MCTSBUG: GENERATING ADVERSARIAL TEXT SEQUENCES VIA MONTE CARLO TREE SEARCH AND HOMOGLYPH ATTACK

Anonymous authors
Paper under double-blind review

ABSTRACT

Crafting adversarial examples on discrete inputs like text sequences is fundamentally different from generating such examples for continuous inputs like images. This paper tries to answer the question: under a black-box setting, can we create adversarial examples automatically to effectively fool deep learning classifiers on texts by making imperceptible changes? Our answer is a firm yes. Previous efforts mostly replied on using gradient evidence, and they are less effective either due to finding the nearest neighbor word (wrt meaning) automatically is difficult or relying heavily on hand-crafted linguistic rules. We, instead, use Monte Carlo tree search (MCTS) for finding the most important few words to perturb and perform homoglyph attack by replacing one character in each selected word with a symbol of identical shape. Our novel algorithm, we call MCTSBug, is black-box and extremely effective at the same time. Our experimental results indicate that MCTSBug can fool deep learning classifiers at the success rates of 95% on seven large-scale benchmark datasets, by perturbing only a few characters. Surprisingly, MCTSBug, without relying on gradient information at all, is more effective than the gradient-based white-box baseline. Thanks to the nature of homoglyph attack, the generated adversarial perturbations are almost imperceptible to human eyes.

1 INTRODUCTION

Recent studies have shown that adding small modifications to continuous inputs can fool state-of-the-art deep classifiers, resulting in incorrect classifications (Szegedy et al., 2014; Goodfellow et al., 2014). This phenomenon was first formulated as adding tiny and often imperceptible perturbations on image pixel values that could fool deep learning models to make wrong predictions. It raises concerns about the robustness of deep learning systems, considering that they have become core components of many safety-sensitive applications. For a given classifier F and a test sample $\mathbf{x}$, recent literature defined such perturbations as vector $\Delta\mathbf{x}$ and the resulting sample $\mathbf{x}' = \mathbf{x} + \Delta\mathbf{x}$ as an adversarial example (Goodfellow et al., 2014).

Deep learning has achieved remarkable results in the field of natural language processing (NLP), including sentiment analysis, relation extraction, and machine translation (Zhang et al., 2015; Miwa & Bansal, 2016; Wu et al., 2016). In contrast to the large body of research on adversarial examples for image classification (Goodfellow et al., 2014; Szegedy et al., 2014; Papernot et al., 2016b; Carlini & Wagner, 2017), less attention has been paid on generating adversarial examples on texts. A few recent studies defined adversarial perturbations on deep RNN-based text classifiers (Papernot et al., 2016c; Samanta & Mehta, 2017). (Papernot et al., 2016c) 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 then projected to the nearest word vector in the word embedding space, resulting in an adversarial sequence (adversarial examples in the text case; we adopt the name in this paper). 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. (Samanta & Mehta, 2017) used the "saliency map" of input words and complicated linguistic strategies to generate adversarial sequences that are semantically meaningful to humans. However, this strategy is difficult to perform automatically. (Gao et al., 2018) proposed greedy scoring strategies to rank words in a text sequence and then applied simple character-level

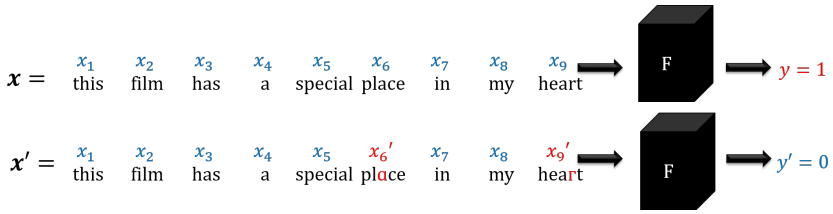


Figure 1: An example of MCTSBug generated black-box adversarial sequence. $\mathbf{x}$ shows an original text sample and $\mathbf{x}'$ shows an adversarial sequence generated from $\mathbf{x}$. From $\mathbf{x}$ to $\mathbf{x}'$, only two characters are modified. This fools the deep classifier to return a wrong classification of sentiment (from positive to negative).

transformations like swapping to fool deep classifiers. Its central idea, minimizing the edit distance of the perturbation makes sense. However, the perturbations are quite visible and the empirical effectiveness needs improvements. We provide more discussions in Section 5.1.

Crafting adversarial examples on discrete text sequences is fundamentally different from creating them on continuous inputs like images or audio signals. Continuous input such as 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 $\Delta\mathbf{x}$ on images appears natural and intuitive. However, for text inputs searching for small text modifications is difficult, because it is hard to define the distance between two discrete sequences. Three possible choices exist:

- Because deep learning NLP models usually use an embedding layer to map discrete inputs to a continuous space (gradients on the embedding layer is calculable). Therefore we can measure the distance among text inputs in the continuous space defined by the word2vec embedding (Pennington et al., 2014). However, state-of-the-art embedding models are still unsatisfactory especially in terms of providing nearest neighbors among words. Figure 5 shows a few examples we chose based on the GloVe (Pennington et al., 2014). Two words close to each other in the GloVe embedding cannot guarantee they are similar, for example, they can be antonyms. For instance, the word "loved" is the nearest neighbor of the word "hated".
- We can use the huge body of linguistic knowledge to measure the distance between two text inputs. However, this strategy is hard to generalize and is difficult to extend to other discrete spaces.
- Shown in Figure 1, we can also use the edit distance between text $\mathbf{x}$ and text $\mathbf{x}'$ being defined as the minimal edit operations that are required to change $\mathbf{x}$ to $\mathbf{x}'$. We focus on this distance in the paper. Intuitively we want to find imperceptible perturbations on a text input (with respect to human eyes) to evade deep learning classifiers.

The second major concern we consider in this paper is the black-box setup. An adversary may have various degrees of knowledge about a deep-learning classifier it tries to fool, ranging from no information to complete information. Figure 6 depicts the Perspective API (Jigsaw, 2017) from Google, which is a deep learning based text classification system that predicts whether a message is toxic or not. This service can be accessed directly from the API website that makes querying the model uncomplicated and widely accessible. The setting is a black-box scenario as the model is run on cloud servers and its structure and parameters are not available. Many state-of-the-art deep learning applications have the similar system design: the learning model is deployed on the cloud servers, and users can access the model via an app through a terminal machine (frequently a mobile device). In such cases, a user could not examine or retrieve the inner structure of the models. We believe that the black-box attack is generally more realistic than the white-box. Previous efforts about adversarial text sequences mostly replied on using gradient evidence. We instead assume attackers cannot access the structure, parameters or gradient of the target model.

Considering the vast search space of possible changes (among all words/characters changes) from $\mathbf{x}$ to $\mathbf{x}'$, we design a search strategy based on Monte Carlo tree search (MCTS) for finding the most important few words to perturb. The search is conducted as a sequential decision process and aims

to make small edit operations such that a human would consider the generated $\mathbf{x}'$ (almost) the same as the original sequence. Inspired by the homoglyph attack (Gabrilovich & Gontmakher, 2002) that attacks characters with symbols of identical shapes, we replace a character in those important words found by MCTS with its homoglyph character (of identical shape). This simple strategy can effectively forces a deep classifier to a wrong decision by perturbing only a few characters in a text input.

Contributions: This paper presents an effective algorithm, MCTSBug , that can generate adversarial sequences of natural language inputs to evade deep-learning classifiers. The techniques we explore here may shed light on discovering the vulnerability of using deep learning on other discrete inputs. 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: on seven real-world text classification tasks, our MCTSBug can fool two state-of-the-art deep RNN models with the success rate of 95% on average (see Figure 4).
- Simple: MCTSBug uses simple character-level transformations to generate adversarial sequences, in contrast to previous works that use projected gradient or multiple linguistic-driven steps.
- Almost imperceptible perturbations to human observers: MCTSBug can generate adversarial sequences that visually identical to seed sequences.

