2018 IEEE Symposium on Security and Privacy Workshops

Black-box Generation of Adversarial Text Sequences to Evade Deep Learning Classifiers

Ji Gao, Jack Lanchantin, Mary Lou Soffa, Yanjun Qi
Department of Computer Science, University of Virginia; {jg6yd,jjl5sw, soffa,yanjun}@virginia.edu

***Abstract*—Although various techniques have been proposed to generate adversarial samples for white-box attacks on text, little attention has been paid to a black-box attack, which is a more realistic scenario. In this paper, we present a novel algorithm, DeepWordBug, to effectively generate small text perturbations in a black-box setting that forces a deep-learning classifier to misclassify a text input. We develop novel scoring strategies to find the most important words to modify such that the deep classifier makes a wrong prediction. Simple character-level transformations are applied to the highest-ranked words in order to minimize the edit distance of the perturbation. We evaluated DeepWordBug on two real-world text datasets: Enron spam emails and IMDB movie reviews. Our experimental results indicate that DeepWordBug can reduce the classification accuracy from 99% to 40% on Enron and from 87% to 26% on IMDB. Our results strongly demonstrate that the generated adversarial sequences from a deep-learning model can similarly evade other deep models.**

I. Introduction

Although deep learning has achieved remarkable results in the field of natural language processing (NLP), including sentiment analysis, relation extraction, and machine translation [1]–[3], a few recent studies pointed out that adding small modifications to text inputs can fool deep classifiers to incorrect classification [4], [5]. Similar phenomenon exist in image classification where adding tiny and often imperceptible perturbations on images could fool deep classifiers. It naturally raises concerns about the robustness of deep learning systems considering that deep learning has become core components of many security-sensitive applications, like text-based spam detection.

Formally, for a given classifier F and test sample $\mathbf{x}$, recent literature defined such perturbations as Δx and the resulting sample $\mathbf{x}'$ as an *adversarial* sample [5]:

$$\begin{aligned} &\mathbf{x}' = \mathbf{x} + \Delta\mathbf{x},\ \|\Delta\mathbf{x}\|_p < \epsilon,\ \mathbf{x}' \in \mathbb{X} \\ &F(\mathbf{x}) \neq F(\mathbf{x}') \textbf{ or } F(\mathbf{x}') = t \end{aligned} \tag{1}$$

Here we denote a machine learning classifier as $F : \mathbb{X} \to \mathbb{Y}$, where X is the sample space, $\mathbf{x} \in \mathbb{X}$ denotes a single sample, and $\mathbb{Y}$ describes the set of output classes. The strength of the adversary, ϵ, measures the permissible transformations. The choice of condition in Eq. (1) indicates two methods for finding adversarial examples: whether they are untargeted ($F(\mathbf{x}) \neq F(\mathbf{x}')$) or targeted ($F(\mathbf{x}') = t$).

The choice of Δ is typically an L_p-norm distance metric. Recent studies [4]–[7] used three norms L_∞, L_2 and L_0. Formally for $\Delta\mathbf{x} = \mathbf{x}' - \mathbf{x} \in \mathbb{R}^p$, the L_p norm is

$$\|\Delta\mathbf{x}\|_p = \sqrt[p]{\sum_{i=1}^{p} |x'_i - x_i|^p} \tag{2}$$

The L_∞ norm measures the maximum change in any dimension. This means an L_∞ adversary is limited by the maximum change it can make to each feature, but can alter all the features by up to that maximum [5]. The L_2 norm corresponds to the Euclidean distance between $\mathbf{x}$ and $\mathbf{x}'$ [6]. This distance can still remain small when small changes are applied to many different features. An L_0 adversary is limited by the number of feature variables it can alter [7].

In addition to targeted/untargeted and Δ choices, a third parameter for categorizing recent methods is whether their assumption of an adversary is black-box or white-box. An adversary may have various degrees of knowledge about the model it tries to fool, ranging from no information to complete information. In the **black box** setting, an adversary is only allowed to query the target classifier and does not know the details of learned models or the feature representations of inputs. Since the adversary does not know the feature set, it can only manipulate input samples by testing and observing outputs. In the **white box** setting, an adversary has access to the model, model parameters, and the feature set of inputs. Similar to the black-box setting, the adversary is not allowed to modify the model itself, or change the training data used to train the model. Most studies of adversarial examples in the literature use the white-box assumption [4], [6]–[8]. One study proposed by [9] showed that it is possible to create adversarial samples that successfully reduce the classification accuracy without knowing the model structure or parameters.

