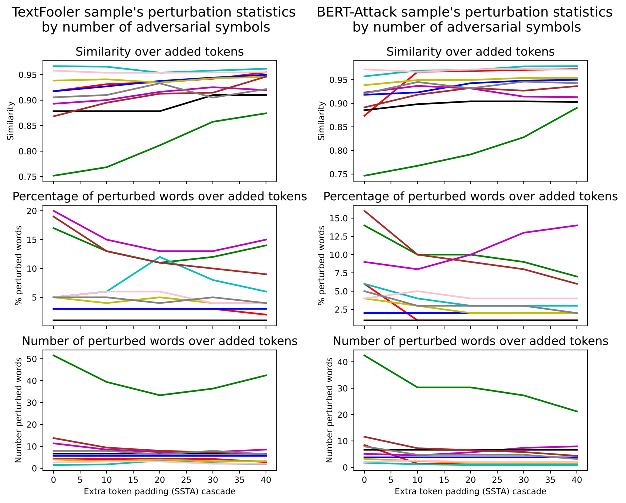


Fig. 2. Ten distinctive samples undergoing the Cascade attack from 0 added symbols to 40 in steps of 10 on Yelp and later running TextFooler (left) BERT-Attack (right)

a) ***After attack accuracy*** *(A_{a-a}):* This method assumes the dataset is balanced and that the classification boundary is well defined (usually at 0.5). After the attack was conducted, it is plausible that the method is overly miss classifying one class compared to the other. What this means is that, for example, if all samples are classified as negative, the A_{a-a} will be approximately 50% as the negative samples are classified correctly, while all positive samples will be miss-classified. Textfooler even mentioned this is a problem, in fact their attack finds it easy to convert real news articles to fake ones but not the opposite. Therefore A_{a-a} will not show the full picture. An alternative is to use ROC curves and visualizing the confusion matrix.

b) ***Percentage of perturbed words****:* This metric calculates the percentage of words that have been changed in a sentence. It is simple but often inappropriate because it does not record how much this word has been changed e.g. in sentiment. BERT-attack exploits this by changing just a few words by a lot e.g. by inverting the word's sentiment.

c) ***Semantic similarity****:* Both TextFooler and BERT-attack use the Universal sentence encoder (USE) [3] to calculate the similarity between the original sample and it is perturbed counterpart. Similar to the problem highlighted when calculating the % of perturbed words, the semantic similarity score remains high if only one word has been heavily changed, compared with more words changed slightly. The USE model has been tested to investigate it's behaviour when only adding punctuation. The embedding values changed but not enough to drop the semantic similarity score below 1. Assuming this holds across our whole dataset it would highlight how our attack vector could take advantage of this weakness to report perfect semantics.

D. Results on Sentiment Classification

1) Adversarial Symbol Discovery: As previously discussed, Algorithm 1 is used to discover a list of adversarial symbol pairs associated with a dataset. An example of this step is given in Table IV, it is a non-exhaustive list of adversarial symbol pairs from Yelp, with a threshold $e = 0.5$, this threshold value is a guide for reproducibility, in practice, the threshold can be decreased gradually until adversarial tokens are found. There are about 23 symbol pairs. Portrayed are a few interesting ones. The symbol pairs are then narrowed down with Algorithm 2 where the symbol pairs are used together with the Cascade method on a 200 sample partition of the Yelp dataset to attack the system. In this case, the second row represents the best symbol pair (due to the lowest $A_{a-a} = 94.25\%$). The optimal symbols, that have the greatest score discrepancy for the 4 main datasets are summarised in Table III. If more complete/complex dictionaries are used by the victim, the symbol list in Table I can be potentially expanded by including symbols in other languages, allowing for better symbol pairs than in Table III.