Att: 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 including n tokens (each token could be either a word or a character in different models) and y set including $\{1, ..., K\}$ is a label of K classes. A deep learning classifier is represented as $F : \mathbb{X} \to \mathbb{Y}$, a function mapping from the input space to the label space.

1.1 Background of Adversarial Examples

Basics of Adversarial Examples: Formally, we can define *adversarial* sample(Goodfellow et al., 2014) via the following equation:

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

Here we denote a machine learning classifier as $F : \mathbb{X} \to \mathbb{Y}$, where $\mathbb{X}$ is the sample space, $\mathbf{x} \in \mathbb{X}$ denotes a single sample and $\mathbb{Y}$ describes the set of output classes. $\Delta\mathbf{x}$ describes the perturbation vector and the resulting sample $\mathbf{x}'$ is an *adversarial* example (Goodfellow et al., 2014). 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 ($F(\mathbf{x}') = t$) (Barreno et al., 2006).

The choice of Δx is typically an L_p-norm distance metric. Recent studies (Szegedy et al., 2014; Goodfellow et al., 2014; Carlini & Wagner, 2017; Papernot et al., 2016b) used three norms L_∞, L_2 and L_0. Formally for $\Delta\mathbf{x} = \mathbf{x}' - \mathbf{x} \in \mathbb{R}^d$, the L_p norm is $\|\Delta\mathbf{x}\|_p = \sqrt[p]{\sum_{i=1}^{p} |x'_i - x_i|^p}$. 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 (Goodfellow et al., 2014). The L_2 norm corresponds to the Euclidean distance between $\mathbf{x}$ and $\mathbf{x}'$ (Carlini & Wagner, 2017). 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 (Papernot et al., 2016b).

There exist a large body of recent studies focusing on adversarial examples for image classification that typically created imperceptible modifications to pixel values through an optimization procedure (Goodfellow et al., 2014; Szegedy et al., 2014; Papernot et al., 2016b; Carlini & Wagner, 2017). We put more details in Section 5.4.

Black or White Box Adversarial Examples: A third parameter for categorizing recent methods, in addition to targeted/untargeted and Δ choices, is whether the 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 a classification model's 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 still not allowed to modify the model itself or change the training data. Most studies of adversarial examples use the white-box assumption (Szegedy et al., 2014; Goodfellow et al., 2014; Carlini & Wagner, 2017; Papernot et al., 2016b).

Multiple recent studies extended adversarial examples to the black-box. One study proposed by (Papernot et al., 2016a) showed that it is possible to create adversarial samples that successfully reduce the classification accuracy without knowing the model structure or parameters on image classification tasks. The method is to estimate a local model and attack the local model instead. Another paper (Chen et al., 2017) generates black-box adversarial sample by estimating the gradient directly via queries. (Ilyas et al., 2018a) follows this direction, proposes three different threat models of black-box attack, and uses statistical gradient estimation. (Ilyas et al., 2018b) uses prior information about images and bandit to optimize the gradient estimation and adversarial sample search process.

2 METHOD

We design a method we call MCTSBug to generate adversarial modifications on a text sequence directly, without the guidance of gradients. Considering that the natural form of English text has white-spaces separating words, we treat the search of perturbations as a two-stage task:

- **Step 1:** Determine the important words to change.
- **Step 2:** Modify the important words slightly by creating "imperceivable" modifications.

To find the important words, we formalize the problem as a search problem and use Monte Carlo Tree Search to achieve a good approximate to the global maximum. We then use homoglyph characters (Gabrilovich & Gontmakher, 2002) to create stable and effective imperceivable modifications. On the other hand, these modifications create big difficulties (i.e. large changes) to a target deep learning model.

2.1 HOMOGLYPH ATTACK TO ADD "IMPERCEIVABLE" PERTURBATION ON A WORD

We first assume that words in a text input contribute differently to the final prediction made by a deep classifier. In Section 2.2, we propose a search process to find those important words through MCTS.

After we find an important word, we need to use an efficient strategy to modify it by making imperceptibly changes (to human) while at the same time such changes can force a deep model's output classification of the whole text sequence to change. We therefore define such a small change on a word as one character modification based on homoglyph attack (Gabrilovich & Gontmakher, 2002). Homoglyph attack is an attack based on symbols with identical shapes. Figure 2 shows a table including all the text characters together with its homoglyph pair we use. In complicated modern computer character systems such as Unicode, many symbols have the very similar form to human observers but coming from different origins. For example, Cyrillic letter 'a' and the Latin letter 'a' are visually identical in every ways. However, computer systems treat them as different characters. Homoglyph attacks, though simple, have created a severe problem in cyber security when attackers spoof the domain name with homoglyphs of influential websites.

Words are symbolic, and learning-based classifiers handle words through a dictionary to represent a finite set of possible words. However, the size of the typical NLP dictionary is much smaller than the huge space built by the combinations of all possible characters. This means if we deliberately create (visually) similar but mis-spelled version of important words, we can easily convert those crucial words to "*unknown*" (i.e., words not in the dictionary). Unknown words are mapped to

the "*unknown*" embedding vector, which is likely to be vastly different from the embedding of the original word. Our empirical results (Section 3) strongly indicate that this simple strategy can effectively force deep learning models to make wrong classifications.

Based on our empirical results, changing one character to its homoglyph pair is enough to make a word to be unrecognizable ("*unknown*") by a deep learning system. We say a word is "neutralized" to mean a deep learning system loses the information of this word. In the following, when we say "to neutralize a certain word", we mean to modify a random character in that word to its homoglyphs pair in Figure 2.

-	-	0	O	i	l	s	s
9	۹	'	`	j	j	t	t
8	8	a	ɑ	k	*k*	u	u
7	7	b	Ь	l	l	v	ν
6	6	c	c	m	m	w	w
5	5	d	d	n	n	x	×
4	4	e	e	o	o	y	y
3	3	f	f	p	p	z	z
2	2	g	g	q	q		
1	l	h	h	r	r		

Figure 2: Homoglyph character pairs used in the attack.

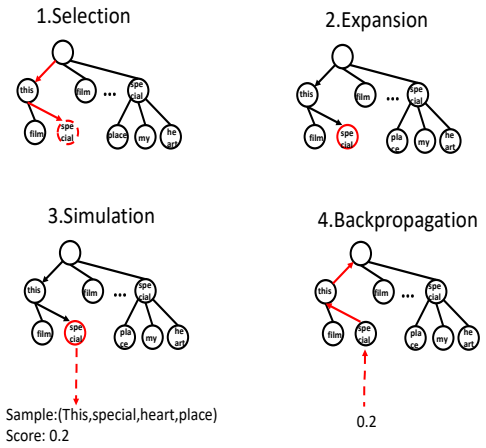


Figure 3: Steps of MCTS based sequential selection of adversarial sequences.