Recent studies have focused on image classification and typically created imperceptible modifications to pixel values through an optimization procedure [4]–[7]. Szegedy et al. [4] first observed that DNN models are vulnerable to adversarial perturbation (by limiting the modification using L_2 norm) and used the Limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) algorithm to find adversarial examples. Their study also found that adversarial perturbations generated from one Convolutional Neural Network (CNN) model can also force other CNN models to produce incorrect outputs. Subsequent papers have explored other strategies to generate adversarial manipulations, including

DOI 10.1109/SPW.2018.00016

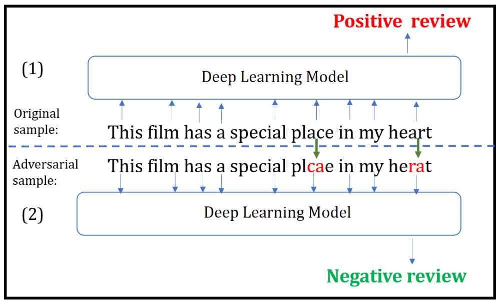


Figure 1: An example of WordBug generated adversarial sequence. Part (1) shows an original text sample and part (2) shows an adversarial sequence generated from the original sample in Part (1). From part (1) to part (2), only a few characters are modified; however this fools the deep classifier to a wrong classification.

using the linear assumption behind a model [5] (by limits on L_∞ norm), saliency maps [7] (by limits on L_0 norm), and evolutionary algorithms [10]. Recently, Carlini et al. proposed a group of attacking methods with optimization techniques to generate adversarial images with even smaller perturbations [6].

Images can be naturally represented as points in a continuous $\mathbb{R}^d$ space (d denotes the total number of pixels in an image). Using an L_p-norm based distance metric to limit the modification of images appears natural and intuitive. However, for text sequence inputs it is hard to search for small text modifications because of the following reasons:

1) Text tokens are categorical features. Imperceptible perturbations using L_p-norms makes sense on continuous pixel values, but not on letters since they are discrete.
2) Each text sample includes a linearly-ordered sequence of words, and the length of sequences varies.

Due to above reasons, the original definition of adversarial modifications: $\Delta\mathbf{x} = \mathbf{x}' - \mathbf{x}$ (from Equation (1)) cannot apply directly to text inputs. One feasible definition of adversarial modifications on text is the edit distance between text x and text x' that is defined as the minimal edit operations that are required to change $\mathbf{x}$ to $\mathbf{x}'$.

A few recent studies [11], [12] defined adversarial perturbations on RNN-based text classifiers. [11] first chose the word at a random position in a text input, then used a projected Fast Gradient Sign Method to perturb the word's embedding vector. The perturbed vector is projected to the nearest word vector in the word embedding space, resulting in an adversarial sequence (adversarial examples in the text case). This procedure may, however, replace words in an input sequence with totally irrelevant words since there is no hard guarantee that words close in the embedding space are semantically similar. [12] used the "saliency map" of input words and complicated linguistic strategies to generate adversarial sequences that are semantically meaningful to a human. However, this strategy is difficult to perform automatically.

We instead design scoring functions to adversarial sequences by making small edit operations to a text sequence such that a human would consider it similar to the original sequence. I.e., the small changes should produce adversarial words which are imperceptibly different from the original words. We do this by first targeting the important tokens in the sequence and then executing a modification on those tokens (defined in Section II) that can effectively force a deep classifier to make a wrong decision. An example of the adversarial sequence we define is shown in Figure 1. The original text input is correctly classified as positive sentiment by a deep RNN model. However, by changing only a few characters, the generated adversarial sequence can mislead the deep classifier to a wrong classification (negative sentiment in this case).