2) Evaluation of Attack Performance: Results are shown in Table VI. The A_{a-a} when only using adversarial symbols is showcased in the 'Just Symbols (Cascade)' rows. On their own, using the Cascade method, symbol perturbations have only moderate success as an attack. We therefore also run a high-level optimization method on-top of them; we use TextFooler as it's a popular baseline and through Table VI we are able to show how 'Replace + Textfooler' and 'Cascade + TextFooler' achieve better results (values represented in bold) than just using TextFooler where the A_{a-a} and % of perturbed words should be as low as possible, while semantic similarity as high as possible. As an ablation example, 'No Symbols + TextFooler' is a test where all symbols are removed from the sample before running TextFooler. This shows how removing punctuation before doing inference does not necessarily increase robustness. Since the A_{a-a} of IMDB is already low with TextFooler, we have found adding our method has little relative improvement.

3) Qualitative Analysis: Example attacks on the Yelp dataset are shown in Table V. The first example is positive being perturbed to negative, while the second is the other way around,

TABLE V
QUALITATIVE EXAMPLES OF ADVERSARIES FOR A MODEL TRAINED ON YELP-POLARITY. THE **PERTURBED** COLUMN REPRESENTS THE NUMBER OF CHANGED WORDS IN THE SAMPLE BY TEXTFOOLER, WHERE THE ACTUAL WORDS ARE REPRESENTED IN **BOLD**.

Method	**Yelp Polarity Qualitative Examples** **Text**	**Class**	**Score**	**Perturbed**
Original	Sister Sister does amazing work. And they are super fast at what they do. \nI recommend this place for braiding and weaves.	Positive	0.99	N/A
TextFooler	Sister Sister does **fantastic acted**. And they are **nice** fast at what they do. \nI **propositions** this **put** for braiding and **intertwine**.	Negative	0.54	6
(SSTA) Cascade + TextFooler	Sister Sister does amazing work. And they are super fast at what they do. \nI **advise** this place for braiding and weaves. _	Negative	0.64	1
(SSTA) Replace + TextFooler	Sister Sister does **whopping** work _ And they are **funky** fast at what they do _ _ nI **advised** this place for braiding and **intertwine** _	Negative	0.99	4
Original	This place holds a nostalgic appeal for people born and raised in Pittsburgh who grew up eating here. If that experience is what your looking for, please visit. If you're looking for a tasty meal, go somewhere else. 5 stars for history, 0 for food quality and flavor.	Negative	0.99	N/A
TextFooler	This place holds a nostalgic **resorted** for people born and raised in Pittsburgh who grew up eating here. **Though** that experience is what your looking for, please visit. If you're looking for a tasty meal,**expend** somewhere else. 5 stars for tradition, 0 for food quality and flavor.	Positive	0.82	3
(SSTA) Cascade + TextFooler	This place holds a nostalgic appeal for people born and raised in Pittsburgh who grew up eating here. If that experience is what your looking for, please visit. If you're looking for a tasty meal, go somewhere else. 5 stars for history, 0 for **lunches calibre** and flavor. ∼	Positive	0.51	2
(SSTA) Replace + TextFooler	This place holds a nostalgic appeal for people born and raised in Pittsburgh who grew up eating here ∼If that experience is what your looking for ∼ please visit ∼ If you ∼re looking for a tasty meal ∼ go somewhere else ∼5 stars for history ∼ 0 for food quality and flavor ∼	Positive	0.68	0