Function MCTS (f_s, A, l, d, R) **:**
  Input : Score function f_s, Action list A, class c, maximum search depth d, maximum number of iterations R
  Output: Best action a
  Initialization T with an empty root;
  while *not reach maximum iteration* R **do**
    s = root;
    while *n is in T* **do**
      $s = \arg\max_{s' \in s.child} v_{s'} + UCB_{s'}$;
      # *jump to next node*
    end
    s_T = random_rollout(s);
    $v_{s_T} = f_s(s_T, c)$;
    while $s \neq$ *root* **do**
      $v_s = v_s \times \frac{n_s}{n_s+1} + v_{s_T} \times \frac{1}{n_s+1}$;
      s = s.parent;
    end
  end
  $s' = \arg\max_{s' \in root.child} v_s + UCB_{s'}$;
  return $a_{out} = s'.action$;
Function MCTSBug (X, d, c_l, R) **:**
  Input : Sample $X = x_1 x_2 .. x_n$, maximum search depth d, classification of the sample c_l, maximum iterations R
  Output: Adversarial sample x_{act}
  Initial action list A = set(X);
  $X_{adv} = X$ # Initialize Sample;
  $A = set(X)$ # Initialize Action set;
  while ¬*Success*(X_{adv}) ∧ *A!*=∅ **do**
    if *targeted* **then**
      c = random class except c_l;
    end
    else
      c = c_l;
    end
    X_{act} = MCTS(f_n,A,c,d,R);
    X_{adv}.update(X_{act},Homoglyph(X_{act}));
    A.remove(X_{act});
  end
  return X_{act};

Algorithm 1: MCTSBug algorithm

Connecting to related studies: Any method to generate adversarial text sequence needs a token transformer; that is, a mechanic to perturb text. On the well studied adversarial image samples, a perturbation is often following the gradient direction. However, such idea is problematic when directly being applied to text samples. One important issue is that gradient on the original text sample is hard to defined. Deep learning classifiers often accept the text sample as a sequence of id numbers to a dictionary, and gradients on those id numbers are not entirely meaningful. (Papernot et al., 2016c) suggested to create perturbation based on gradient of the word embedding layer. However,

generating such perturbation requires the assumption of continuity, that is, words near each other shares some similarity. However, two words close to each other in the embedding layers are often neither similar on shape nor on meaning (Figure 5). Besides, generating such perturbation requires a projection from a perturbed embedding vector to the discrete space. Hard to control the distance of such perturbations. (Samanta & Mehta, 2017) used linguistic based rules to transform words. While linguistic rules might be able to successfully generate appropriate perturbations in some cases, their method is complicated and time consuming. More importantly, the generated samples cannot provide any guarantee on the size of the perturbation.

2.2 FINDING IMPORTANT WORDS THOUGH MONTE CARLO TREE SEARCH

Homoglyph attack can successfully neutralize a word for a deep-learning classifier. To perform the minimal edit operations for generating an adversarial sequence from a seed $\mathbf{x}$, we still need a powerful way to find out those important words (with respect to their influence on a deep classifier's prediction). Monte Carlo Tree Search is a method that creates a search tree to find optimal results in some decision space, which combines Monte Carlo Method and Bandit. The Upper Confidence Tree (UCT) framework (Kocsis & Szepesvári, 2006) includes Upper Confidence Bound algorithm (UCB)(Auer et al., 2002) to balance the exploration and exploitation in the tree search process, and achieve significant improvement in Go AI(Gelly & Silver, 2011). Later, many different applications choose MCTS as the solving algorithm, including game AI(Chaslot et al., 2008), single player game(Cazenave, 2009), and even feature selection(Gaudel & Sebag, 2010).

Monte Carlo Tree Search is known to be able to find near optimal solutions in the vast search space. It has shown great success recently in different scenarios. To make it work, it requires a value function which can be queried at any time, which in our case can be calculated in a black-box manner through querying the target deep learning neural network.

In details, we formalize adversarial perturbation generation problem as a reinforcement learning problem as follow. Let $\mathbb{X} = \{x_1, x_2..., x_n\}$ denote the set of all possible words to change. We define state s to be the current set of words will be sent to the homoglyph transformer, and action a to be the action of select a single word and add it to current set. The whole process terminates when the size of current state s reaches a pre-defined limitation of d (or there's no more items to add). Therefore, the state space $\mathbf{S}$ is the powerset of word set $\mathbb{X}$. The action space A is then equivalent to the set $\mathbb{X}$. In this setting, a determined transition function over state and actions is then defined: $p : S \times A \to S$, which is simply removing the word action a represents from current s. A policy $\pi(s)$ is a probability distribution over all possible action a.

In the MCTS algorithm creates a search tree, in which every node represents a state. Every node stores a value v and number of times visits n_c. Each edge represents an action a, that connects parent s and child s' if state s takes action a comes to state s'. The goal of the MCTS is to find a node(state) that minimize some loss function, which in our setting should be maximize a score which correctly evaluates the quality of adversarial samples. Formally, the goal of MCTS is to find the optimal terminal state s_T over a predefined score function f_s, that $s_T = \arg\max_{s \in T_s} f_s(s)$, T_s stands for the set of terminal states.

In our setting, the score function f_s have multiple choices, one example can be the cross-entropy loss to its label. However, this loss function doesn't typically reflect the terminal condition of non targeted scenario: A higher score doesn't necessarily indicate the sample is better. A successful adversarial sample may have a lower cross entropy than an unsuccessful adversarial sample, if it's score is more balanced on different terms.

Suppose f_s the We defined directly by the definition of untargeted attack:

$$f_s(s) = max_{c' \in C/c_l} \Pr(c'|s) - \Pr(c_l|s) \tag{2}$$

Here c_l is the original predicted class, which the class that adversary trying to deviate the sample from. The equation stands for the maximum predicted probability on any class except c_l minus the predicted probability of target class c_l.

We defined f_s for the targeted attack as:

$$f_s(s) = \Pr(c_{target}|s) \tag{3}$$

c_{target} is the class that adversary is trying to convert the sample into.

Such score is not fully differentiable, so that it can't be used in other optimization process, SGD for example. However, in MCTS the gradient is not required. That means unlike traditional optimization, MCTS is free to choose any function.

We use a modified version of the UCT-1 algorithm(Kocsis & Szepesvári, 2006). Generally the algorithm treats every node as a independent bandit problem. Every iteration of MCTS algorithm includes four steps: Selection, Expansion, Simulation and Backpropagation. In the selection phase, the algorithm start from the root node. And on each node (s, v_s, n_s) where s is a set of words, the algorithm selects the action following the Upper Confidence Bound (UCB) criterion: $s' = \arg\max_{s'} v_{s'} + 2C\sqrt{\frac{2\ln n_s}{2n_{s'}}}$, $C > 0$ is a hyperparameter constant. The search ends at a node s' doesn't belong to the tree, and it is then added to the tree in the following expansion phase, with $v_{s'}$ and $n_{s'}$ initialized as 0. After that, the algorithm starts a simulation phase that that selects random actions from the node s', until a terminal state is reached. It will then calculate the score function to return a score for the terminal state. Finally, in the backpropagation update the value of all the tree nodes, from s' go up to all its parent nodes. The visit count n is increase by 1 and the value is updated to the average of all the scores in the trails.

The total algorithm is described in Algorithm 1. To summarize, MCTSBug use MCTS to search for the most appropriate set of d words to change. After it found that set of words, it then modifies the words by using homoglyphs. One character in one word can neutralize the set of words. The effect to the prediction is correctly simulated in the MCTS value function. If the adversarial sample generated is not enough, the algorithm increase the value d. The algorithm will stop either the adversarial samples is a success or the maximum search limit is reached.

Connecting to related studies: One previously proposed way to select words used the saliency over those words on the word embedding (Samanta & Mehta, 2017). However, that breaks the black-box assumption. Besides, saliency can't accurately reflect the effect of the perturbation. A more accurate way is to use simulated leave-one-out score to sort all the words, such as in (Gao et al., 2018). While the method is shown to effective to text input, it can be sub-optimal as it is essentially a greedy selection based on single round result.

Instead, we model the important words' selection problem as a combinatorial optimization problem and use Monte Carlo Tree Search to solve it. Comparing to the leave-one-out based greedy strategy, the greedy approach only focuses on local optimum, and MCTS focus on the global optimum. MCTS instead search more space, and can provide a good estimation of the global maximum.

3 EXPERIMENT

We evaluated the effectiveness of our algorithm by conducting experiments on multiple deep learning models across multiple real-world text classification datasets. In particular, we wanted to answer the following questions empirically:

- Do adversarial samples generated by MCTSBug look valid?
- Does the accuracy of deep classifier decrease by the crafted adversarial samples?
- Can the adversarial samples generated by MCTSBug transfer between models?
- Are MCTSBug strategies robust to configuration parameters?