Contributions: This paper presents an effective algorithm, DeepWordBug (or WordBug in short), that can generate adversarial sequences for natural language inputs to evade deep-learning classifiers. Our novel algorithm has the following properties:

- Black-box: Previous methods require knowledge of the model structure and parameters of the word embedding layer, while our method can work in a black-box setting.
- Effective: Using several novel scoring functions, with two real-world text classification tasks our WordBug can fool two different deep RNN models more successfully than the state-of-the-art baseline.
- Simple: WordBug uses simple character-level transformations to generate adversarial sequences, in contrast to previous works that use projected gradient or multiple linguistic-driven steps.
- Small perturbations to human observers: WordBug can generate adversarial sequences that look quite similar to seed sequences.

II. DeepWordBug

For the rest of the paper, we denote samples in the form of pair $(\mathbf{x}, y)$, where $\mathbf{x} = x_1x_2x_3...x_n$ is an input text sequence and $y \in \{1, ..., K\}$ is a label of K classes. A machine learning model is represented as $F : \mathbb{X} \to \mathbb{Y}$, a function mapping from the input set to the label set.

A. Recurrent Neural Networks

Recurrent neural networks (RNN) [13] are a group of neural networks that include a recurrent structure to capture the sequential dependency among items of a sequence. RNNs have been widely used and have been proven to be effective on various NLP tasks including sentiment analysis [14], parsing [15] and translation [16]. Due to their recursive nature, RNNs can model inputs of variable length and can capture the complete set of dependencies among all items being modeled, such as all spatial positions in a text sample. To handle the "vanishing gradient" issue of training basic RNNs, Hochreiter *et al.* [17] proposed an RNN variant called the Long Short-term Memory (LSTM) network that achieves better performance comparing to vanilla RNNs on tasks with long-term dependencies.

B. Word based modification for adversarial sequences

In typical adversarial generation scenarios, gradients are used to guide the change from an original sample to an adversarial sample. However, in the black-box setting, calculating gradients is not available since the model parameters are not observable.

Therefore we need to change the words of an input directly without the guidance of gradients. Consider the vast search space of possible changes (among all words and all possible character changes), we propose to first determine the important words to change, and then modify them slightly by controlling the edit distance to the original sample. More specifically, we need a scoring function to evaluate which words are important and should be changed to create an adversarial sample and a method that can be used to change those words with a control of the edit distance.

To find critical words for the model's prediction in a black-box setting, we introduce a temporal score (TS) and a temporal tail score (TTS). These two scoring functions are used to determine the importance of any word to the final prediction.

We assume the perturbation happens directly on the input words (i.e., not on embedding, or at the "semantic" level). We assume the perturbation approximately minimizes the edit distance to the seed sample. We find an efficient strategy to change a word slightly and is sufficient for creating adversarial text sequences.

In summary, the process of generating word-based adversarial samples on NLP data in the black-box setting is a 2-step approach: (1) use a scoring function to determine the importance of every word to the classification result, and rank the words based on their scores, and (2) use a transformation algorithm to change the selected words.

C. Step 1: Token Scoring Function and Ranking

First, we construct scoring functions to determine which words are important for the final prediction. The proposed scoring functions have the following properties:

- 1. Our scoring functions are able to correctly reflect the importance of words for the prediction.
- 2. Our scoring functions calculate word scores without the knowledge of the parameters and structure of the classification model.
- 3. Our scoring functions are efficient to calculate.

In the following, we explain three scoring functions we propose: temporal score, temporal tail score, and a combination of the two.

1) Temporal Score (TS)

Suppose the input sequence $\mathbf{x} = x_1 x_2 ... x_n$, where x_i represents the word at the i^{th} position. To rank words by importance for prediction, we need to measure the effect of the i^{th} word on the output classification.

In the continuous case (e.g., image), suppose a small perturbation changes x_i to x_i'. The resulting change of prediction output $\Delta_i F(x)$ can be approximated using the partial derivative of this i^{th} feature:

$$\Delta_i F(\mathbf{x}) = (x_i' - x_i) \nabla_{x_i} F(\mathbf{x})$$

However, in a black-box setting, $\nabla_{x_i} F(\mathbf{x})$ is not available. Also in the text case it is difficult to measure $x_i' - x_i$ since words are discrete.