TABLE VI
RESULTS ON SENTIMENT CLASSIFICATION & FAKE NEWS

Sentiment Classification		**Original Accuracy**	**After Attack Accuracy**	**% of Perturbed Words**	**Semantic Similarity**	**Average text length**	**Average number of tokens**
Just Symbols (Cascade)	**Yelp**	97	94.2	0	1	113	195.5
	IMDB	94.5	93.5	0	1	163.7	264.7
	MR	85.2	77.3	0	1	18.7	67.7
	Fake	99.8	81.7	0	1	257	373
TextFooler	**Yelp**	97	5.3	11.6	0.89	113	145.8
	IMDB	94.5	1.35	9.81	0.91	163.7	214.7
	MR	85.2	9.1	17.7	0.84	18.7	25.8
	Fake	99.8	47.7	31.7	0.828	257	323
No Symbols + TextFooler	**Yelp**	97	4.2	10.8	0.9	115	123
	IMDB	94.5	**0.85**	8.6	0.91	163.7	178
	MR	85.2	7.1	16.6	0.85	18.7	23.2
	Fake	99.8	43.5	20.5	0.895	257	281
(SSTA) Replace + TextFooler	**Yelp**	97	2.8	9.69	0.9	115.6	145.8
	IMDB	94.5	0.95	**8.5**	0.91	163.7	214.7
	MR	85.2	7.3	16.5	0.86	18.7	25.8
	Fake	99.8	36.4	**19.3**	**0.898**	257	326
(SSTA) Cascade + TextFooler	**Yelp**	97	**1.5**	**8.07**	**0.93**	113	195.5
	IMDB	94.5	1	8.65	**0.92**	163.7	264.7
	MR	85.2	**3.7**	**11.9**	**0.91**	18.7	67.7
	Fake	99.8	**35.7**	21.2	0.881	257	373

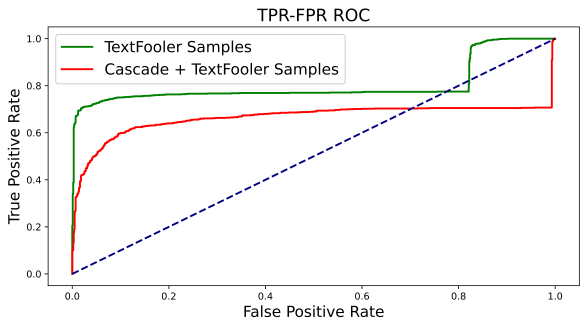
note how for the 'Replace + TextFooler' in the second example replacing all the tokens with the adversarial token is enough to perturb the sample to be positive, therefore TextFooler is not needed to optimize it, this would result in this sample having a perfect similarity score. Unless the sample is already semantically similar we record that TextFooler and/or Bert-Attack needs to change fewer words to achieve comparable results (the downstream trend in the middle and bottom plots) in Figure 2. Theoretically, this is due to the added symbols lowering the amount of energy required to move a sample across the classification boundary. Hence, not only does the optimization procedure often need to change fewer words, but the changed words are also more semantically similar to the original (upward trend in the top plot) in Figure 2. We restrain from further testing our methods on BERT-Attack due to computational costs as at the time of writing, BERT-Attack through the TextAttack framework only worked with the CPU interface.

E. Results on Fake News Detection

1) Evaluation of Attack Performance.: This section does a practical experiment on fake news detection, where we idealize a case study where an attacker wants to propagate a specific message, that was originally picked up by authorities and used to train a fake news detection algorithm. In this scenario, ideally, the message would be kept as similar as possible to increase readability, Adding tokens either at the start or end of the sentence comes at a cheap cost, since the attacker's priority is to propagate the content of the message. In this example, we find adding the '-' symbol with the **Replacement** function is enough to fool the model to predict a reliable news article as fake, changing a fake news article to a reliable one however remains a challenge, a challenge that is non the less simplified with the addition of symbols, portrayed by the confusion matrix in Table VII, where we were able to increase the probability of a fake news sample passing through the filter by 30.2% using the Cascade method. Furthermore, As the test in Table VI shows, by propagating symbols it would allow such attacker to change fewer words than just using TextFooler. It is important to take into account that omitting all punctuation at training reduces the downstream performance for this particular case from 99.8% to 96.2% and training on punctuation while removing punctuation at inference, as shown from the 'No symbol + TextFooler' section in Table VI, is less robust.