To evaluate the quality of adversarial samples generated by MCTSBug , we implemented three other baselines and compared them to MCTSBug . For fair comparisons, all three baselines also use the homoglyph word transformer. In summary we compare the following four methods:

1. **Random (baseline):** Random select words for the perturbation.
2. **Gradient (baseline):** Contrary to random selection which uses no knowledge of the model, we also compare to full knowledge of the model, where gradients are used to find the most important tokens. Following equation 3, the gradient method uses the magnitude of gradient w.r.t the original classification to decide which tokens should be changed. This method of selecting tokens is proposed in (Samanta & Mehta, 2017).

Data		Message	Prediction
AG's News Dataset	Original Message	viruses keep on growing most it managers won 39 t question the importance of security but this priority has been sliding between the third and fourth most important focus for companies	Sci/Tech
	Projected Gradient Descent	viruses keep on growing print it troy glance 39 t leftwich the guy dollars changed contributors this priority has been carriers between the debian and fourth most important focus for companies	Sports
	Greedy	viruses keep on growing most it managers won 39 s quertion the importance of securits but this priority has been sliding between the third and fourth most important focus for companies	Business
	MCTSBug	viruses keep on growing most it managers won 39 t question the importance of security but this priority has been sliding between the third and fourth most important focus for companies	Business
Amazon Reviews Full Dataset	Original Message	dave matthews band even in the slightest add this to your collection there is one song angle where his back up singers take over a bit and it is so incredible that one song is worth buying this dvd for but trust me you wont be disappointed in the rest	5 Stars
	Projected Gradient Descent	dave matthews band even in the slightest add this to your collection there is one song angle where his back up singers take over a bit and it inventions mfc terrific that one song is worth buying this dvd sensitivity hathaway clicks me hone transaction be studio in the olsen	5 Stars
	Greedy	dave matthews band even in the slightest add this to your collection there is one song angle where his back up singers take over a bit and it is so incredible that one song is worth buying this dvd for but trust me you eont be disappointed in the rest	3 Stars
	MCTSBug	dave matthews band even in the slightest add this to your collection there is one song angle where his back up singers take over a bit and it is so incredible that one song is worth buying this dvd for but trust me you wont be disappointed in the rest	3 Stars
DBPedia Dataset	Original Message	...20 and 100 m they can be found worldwide in tropical and subtropical waters they are predators that wait for prey fish to pass by they have which they move to attract the prey they have little economic value other than a minor role in the aquarium trade	Animal
	Projected Gradient Descent	...goran and stretched cbd nationale luc jesus oakland stacy in sunflower and spice waters they are predators that wait for prey fish to pass by they have which they move to attract the prey they have little economic value other than a minor role in the aquarium trade	Building
	Greedy	...20 and 100 m they can be fcund worldwide in tropical and subtropical waters they ane aredators that wait for prey fivh to pass by they have which they move to attract the prey they have little economic value other than a minor role in the aquarium trade	Building
	MCTSBug	...20 and 100 m they can be found worldwide in tropical and subtropical waters they are predators that wait for prey fish to pass by they have which they move to attract the prey they have little economic value other than a minor role in the aquarium trade	Building

Table 1: Examples of generated adversarial samples: The red part indicates the difference to the original message.

3. **Greedy (baseline):** Following DeepWordbug paper(Gao et al., 2018), the greedy baseline first evaluate the importance of each word by calculating a leave-one-out score on every word, and perform a greedy selection on the words.
4. **MCTSBug (our method):** Use MCTS to find the most important tokens (Section 2).

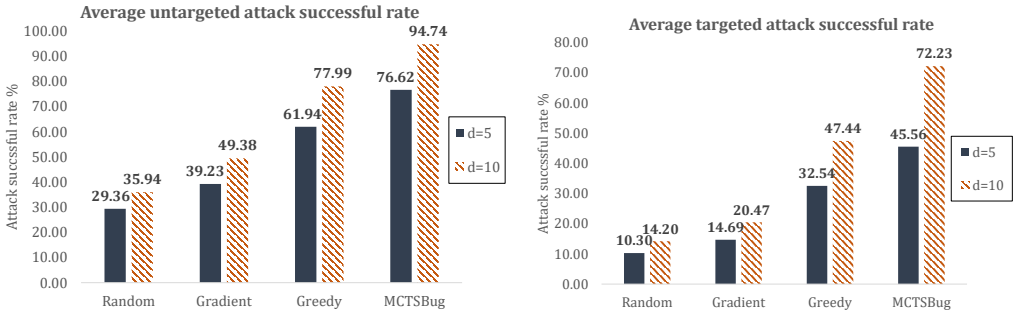
Due to space limit, we have put the details of datasets, experimental setups, and details of target deep models in the appendix. We also put two detailed figures (Figure 8) and Figure 7) showing how various methods behave by varying the number of characters they can modify and two tables (Table 7 and Table 6) showing the effective rates under targeted and untargeted attacks in the appendix. Besides, the empirical results of transferbility are shown in Figure 10. Figure 12 shows the effective curves for different output classes and Figure 9 shows the sensitivity analysis of how MCTSBug behaves when varying the value of C. Lastly we tried to conduct adversarial training by retraining the deep model using adverarial sequences generated by MCTSBug on the AG's News datase . However Figure 11 shows that this retraining strategy does not effectively defend against our attack.

3.1 A DISPLAY TABLE OF SELECTED ADVERSARIAL SEQUENCES:

To evaluate the quality of adversarial samples, we first compare our generated samples to state-of-the-art baselines. The result is summarized in 1. From Table 1, we can see that our MCTSBug creates samples with minimal perturbation. Comparing to baselines that changes many words or characters, adversarial samples from MCTSBug are extremely hard to be found to human eyes.

3.2 RESULTS ABOUT EFFECTIVENESS OF GENERATED ADVERSARIAL SAMPLES

We first present the result of untargeted attack. Here, we calculate attacking successful rates on all 7 datasets when gradually relaxing the limitation on maximum edit distance difference. Figure 7



(a) (b)

Figure 4: Summary of Comparisons showing the average successful rates of generated adversarial samples from each method averaged over all seven benchmark datasets about text classification. (a) represent the results from untargeted attack setup. (b) represent the results from the targeted setup.

shows a direct comparison of the effectiveness of our attack to the baseline methods: With the attack that flips at most 20 words, MCTSBug have 100% or nearly 100% attacking successful rate on all 7 datasets, which means MCTSBug can successfully flip the prediction of any samples, which clearly outperforms all the baselines in comparison. We also notice that for most datasets, manipulating no more than 5 words can successfully flip the prediction of 70% to 90% samples.

Comparing to the baselines, the Random baseline not surprisingly doesn't perform very well, which indicates that even with a powerful transformation function like homoglyph attack, generating an adversarial text input is not a trivial task. The Gradient baseline (Green line) also doesn't perform very well.

The basic reason for the bad performance of gradient based white-box method is it doesn't fit our task. Gradient, as a estimator of function value change by perturbation, is only accurate when the perturbation is small enough, and has the same perturbation over every input features. In our case, our modification on the target is not small and also not same perturbation over different input features (as every word vector are different). In this case, gradient is not a good estimation of the effect of our transformation.

We then present the result of targeted attack. One may assume as targeted attack are much harder than untargeted attack, the power of MCTSBug will be severely undermined when comes to the targeted scenario. To justify it, we similarly present the targeted attacking successful rates on all 7 datasets when relaxing the limitation on maximum edit distance difference. Figure 8 shows a direct comparison of the effectiveness of our attack to the baseline methods: MCTSBug have the attacking successful rate 70%-90% on the Basic-LSTM model.

4 Conclusion

Due to the combinatorial nature of a large and discrete input space, searching adversarial text sequences is not a straightforward extension from image-based techniques generating adversarial examples. In this paper, we present a novel framework, MCTSBug which can generate imperceptible adversarial text sequences in a black-box manner. By combining homoglyph attack and MCTS, the proposed method can successfully fool two deep RNN-based text classifiers across seven large-scale benchmarks.