Therefore, we directly measure $\Delta_i F(\mathbf{x})$ by removing the i^{th} word. Comparing the prediction before and after a word is removed reflects how the word influences the classification result. RNNs models words of an input in a sequential (temporal) manner. Therefore we define a so-called temporal score (TS) of the i^{th} word in an input $\mathbf{x}$ as

$$\text{TS}(x_i) = F(x_1, x_2, ..., x_{i-1}, x_i) - F(x_1, x_2, ..., x_{i-1})$$

The temporal score of every word in an input $\mathbf{x}$ can be calculated by one forward pass of the RNN, which is inexpensive.

2) Temporal Tail Score (TTS)

The problem with the temporal score is that it scores a word based on its preceding words. However, words following a word are often important for the purpose of classification. Therefore we define the Temporal Tail Score as the complement of the temporal score. It compares the difference between two trailing parts of a sentence, the one containing a certain word versus the one that does not. The difference reflects whether the word influences the final prediction when coupled with words after itself. The Temporal Tail Score (TTS) of word i is calculated by:

$$\text{TTS}(x_i) = F(x_i, x_{i+1}, x_{i+2}, ..., x_n) - F(x_{i+1}, x_{i+2}, ..., x_n)$$

3) Combined Score

Since the temporal score and temporal tail scores model the importance of a word from two opposition directions of a text sequence, we can combine the two. We calculate the combined scoring function as:

$$\text{Combined Score} = \text{TS} + \lambda(\text{TTS}),$$

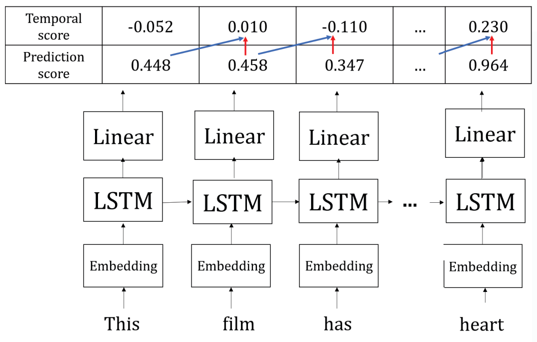
where λ is a hyperparameter.

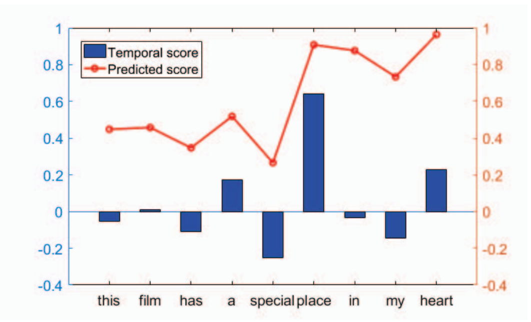
Once we calculate the importance score of each word in an input, we select the top m words to perturb in order to create an adversarial sequence.

D. Step 2: Token Transformer

Previous approaches (summarized in Table V) change words following the gradient direction (gradient of the target adversarial class w.r.t the word), or following some perturbation guided by the gradient. However, in our case there is no gradient direction available. Therefore, we propose an efficient method to modify a word, and we do this by deliberately creating misspelled words.

The key observation is that words are symbolic and learning-based classification programs handle NLP words through a dictionary to represent a finite set of possible words. The size of the typical NLP dictionary is much smaller than the possible combinations of characters at a similar length (e.g., about 26^n for the English case).





(a) Prediction process on an input sentence. (b) The curve of prediction score and temporal score.

Figure 2: Illustration of RNN model prediction process and Temporal Score

Table I: Different transformer functions and their results

Original		Substitute	Swap	Delete	Insert
Team	→	Texm	Taem	Tem	Tezam
Artist	→	Arxist	Artsit	Artst	Articst
Computer	→	Computnr	Comptuer	Compter	Comnputer

This means if we deliberately create misspelled words on important words, we can easily convert those important words to "*unknown*" (i.e., words not in the dictionary). The unknown words are mapped to the "*unknown*" embedding vector in deep-learning modeling. Our results (Section III) strongly indicate that this simple strategy can effectively force RNN models to make a wrong classification.