2) Analysis Of Attacks.: Just like TextFooler, since our methods build on top of it, the Cascade and Replacement methods find difficult to convert a fake news sample to a real one, This behaviour can be highlighted in Table VII and Figure 3, which are the confusion matrix, TPR-FPR ROC curve and Precision-Recall ROC respectively. None the less, using Cascade and Replacement still improves the performance, this is highlighted by an increase of miss-classified news articles from $225 \to 272 \to 293$ in Table VII, turning more fake news samples to real ones. The results indicate that, across a dataset, there is an increase of 20.9% and 30.2% of a fake news sample being miss-classified by using Replace or Cascade respectively. The robust counterpart is showcased in Table IX.

3) Adversarial Symbol Defence: As discussed at the start of this section, removing punctuation across the learning and inference task has a performance cost (For the Fake experiment $99.8\% \to 96.2\%$ while for sentiment classification a reduction in A_{a-a}, % of perturbed words and an increase in semantic similarity, as pointed out by the 'No Symbols + TextFooler' row in Table VI, compared to running TextFooler on the normal samples), hence, the trade-off between higher original accuracy



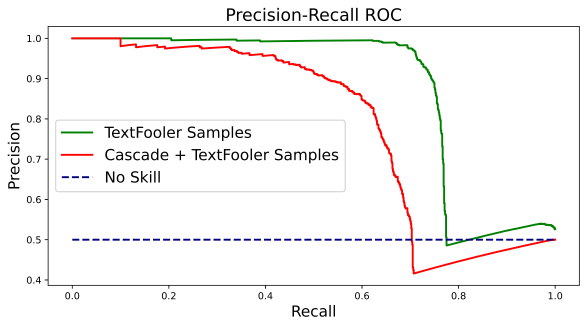


Fig. 3. Results for Fake News Detection Task. **Top:** ROC Curve. **Bottom:** Precison-Recall Curve

being at odds with robustness. A trivial defence mechanism on **Cascade** is to remove all punctuation at the end of a sentence, this however, cannot be done for the **Replacement** or other algorithms that rely on clever symbol placement. Table IX shows how using Algorithm 1 and Algorithm 2 together with adversarial training can alleviate this vulnerability by making the model more robust. After using Algorithm 1 and Algorithm 2 to identify the symbol pair, they are introduced in the training procedure following ideas from [11], but applied to symbols instead of words, where each sample has a probability p of undergoing the **Replacement** operation, where Replacement with the negative and positive token occurs $np = p/2$ and $pp = p/2$ of the time respectively. In this test $p = 0.05$ and BERT was Fine-tuned for one epoch. As Table IX shows The system has become more robust to miss-classifying fake news as real news. This is shown by the Fake vs Predicted Reliable in the 'Replace + TextFooler' column being lower than in Table VII. The system would likely be more robust against the Cascade attack if a similar adversarial schema was introduced specifically for the Cascade adversary. Although the method improves the performance while under attack it slightly decreases normal operating performance. Table VIII can be compared with Table VI.

V. Conclusion

This paper introduces two techniques (Cascade and Replacement) that can be used to boost the performance of TextFooler, generating more perturbing samples that are more semantically similar, we achieve this taking advantage of symbols that although should not, have downstream information associated

TABLE VII

FAKE NEWS CONFUSION MATRIX

	TextFooler			**(SSTA) Replace + TextFooler**			**(SSTA) Cascade + TextFooler**	
Reliable	179	821	Reliable	0	1000	Reliable	7	993
Fake	225	775	Fake	272	728	Fake	293	707
	Predicted Reliable	Predicted Fake		Predicted Reliable	Predicted Fake		Predicted Reliable	Predicted Fake

TABLE VIII

TABLE SHOWING IMPROVED ROBUSTNESS BY APPLYING THE ADVERSARIAL LEARNING SCHEMA INTRODUCED IN SECTION IV-E3

Fake News Robust	**(SSTA) Replace + TextFooler (Non Robust)**	**(SSTA) Replace + TextFooler (Robust)**	**(SSTA) Cascade + TextFooler (Non Robust)**	**(SSTA) Cascade + TextFooler (Robust)**
Original Accuracy	**99.8**	99.4	**99.8**	99.4
After attack accuracy A_{a-a}	36.4	**41.8**	35.7	**37.6**
% of perturbed words	19.3	**21.7**	21.2	**23**
Semantic Similarity	0.898	**0.887**	0.881	**0.874**
Average text length	257	257	257	257
Average number of tokens	326	326	373	373