The key idea we utilized in MCTSBug is that words with one character changed to its homoglyph pair are usually viewed as "*unknown*" by the deep-learning models. The shape change is almost invisible to humans, however, it is a huge change to a deep-learning model. More fundamentally this is caused by the fact that NLP training datasets normally only cover a very small portion of the huge space built by the combination of possible NLP letters. Then the question is: should the deep-learning classifiers model those words having identical shapes to human eyes as similar representations? We think the answer should be Yes.

REFERENCES

Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.

Marco Barreno, Blaine Nelson, Russell Sears, Anthony D. Joseph, and J. Doug Tygar. Can machine learning be secure? In *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, pp. 16–25. ACM, 2006. URL http://dl.acm.org/citation.cfm?id=1128824.

Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *Security and Privacy (SP), 2017 IEEE Symposium on*, pp. 39–57. IEEE, 2017.

Nicholas Carlini and David A. Wagner. Audio adversarial examples: Targeted attacks on speech-to-text. *CoRR*, abs/1801.01944, 2018. URL http://arxiv.org/abs/1801.01944.

Tristan Cazenave. Nested monte-carlo search. In *IJCAI*, volume 9, pp. 456–461, 2009.

Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. Monte-carlo tree search: A new framework for game ai. In *AIIDE*, 2008.

Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pp. 15–26. ACM, 2017.

Minhao Cheng, Jinfeng Yi, Huan Zhang, Pin-Yu Chen, and Cho-Jui Hsieh. Seq2sick: Evaluating the robustness of sequence-to-sequence models with adversarial examples. *arXiv preprint arXiv:1803.01128*, 2018.

Moustapha M Cisse, Yossi Adi, Natalia Neverova, and Joseph Keshet. Houdini: Fooling deep structured visual and speech recognition models with adversarial examples. In *Advances in Neural Information Processing Systems*, pp. 6977–6987, 2017.

Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pp. 160–167. ACM, 2008.

Nilesh Dalvi, Pedro Domingos, Sumit Sanghai, Deepak Verma, et al. Adversarial classification. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 99–108. ACM, 2004.

Evgeniy Gabrilovich and Alex Gontmakher. The homograph attack. *Communications of the ACM*, 45(2):128, 2002.

Ji Gao, Jack Lanchantin, Mary Lou Soffa, and Yanjun Qi. Black-box generation of adversarial text sequences to evade deep learning classifiers. In *IEEE Security and Privacy Workshops (SPW)*, 2018.

Romaric Gaudel and Michele Sebag. Feature selection as a one-player game. In *International Conference on Machine Learning*, pp. 359–366, 2010.

Sylvain Gelly and David Silver. Monte-carlo tree search and rapid action value estimation in computer go. *Artificial Intelligence*, 175(11):1856–1875, 2011.

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

Andrew Ilyas, Logan Engstrom, Anish Athalye, and Jessy Lin. Black-box adversarial attacks with limited queries and information. *arXiv preprint arXiv:1804.08598*, 2018a.

Andrew Ilyas, Logan Engstrom, and Aleksander Madry. Prior convictions: Black-box adversarial attacks with bandits and priors. *arXiv preprint arXiv:1807.07978*, 2018b.

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

Google Jigsaw. Perspective api. `https://www.perspectiveapi.com/`, 2017.

Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pp. 282–293. Springer, 2006.

Viktor Krammer. Phishing defense against idn address spoofing attacks. In *Proceedings of the 2006 International Conference on Privacy, Security and Trust: Bridge the Gap Between PST Technologies and Business Services*, pp. 32. ACM, 2006.

Bin Liang, Hongcheng Li, Miaoqiang Su, Pan Bian, Xirong Li, and Wenchang Shi. Deep text classification can be fooled. *arXiv preprint arXiv:1704.08006*, 2017.

Daniel Lowd and Christopher Meek. Adversarial learning. In *Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, pp. 641–647. ACM, 2005a.

Daniel Lowd and Christopher Meek. Good word attacks on statistical spam filters. In *CEAS*, volume 2005, 2005b.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

Makoto Miwa and Mohit Bansal. End-to-end relation extraction using lstms on sequences and tree structures. *arXiv preprint arXiv:1601.00770*, 2016.

Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *CVPR*. IEEE, 2015.

Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against deep learning systems using adversarial examples. *arXiv preprint arXiv:1602.02697*, 2016a.

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

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

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

Suranjana Samanta and Sameep Mehta. Towards crafting text adversarial samples. *arXiv preprint arXiv:1707.02812*, 2017.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations (ICLR)*, 2014.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

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

5 MORE ABOUT RELATED WORKS

5.1 RELATED EFFORTS ABOUT ADVERSARIAL SEQUENCES FOR TEXT CLASSIFICATIONS

Early research on attacking text classifiers starts from (Dalvi et al., 2004) and (Lowd & Meek, 2005a), when researchers show that malicious manipulation on input can cause machine learning based spam detectors to generate false positives. (Lowd & Meek, 2005b) propose the Good Word attack, which is a practical attack that adds positive, non-spam words into a spam message to evade machine learning classifiers. These attacks are designed for simple classifiers such as Naive Bayes, which work directly on different features. However, modern deep learning models work differently from traditional machine learning models. Also, such methods give no guarantee of the quality of the generated sample. For example, a spam message could become non-spam if too many "good words" were added to it, but the output message is almost guaranteed to be meaningless.

In 2013, (Szegedy et al., 2014) proposes the concept of adversarial sample: imperceptible perturbation on images can fool deep learning classifiers. This discovery is interesting because small modifications guarantee the validity of generated samples. Compared to studies of adversarial examples on images, little attention has been paid on generating adversarial sequences on text. Papernot et al. applied gradient-based adversarial modifications directly to NLP inputs targeting RNN-based classifiers in (Papernot et al., 2016c). 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.

Recently, (Samanta & Mehta, 2017) used the embedding gradient to determine important words. The technique used heuristic driven rules together with hand-crafted synonyms and typos. The method is a white-box attack since it accesses the gradient of the model. Another paper (Liang et al., 2017) measures the importance of each word to a certain class by using the word frequency from that class's training data. Then the method uses heuristic driven techniques to generate adversarial samples by adding, modifying, or removing important words. This method needs to access a large set of labeled data.

All after-mentioned methods are under the typical white-box setting of adversarial generation scenarios, in which gradients are used to guide the modification of input tokens from an original sample to an adversarial sample. However, gradients are hard to define on discrete symbolic text inputs. Also in black-box settings, calculating gradients is not possible since the model parameters are not observable. Recently, (Gao et al., 2018) proposed greedy scoring strategies to rank words in a text sequence and then applied simple character-level transformations like swapping to fool deep classifiers. Its central idea, minimizing the edit distance of the perturbation makes sense. However the perturbations are quite visible and the empirical effectiveness needs improvements. (Gao et al., 2018) propose multiple scoring strategies. However, it lacks the comparison between those scoring strategies.

Our method does not require knowing the structure, parameters or gradient of the target model, while previous methods do. Also, most previous approaches used heuristic-driven and complicated modification approaches. We summarize the differences between our method and previous methods on generating adversarial text samples in Table 2.

Besides, a few recent studies tried to generate adversarial examples for other NLPS tasks, like seq2seq based machine translation or machine-learning based reading comprehension NLP tasks. For example, (Cheng et al., 2018) attacks seq2seq translation models. Use projected gradient based input, (Cheng et al., 2018) can make the output of the translation model be completely different.(Jia & Liang, 2017) generate samples that attacks Question Answering models. The authors propose algorithms which imitate the pattern of the question and feed additional wrong information into the sample that mislead the model to give a wrong answer.