To create such a misspelling, many strategies can be used. However, we prefer small changes to the original word as we want the generated adversarial sequences and its seed input appear (visually or morphological) similar to human observers. Therefore, we prefer methods with a small edit distance and use the Levenshtein distance [18], which is a metric measuring the similarity between sequences. We propose four similar methods: (1) substitute a letter in the word with a random letter, (2) delete a random letter from the word, (3) insert a random letter in the word, and (4) swap two adjacent letters in the word. The edit distance for the substitution, deletion and insertion operations is 1 and 2 for the swap operation.

These methods do not guarantee the original word is changed to a misspelled word. It is possible for a word to "collide" with another word after the transformation. However, the probability of collision is very small as there are $26^7 \approx 8 \times 10^9$ combinations for 7 letter words without hyphens and apostrophes, but a dictionary often includes no more than 30000 words, making the space very sparse.

The adversarial sample generation of DeepWordBug is summarized in Algorithm 1.

III. Experiments on Effectiveness of Adversarial Sequences

We evaluate the effectiveness of our algorithm by conducting experiments on different RNN models across two real-world NLP datasets. In particular, we want to answer the following research questions: (1). Does the accuracy of deep learning models decrease when feeding the adversarial samples? (2). Does the adversarial samples generated by our method transfers between models?

Algorithm 1 DeepWordBug algorithm with the combined score

Input: Input sequence $\mathbf{x} = x_1x_2 \dots x_n$, RNN classifier $F(\cdot)$, maximum allowed number of words changed m, hyperparameter λ.

```
1: for $i = 1..n$ do
2:     $S_{temporal}(i) = F(x_1x_2...x_i) - F(x_1x_2...x_{i-1})$
3: end for
4: for $i = n..1$ do
5:     $S_{tail}(i) = F(x_{i+1}x_{i+2}...x_n) - F(x_ix_{i+1}...x_n)$
6: end for
7: $S_{combined} = S_{temporal} + \lambda S_{tail}$
8: Sort $S_{combined}$ into an ordered index list: $L_1$ .. $L_n$
9: $\mathbf{x}' = \mathbf{x}$
10: for $i = 1..m$ do
11:     $x'_{L_i}$ = Transform($x'_{L_i}$)
12: end for
13: Return $\mathbf{x}'$
```

A. Experimental Setup

Datasets: In our experiments, we use the Large Movie Review Dataset (IMDB Dataset) [19] and the Enron Spam Dataset [20].

The IMDB Movie Review Dataset contains 50000 highly polarized movie reviews, 25000 for training and 25000 for testing. We train an RNN model to classify the movie reviews into 2 classes: positive and negative.

The Enron Spam Dataset is a subset of the original Enron Email Dataset. The goal is to train a spam filter that can determine whether a certain message is spam or not. We use a subset containing 3,672 ham (i.e. not spam) emails, and 1,500 spam emails.

Details of the datasets are listed in Table II.

Target deep models: To show that our method is effective, we performed our experiments on both uni- and bi-directional LSTMs.

The first model contains a random embedding layer, a uni-directional LSTM with 100 hidden nodes and a fully connected layer for the classification. Without adversarial examples, this model achieves 84% accuracy on the IMDB Dataset and 99% accuracy on the Enron Spam Dataset.

The second model is the same as the first, except with a

Table II: Dataset details

	IMDB Movie Review Dataset	Enron Spam Dataset
Sample type	Movie reviews	Emails
Task	Sentiment analysis	Spam Detection
#Training	25,000	3,672
#Testing	25,000	1,500
Avg. length	215.63 words	148.96 words

Table III: Comparison of the accuracy on different methods, first row shows the original model accuracy in non-adversarial setting. Each row after show the model accuracy on generated adversarial samples of one algorithm. (Number of words changed $m = 20$, The lower score represents a better performance.)