TABLE IX

CONFUSION MATRIX FOR FAKE NEWS DETECTION TASK WITH ADVERSARIAL TRAINING

	(SSTA) Replace + TextFooler			**(SSTA) Cascade + TextFooler**	
Reliable	0	1000	Reliable	9	991
Predicted Fake	163	837	Predicted Fake	256	744
	Predicted Reliable	Predicted Fake		Predicted Reliable	Predicted Fake

with them. We hope the tests introduced in this paper motivates researchers to introduce symbols in their NLP adversarial attack workflow to improve their performance and ultimately come up with more powerful data augmentation techniques that also protect against adversarial symbols. Alternatively, we have shown how attackers could take advantage of a system that processes punctuation in the pipeline. Therefore, It could be beneficial to simply remove the punctuation, as punctuation is a feasible attack vector.

REFERENCES

[1] Moustafa Alzantot, Yash Sharma, Ahmed Elgohary, Bo-Jhang Ho, Mani Srivastava, and Kai-Wei Chang. Generating natural language adversarial examples. *arXiv preprint arXiv:1804.07998*, 2018.

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

[3] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, et al. Universal sentence encoder. *arXiv preprint arXiv:1803.11175*, 2018.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[5] Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. Hotflip: White-box adversarial examples for text classification. *arXiv preprint arXiv:1712.06751*, 2017.

[6] Steffen Eger and Yannik Benz. From hero to z\'eroe: A benchmark of low-level adversarial attacks. *arXiv preprint arXiv:2010.05648*, 2020.

[7] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[8] Hossein Hosseini, Sreeram Kannan, Baosen Zhang, and Radha Poovendran. Deceiving google's perspective api built for detecting toxic comments. corr abs/1702.08138 (2017). *arXiv preprint arxiv:1702.08138*, 2017.

[9] Di Jin, Zhijing Jin, Joey Tianyi Zhou, and Peter Szolovits. Is bert really robust? a strong baseline for natural language attack on text classification and entailment. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8018–8025, 2019.

[10] Bernard Jones. Exploring the role of punctuation in parsing natural text. *arXiv preprint cmp-lg/9505024*, 1995.

[11] Xin Liu Kai Liu. A robust adversarial training approach to machine reading comprehension. 2020.

[12] A Kurakin, I Goodfellow, and S Bengio. Adversarial examples in the physical world. arxiv 2016. *arXiv preprint arXiv:1607.02533*, 2016.

[13] Dianqi Li, Yizhe Zhang, Hao Peng, Liqun Chen, Chris Brockett, Ming-Ting Sun, and Bill Dolan. Contextualized perturbation for textual adversarial attack. *arXiv preprint arXiv:2009.07502*, 2020.

[14] Linyang Li, Ruotian Ma, Qipeng Guo, Xiangyang Xue, and Xipeng Qiu. Bert-attack: Adversarial attack against bert using bert. *arXiv preprint arXiv:2004.09984*, 2020.

[15] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[16] John Morris, Eli Lifland, Jin Yong Yoo, Jake Grigsby, Di Jin, and Yanjun Qi. Textattack: A framework for adversarial attacks, data augmentation, and adversarial training in nlp. pages 119–126, 2020.

[17] B Pang and L Lee. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. 2005.

[18] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation*, 23(5):828–841, 2016.

[19] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. Robustness may be at odds with accuracy. arxiv. *Machine Learning*, 2018.

[20] Dongxu Zhang and Zhichao Yang. Word embedding perturbation for sentence classification, 2018.

[21] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. *Advances in neural information processing systems*, 28:649–657, 2015.