	Adversary	Distance	Space	Word Selection Method	Modifications
MCTSBug	Black-box	Edit (L_0)	Input space	MCTS	Homoglyph attack
(Gao et al., 2018)	Black-box	Edit (L_0)	Input space	Greedy based on sensitivity scores	Swapping characters
(Papernot et al., 2016c)	White-box	L_∞	Embedding space	Random	Gradient + Projection
(Samanta & Mehta, 2017)	White-box	Num. words modified (L_0)	Input space	Greedy using the magnitude of gradient	Complicated Linguistic-driven

Table 2: Summary of Relevant methods

5.2 Background: Word Embedding and Nearest Neighbor Words based on Such Embedding

Word	1-NN result	Word	1-NN result
thick	dense	hate	hatred
cat	dog	hated	loved
the	part	cold	warm
on	the	ice	melting
prime	minister	cool	hot
shock	sudden	gallon	gasoline
album	song	prototype	modified

Figure 5: Examples: Nearest neighbor of the words on the GloVe-100d embedding space.

Machine learning models, when processing text, must first discretize a text into tokens. These tokens can then be fed into the model. Words are often used as the smallest unit for input to a model. A word embedding is a mapping that projects every word in a dictionary to a unique vector in some vector space. Such mappings transform the discrete representations of words into features on a continuous space, which is more conducive to use with machine learning models.

Word embeddings have shown success in many NLP tasks (Collobert & Weston, 2008; Pennington et al., 2014; Mikolov et al., 2013). Such models hold a dictionary and build the word embedding based on the dictionary. To be able to work with words not in the dictionary, word embedding based models often add a special entry in the dictionary called "out of vocabulary," or "unknown". Those models work well if the word embedding is well constructed and covers most words that occur in the data.

5.3 More about Homoglyph attack

Homoglyph has been a notorious security problem to many computer systems for many years. Hackers and malicious attackers has been deliberately using homoglyphs as attacks for many years. According to (Gabrilovich & Gontmakher, 2002), back to 2000 some person create a website 'bl00mberg.com'(with 0 instead of o) to release fake financial news and fool many investors. Such issue become more severe as later domain name can include Unicode characters(punycode). Early studies of homoglyph attack includes (Gabrilovich & Gontmakher, 2002) and (Krammer, 2006).

One may suggest since this problem has been popular for many years, it will certainly be solved by modern designs of web systems. However, in 2017 this issue still happens in popular web browsers Chrome and Firefox. Similar name spoofing attack are also used in computer viruses, where they make the name of virus process looks to be very similar to one of the system process (for example, svchost.exe and svch0st.exe).

5.4 Background: Adversarial Examples for Image Classification and More:

There exist a large body of recent studies focusing on adversarial examples for image classification that typically created imperceptible modifications to pixel values through an optimization procedure (Goodfellow et al., 2014; Szegedy et al., 2014; Papernot et al., 2016b; Carlini & Wagner, 2017). Szegedy et al. (Szegedy et al., 2014) first observed that DNN models are vulnerable to ad-

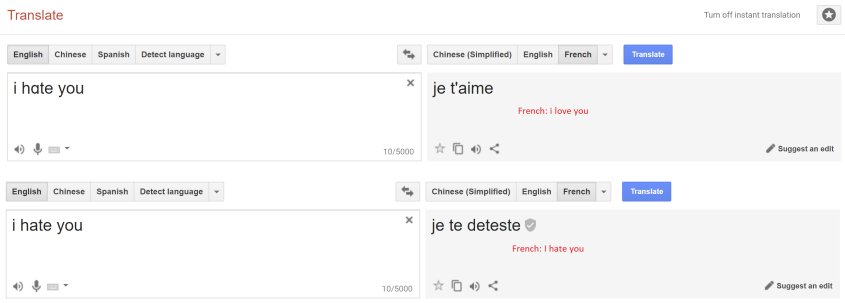


Figure 6: Homoglyph attack on Google Translation: A subtle difference can totally flip the translation result.

versarial 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 using the linear assumption behind a model (Goodfellow et al., 2014) (by limits on L_∞ norm), saliency maps (Papernot et al., 2016b) (by limits on L_0 norm), and evolutionary algorithms (Nguyen et al., 2015). Recently, Carlini et al. proposed a group of attacking methods with optimization techniques to generate adversarial images with even smaller perturbations (Carlini & Wagner, 2017). (Papernot et al., 2016a) shows that it's possible to generate adversarial samples that can successfully reduce the classification accuracy in the black-box manner without the knowledge of model structure or the model parameters in the neural network. Recently multiple studies have extended adversarial examples to domains similar as images such as like speech recognition (Cisse et al., 2017) or speech-to-text (Carlini & Wagner, 2018).

6 More About Experiments

6.1 Experimental Setup Details

Our experimental set up is as follows:

- **Datasets:** In our experiments, we use seven large-scale text datasets gathered from (Zhang et al., 2015), which includes a variety of NLP tasks, e.g., text classification, sentiment analysis and spam detection. Details of the datasets are listed in Table 4.
- **Target Models:**

  We conduct our experiment on several state-of-the-art RNN models.

  Basic-LSTM: A uni-directional LSTM. Detailedly speaking, the network contains a random embedding layer with dimension 100 to accept the word inputs. The embedding vectors are then fed through two LSTM layers (one for each direction) with 100 hidden nodes each. The hidden states of each direction are concatenated at the output and fed to a fully connected layer for the classification.

  Basic-LSTM with GloVe Embedding: Similar to Basic-LSTM, but use pretrained GloVe embedding (Pennington et al., 2014) from Stanford as the embedding layer.

  Bi-LSTM: A bi-directional LSTM, which contains an LSTM in both directions (reading from first word to last and from last word to first). The network contains a random embedding layer to accept the word inputs. The embedding vectors are then fed through two LSTM layers (one for each direction) with 100 hidden nodes each. The hidden states of each direction are concatenated at the output and fed to a fully connected layer for the classification.

  Bi-LSTM with GloVe Embedding: Similar to Bi-LSTM, but use pretrained GloVe embedding (Pennington et al., 2014) from Stanford as the embedding layer.

Dataset \Model	LSTM	BiLSTM
AG's News	88.45	90.49
Amazon Review Full	61.96	61.97
Amazon Review Polarity	95.43	95.46
DBPedia	98.65	98.65
Yahoo! Answers	73.42	73.40
Yelp Review Full	64.86	64.69
Yelp Review Polarity	95.87	95.92

Table 3: Models' accuracy in the non-adversarial setting

	#Training	#Testing	#Classes	Task
AG's News	120,000	7,600	4	News Categorization
Amazon Review Full	3,000,000	650,000	5	Sentiment Analysis
Amazon Review Polarity	3,600,000	400,000	2	Sentiment Analysis
DBPedia	560,000	70,000	14	Ontology Classification
Yahoo! Answers	1,400,000	60,000	10	Topic Classification
Yelp Review Full	650,000	50,000	5	Sentiment Analysis
Yelp Review Polarity	560,000	38,000	2	Sentiment Analysis

Table 4: Dataset details

We summarize the performance of the models without adversarial samples in Table 3. In the non-adversarial setting, these models has achieve state-of-the-art level performance on those datasets.

- **Platform:** We train the target deep-learning models and implement attacking methods using software platform PyTorch 0.3.0. All the experiments run on a server machine, whose operating system is CentOS7 and have 4 Titan X GPU cards.
- **Evaluation:** Performance of the attacking methods is measured by the successful rate of the attack, which is 1 - the model on generated adversarial sequences. The lower the accuracy, the more effective is the attacking method.

	Character View		Word View	
Dataset	Mean	Median	Mean	Median
AG's News	602.96	596	40.6	40
Amazon Review Full	657.87	626	82.61	75
Amazon Review Polarity	657.45	628	82.57	74
DBPedia	639.39	623	54.48	56
Yahoo! Answers	623.44	623	85.2	57
Yelp Review Full	694.64	649	130.8	98
Yelp Review Polarity	697.03	649	129.53	101

Table 5: Dataset Sample Length Statistics. Mean and median words (or characters) per sample in each dataset.