	IMDB		Enron	
	LSTM	bi-LSTM	LSTM	bi-LSTM
No adversary	86.30%	86.70%	99.10%	98.84%
Random	84.00%	84.40%	74.52%	75.48%
Replace-1 score	55.30%	55.00%	82.24%	82.63%
FGSM	66.88%	58.60%	84.94%	80.12%
WordBug - Temporal	72.70%	48.10%	73.17%	72.78%
WordBug - Tail	56.10%	33.20%	69.88%	69.88%
WordBug - Combined	**41.60%**	**25.80%**	**44.79%**	**39.96%**

bi-directional LSTM (also with 100 hidden nodes) instead of uni-directional. Without adversarial examples, it achieves 86% accuracy on the IMDB Dataset and 98% accuracy on the Enron Spam Dataset.

Baselines: We implemented the following attacking algorithms to generate adversarial samples:

- **Projected FGSM:** L_∞ attack from [11]. In our implementation, we use the Fast Gradient Sign Method code from Cleverhans [21], a library developed by the original authors. As we discussed, this method is not black-box.
- **Random + DeepWordBug Transformer:** This technique randomly selects words to change and use our transformer to change the words.

Our method: We use our socring functions to better mutate words. In our implementations, we use different score functions: replace-1 score, temporal score, temporal tail score and the combined score. After that, we use our tranformer to change the words.

Platform: We train the target deep-learning models and implement attacking methods using Keras with Tensorflow as back-end. We use Nvidia GTX Titan cards.

Performance: Performance of the attacking methods is measured by the accuracy of the deep-learning models on the generated adversarial sequences. The lower the accuracy the more effective the attacking method is. Essentialy it indicates the adversarial samples can successfully fool the deep-learning classifier model. The number of words that is allowed for modification is a hyperparameter.

B. Experimental Results on Classification

We analyze the effectiveness of the attacks on two deep models (uni- and bi-directional LSTMs). The results of model accuracy are summarized in Table III. Detailed experimental results at different numbers of allowed word modifications are presented in Figure 3. The results of uni-directional LSTM are in Figure 3 (a)(b), and the results of bi-directional LSTM are in Figure 3 (c)(d).

From Figure 3, we first see that the model has a significantly lower accuracy when classifying the adversarial samples generated by our method on both datasets when compared to the accuracy results from the original test samples. On the IMDB Dataset, changing 20 words per review using WordBug-Combined reduced the model accuracy from 86% to around 41%. As the movie reviews have an average length of 215 words, we consider the 20-word modification as effective. On the Enron Spam Dataset, changing 20 words following WordBug-Combined reduced the model accuracy from 99% to around 44%. For the bi-directional model, changing 20 words on every sequence reduce model accuracy from 86% to around 26% on the IMDB Dataset and from 99% to around 40% on the Enron Spam Dataset. We can see that randomly choosing words to change (i.e., Random in Table III) has little influence on the final result.

Surprisingly our method achieves better results when compared with the projected FGSM which is a white-box attack. The improvement is most likely because the selection of words is more important than how to change the words. Since the projected FGSM selects words randomly, it does not achieve as sound performance as ours.

It is also interesting to compare different score functions that we proposed. On both the IMDB and Enron datasets, the combined score performs notably better than the temporal score and the tail temporal score. It utilizes more information compared to other score functions. The Replace-1 score does not perform well in these datasets, presumably because it does not consider the temporal relationship among words.

C. Transferability of the adversarial sequences

Next, we evaluate the transferability of adversarial sequences generated from our methods. Previous studies have found that transferability is an important property of adversarial image samples: adversarial images generated for a certain DNN model can successfully fool another DNN model for the same task, i.e., transferred to another model.

We use the combined score and the substitution transformer to generate adversarial samples. The number of words we change is 20. The results in Table IV are acquired by feeding adversarial sequences generated by one RNN model to another RNN model on the same task.

Table IV: Result of the transferability of WordBug: The values are the accuracy of the target model tested on the adversarial samples. Different from LSTM1 and Bi-LSTM1 which are trained with randomly-initialized embedding, LSTM2 and Bi-LSTM2 are models trained with a pretrained word embedding.

From \Target at	LSTM1	Bi-LSTM1	LSTM2	Bi-LSTM2
LSTM1	41.60%	37.50%	57.90%	60.70%
Bi-LSTM1	38.20%	25.80%	49.60%	50.60%
LSTM2	39.10%	30.90%	41.40%	44.70%
Bi-LSTM2	40.70%	32.60%	45.10%	44.60%

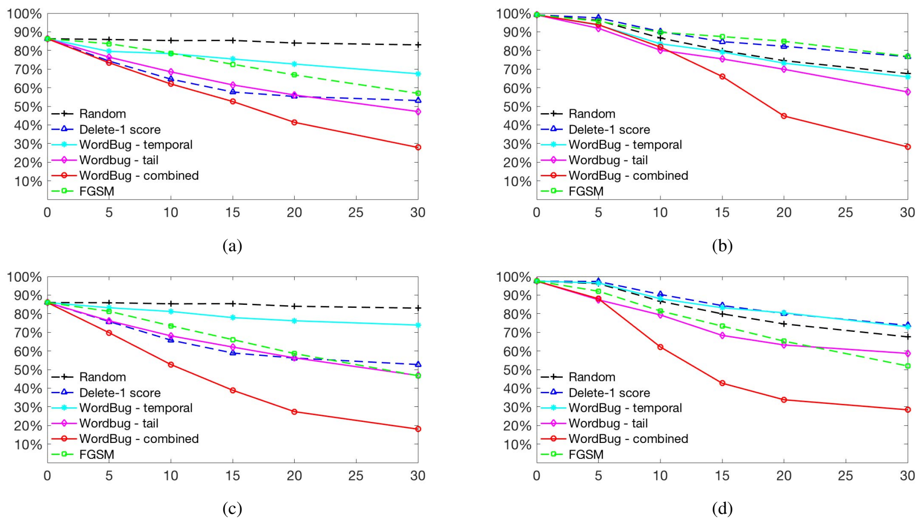


Figure 3: Experiment results. The X axis represents the number of modified words, and the Y axis corresponds to the test accuracy on adversarial samples generated using the respective attacking methods. (a) Uni-directional LSTM on the IMDB Dataset (b) Uni-directional LSTM on the Enron Spam Dataset (c) Bi-directional LSTM on IMDB Dataset (d) Bi-directional LSTM on the Enron Spam Dataset

Table V: Prior works

	Adversary	Distance	Space	Modifications
Ours	Black-box	Edit (L_0)	Input space	Swapping two characters
[11]	White-box	L_∞	Embedding space	Gradient + Projection
[12]	White-box	Num. words modified (L_0)	Input space	Complicated & Linguistic-driven

From the table, we see that most adversarial samples can successfully transfer to other models, even to those models with different word embeddings. This experiment demonstrates that our method can successfully find those words that are important for classification and the transformation is effective across multiple models.

IV. Connecting to Previous Studies

Compared to studies of adversarial examples on images, little attention has been paid on generating adversarial sequences on text. We compare the most relevant two and ours in Table V. (1) Papernot et.al., applied gradient-based adversarial modifications directly to NLP inputs targeting RNN-based classifiers in [11]. The resulting samples are called "adversarial sequence," and we also adopt the name in this paper. The study proposed a white-box adversarial attack called projected Fast Gradient Sign Method and applied it repetitively to modify an input text until the generated sequence is misclassified. It first randomly picks a word, and then uses the gradient to generate a perturbation on the corresponding word vector. Then it maps the perturbed word vector into the nearest word based on Euclidean distance in the word embedding space. If the sequence is not yet misclassified, the algorithm will then randomly pick another position in the input. (2) Recently, [12] used the embedding gradient to determine important words. The technique then uses heuristic driven rules together with hand-crafted synonyms and typos. Differently, from ours, this study is a white-box attack because it accesses the gradient of the model. (3) Another paper [22] measures the importance of each word to a specific class using the word frequency from that class's training data. Then the study uses heuristic driven techniques to generate adversarial samples by adding, modifying or removing important words. Differently, this method needs to access a large set of labeled samples.