6.2 More Results about Comparing Methods

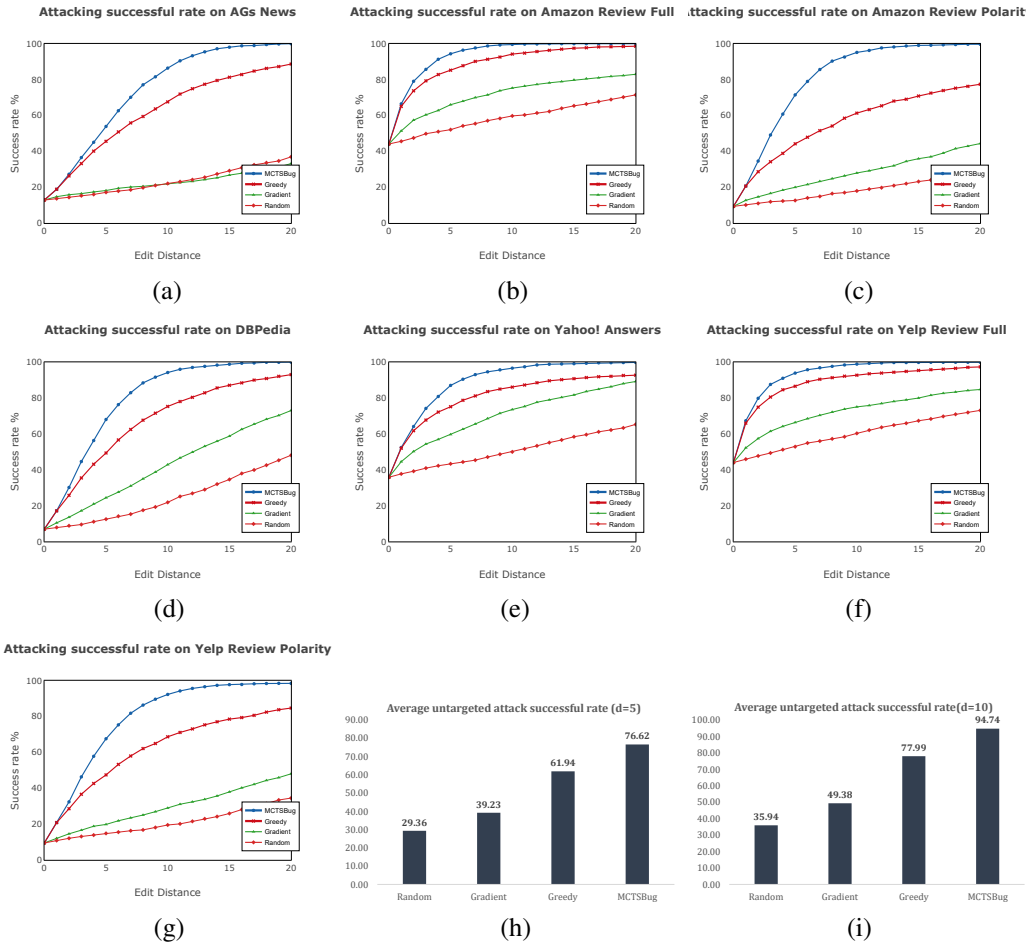


Figure 7: Experiment results of untargeted attack. (a) - (h) shows the attack successful rate on generated adversarial samples, each figure represents the result on a certain dataset. The X axis represents the number of modified characters(d), and the Y axis corresponds to the attack successful rate of the adversarial samples generated using the respective attacking methods.

	Random	Gradient	Greedy	**MCTS**
AGs News	22.11	21.80	67.58	**86.41**
Amazon Review Full	59.77	75.31	94.22	**99.61**
Amazon Review Polarity	17.97	27.97	61.33	**95.16**
DBPedia	21.88	42.89	75.31	**94.22**
Yahoo! Answers	50.00	73.59	86.09	**96.64**
Yelp Review Full	60.31	75.08	92.73	**98.91**
Yelp Review Polarity	19.53	28.98	68.67	**92.27**
Average	35.94	49.38	77.99	**94.74**

Table 6: Untargeted attack successful rate (d=10)

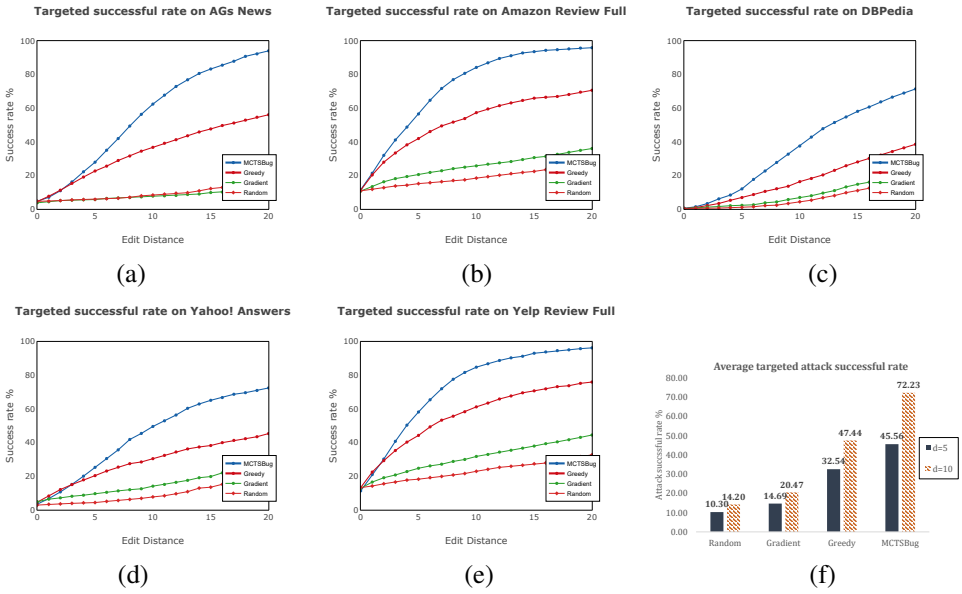


Figure 8: Experiment results of targeted attack. (a) - (e) shows the attack successful rate on generated adversarial samples, each figure represents the result on a certain dataset (We only present those dataset with larger than 2 classes). The X axis represents the number of modified characters(d), and the Y axis corresponds to the attack successful rate of the adversarial samples generated using the respective attacking methods.

	Random	Gradient	Greedy	**MCTS**
AGs News	8.36	7.73	36.72	**62.42**
Amazon Review Full	18.44	25.70	57.34	**84.22**
Amazon Review Polarity	17.97	27.97	61.33	**94.84**
DBPedia	4.38	6.95	16.33	**37.50**
Yahoo! Answers	7.73	14.14	30.47	**49.61**
Yelp Review Full	22.97	31.80	61.25	**84.84**
Yelp Review Polarity	19.53	28.98	68.67	**92.19**
Average	14.20	20.47	47.44	**72.23**

Table 7: Targeted attack successful rate (d=10)

6.3 Sensitivity Analysis: Varying the Value of MCTS Parameter C

We study the effect of different parameters in this section. Our goal is to show our method i s robust to the change of parameters.

In the MCTSBug algorithm, c is a parameter of UCB that balances the exploration and exploitation. We test several different choices of c value on AG's News Dataset. The result shows that MCTSBug algorithm is robust to the change of C.