In summary, previous approaches do not apply to black-box settings. Besides previous approaches mostly used heuristic-driven and complicated modifications. We summarize the differences between our method and the previous studies on generating adversarial text samples in Table V. Our method is black-box while previous approaches all used the stronger white-box assumption. Our method uses the edit distance at the sequence input space to search for the adversarial perturbations. Also, our modification algorithm is simpler compared to previous approaches.

V. Conclusion

In this paper we introduce a vulnerability with deep learning models for text classification. We present a novel framework, DeepWordBug to generate adversarial text sequences that can mislead deep learning models by exploiting this vulnerability. Our method has the following advantages:

- Black-box: DeepWordBug generates adversarial samples in a black-box manner.
- Performance: While minimizing edit distance (approximately minimized), DeepWordBug achieves better performance comparing to baseline methods on two NLP datasets across multiple deep learning architectures.

Our experimental results indicate that DeepWordBug results in about 70% decrease from the original classification accuracy for two state-of-the-art word-level LSTM models across two different datasets. We also demonstrate that the adversarial samples generated on one model can be successfully transferred to other models, reducing the target model accuracy from around 90% to 30-60%.

References

[1] X. Zhang, J. Zhao, and Y. LeCun, "Character-level convolutional networks for text classification," in *Advances in neural information processing systems*, 2015, pp. 649–657.

[2] M. Miwa and M. Bansal, "End-to-end relation extraction using lstms on sequences and tree structures," *arXiv preprint arXiv:1601.00770*, 2016.

[3] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[4] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," in *International Conference on Learning Representations (ICLR)*, 2014.

[5] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[6] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *Security and Privacy (SP), 2017 IEEE Symposium on*. IEEE, 2017, pp. 39–57.

[7] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *IEEE European Symposium on Security and Privacy (EuroS&P)*, 2016.

[8] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and Harnessing Adversarial Examples," *arXiv:1412.6572 [cs, stat]*, Dec. 2014, arXiv: 1412.6572. [Online]. Available: http://arxiv.org/abs/1412.6572

[9] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against deep learning systems using adversarial examples," *arXiv preprint arXiv:1602.02697*, 2016.

[10] A. Nguyen, J. Yosinski, and J. Clune, "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," in *CVPR*. IEEE, 2015.

[11] N. Papernot, P. McDaniel, A. Swami, and R. Harang, "Crafting adversarial input sequences for recurrent neural networks," in *Military Communications Conference, MILCOM 2016-2016 IEEE*. IEEE, 2016, pp. 49–54.

[12] S. Samanta and S. Mehta, "Towards crafting text adversarial samples," *arXiv preprint arXiv:1707.02812*, 2017.

[13] D. E. Rumelhart, G. E. Hinton, R. J. Williams *et al.*, "Learning representations by back-propagating errors," *Cognitive modeling*, vol. 5, no. 3, p. 1, 1988.

[14] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Ng, and C. Potts, "Recursive deep models for semantic compositionality over a sentiment treebank," in *Proceedings of the 2013 conference on empirical methods in natural language processing*, 2013, pp. 1631–1642.

[15] R. Socher, C. C. Lin, C. Manning, and A. Y. Ng, "Parsing natural scenes and natural language with recursive neural networks," in *Proceedings of the 28th international conference on machine learning (ICML-11)*, 2011, pp. 129–136.

[16] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[17] S. Hochreiter and J. Schmidhuber, "Long short-term memory," vol. 9, no. 8. MIT Press, 1997, pp. 1735–1780.

[18] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," in *Soviet physics doklady*, vol. 10, no. 8, 1966, pp. 707–710.

[19] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*. Association for Computational Linguistics, 2011, pp. 142–150.

[20] V. Metsis, I. Androutsopoulos, and G. Paliouras, "Spam filtering with naive bayes-which naive bayes?" in *CEAS*, vol. 17, 2006, pp. 28–69.

[21] N. Papernot, I. Goodfellow, R. Sheatsley, R. Feinman, and P. McDaniel, "cleverhans v1.0.0: an adversarial machine learning library," *arXiv preprint arXiv:1610.00768*, 2016.

[22] B. Liang, H. Li, M. Su, P. Bian, X. Li, and W. Shi, "Deep text classification can be fooled," *arXiv preprint arXiv:1704.08006*, 2017.