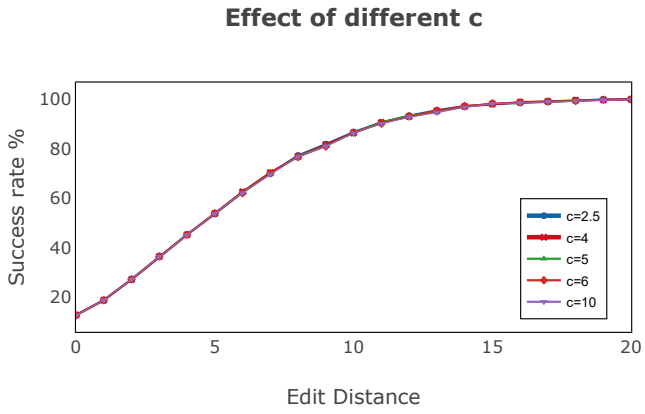


Figure 9: MCTSBug successful rate for different values of C on AG's News Dataset.

6.4 TRANSFERABILITY

Adversarial samples are famous for their transferability, which means a lot of adversarial samples that are generated for a certain model can also successfully fool other DNN models.

We tested the transferability of adversarial samples generated using MCTSBug .

We evaluate the transferability using four LSTMs. In the following table, LSTM1/BiLSTM1 are trained with randomly-initialized embedding, and LSTM2/BiLSTM2 are trained with GLoVe word embeddings.

Figure 10 shows the accuracy results from feeding adversarial sequences generated by one model to another model on the same task. The results show that the target model accuracy in these circumstances is reduced from around 90% to 30-60%. Most adversarial samples can be successfully transferred to other models, even to those models with different word embedding. This experiment demonstrates that our method can successfully find those words that are important for classification and that the transformation is effective across multiple models.

6.5 POSSIBLE DEFENSE: ADVERSARIAL TRAINING VIA MCTSBUG SEQUENCES

From 11, adversarial training only slightly increase the capacity of adversarial samples

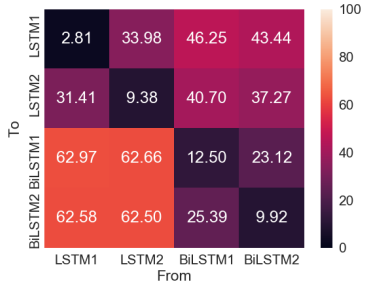


Figure 10: Heatmap that shows the transferability of MCTSBug : Numbers represents the accuracy of the target model tested on the adversarial samples of AG's News Dataset.

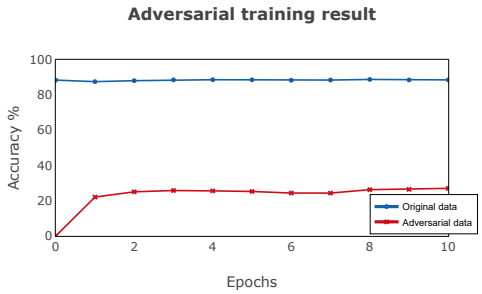


Figure 11: Adversarial training on AG's News Dataset

6.6 EFFECT ON DIFFERENT CLASS OF SAMPLES

In another experiment, we study the performance of DeepWordBug attack on different classes of samples. We present the results in the Figure 12. From the figure, we can see that our attack works differently on different classes of inputs. It reveals that there may exist some bias in the target deep learning model towards some certain class.

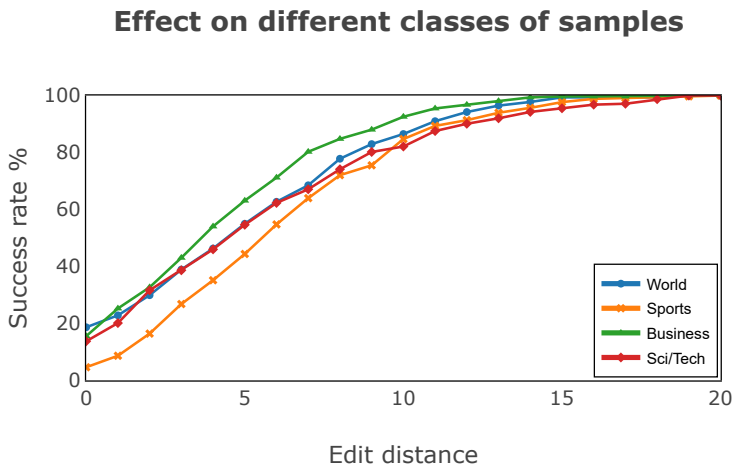


Figure 12: The effectiveness of MCTSBug over different classes of inputs on Ag's New's Dataset

6.7 CLASSIFICATION CERTAINTY ON ADVERSARIAL SAMPLES

We present a histogram to analyze the distribution of the output probability of the target machine learning model on the fooled class in Figure 13. The result shows that even though we increase the word modification one by one, most adversarial samples we generated are predicted wrongly by the model with a high certainty, which indicated that the generated adversarial samples are hard to be filtered directly using the probability distribution.

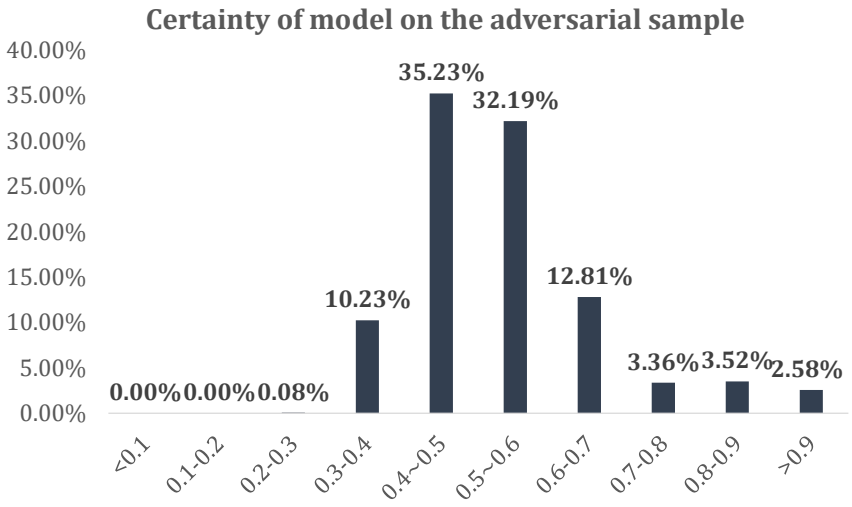


Figure 13: The certainty of generated samples by MCTSBug over different classes of inputs on Ag's New's Dataset

6.8 MCTSBUG IS APPLICABLE TO IMAGES: RESULTS ON EVADING MNIST DNN MODEL

The MCTS attack algorithm can be also applied to other discrete type inputs, with a properly defined value function and a properly defined transformation function. As an example, we show the result to generate on the MNIST dataset in Figure 14.

The idea is to generate adversarial samples on L0 norm. Because the uniqueness of the task, we can define a discrete action set over the pixels. Currently, for every pixel we define two types of actions: Change the value to 255 or change the value to 0. Doing so, the question can be viewed as a combinitorial optimization problem once again. We solve this problem: By performing at most 20 actions, we successfully flip 37 samples out of 100 samples we tested. We show some samples in Figure 14.

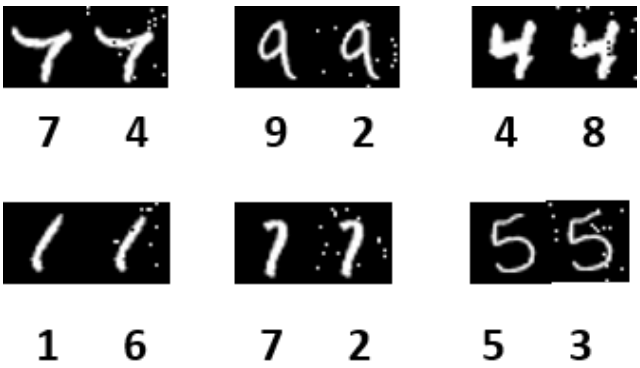


Figure 14: MCTSBug Generated adversarial samples on MNIST dataset. For each pair, left side is the original sample, right side is the corresponding adversarial sample