LeapAttack: Hard-Label Adversarial Attack on Text via Gradient-Based Optimization

Muchao Ye
The Pennsylvania State University
University Park, Pennsylvania, USA
muchao@psu.edu

Jinghui Chen
The Pennsylvania State University
University Park, Pennsylvania, USA
jzc5917@psu.edu

Chenglin Miao
University of Georgia
Athens, Georgia, USA
cmiao@uga.edu

Ting Wang
The Pennsylvania State University
University Park, Pennsylvania, USA
ting@psu.edu

Fenglong Ma*
The Pennsylvania State University
University Park, Pennsylvania, USA
fenglong@psu.edu

ABSTRACT

Generating text adversarial examples in the hard-label setting is a more realistic and challenging black-box adversarial attack problem, whose challenge comes from the fact that gradient cannot be directly calculated from discrete word replacements. Consequently, the effectiveness of gradient-based methods for this problem still awaits improvement. In this paper, we propose a gradient-based optimization method named LeapAttack to craft high-quality text adversarial examples in the hard-label setting. To specify, LeapAttack employs the word embedding space to characterize the semantic deviation between the two words of each perturbed substitution by their difference vector. Facilitated by this expression, LeapAttack gradually updates the perturbation direction and constructs adversarial examples in an iterative round trip: firstly, the gradient is estimated by transforming randomly sampled word candidates to continuous difference vectors after moving the current adversarial example near the decision boundary; secondly, the estimated gradient is mapped back to a new substitution word based on the cosine similarity metric. Extensive experimental results show that in the general case LeapAttack can efficiently generate high-quality text adversarial examples with the highest semantic similarity and the lowest perturbation rate in the hard-label setting.[1]

CCS CONCEPTS

• **Security and privacy**; • **Computing methodologies** → **Discrete space search**;

KEYWORDS

hard-label text adversarial attack; gradient-based optimization

* indicates corresponding author.

[1] Code is available at https://github.com/machinelearning4health/LeapAttack.

KDD '22, August 14–18, 2022, Washington, DC, USA

ACM ISBN 978-1-4503-9385-0/22/08…$15.00
https://doi.org/10.1145/3534678.3539357

ACM Reference Format:
Muchao Ye, Jinghui Chen, Chenglin Miao, Ting Wang, and Fenglong Ma. 2022. LeapAttack: Hard-Label Adversarial Attack on Text via Gradient-Based Optimization. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22), August 14–18, 2022, Washington, DC, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3534678.3539357

1 INTRODUCTION

Deep neural networks (DNNs) have performed outstandingly in the natural language processing (NLP) tasks including text classification [26] and natural language inference [1] recently. However, it is observed that DNNs can be easily fooled by adversarial examples [11], and thus their robustness against adversarial attacks is still in doubt despite their high accuracy. To reveal and improve the robustness of DNNs for NLP, researchers have focused on the *text adversarial attack* [19], which is the task of generating adversarial examples with high semantic similarity by adding small perturbation to original text samples to make DNNs change their correct predictions into incorrect ones.

Existing studies on text adversarial attack can be classified into two categories, i.e, *white-box* ones [9] and *black-box* ones [16, 18, 23]. The latter scenario is more challenging because attackers have no access to model parameters. Early black-box methods work in the *soft-label* setting, which allows attackers to access the prediction probability distribution associated with each input. Recently, a more realistic and difficult **hard-label black-box** setting raises more attention, where *attackers only know the top-1 prediction label* instead of distribution. Exisiting methods in the black-box setting usually adopt (1) score-based greedy algorithms, (2) heuristic algorithms such as the genetic algorithm (GA), or (3) gradient-based algorithms. However, current techniques still fall short in the hard-label setting. Firstly, the hard-label information is difficult for *scored-based greedy algorithms* [16, 23] to distinguish the importance and effect of different word positions and substitutions during adversarial attacks for they require soft-label prediction score distributions. Secondly, relying on past adversarial examples to generate new ones, *heuristic algorithms* [18] such as GA suffer from the threat of getting stuck in the local optimum and consuming numerous victim model queries due to the repetitive operations for all candidates and the fact that inappropriate substitutions of past ones can continually remain

in the new ones. Lastly, existing hard-label *gradient-based algorithms* [25] use virtual random direction to estimate the gradient, which has a potential problem that the sampled virtual directions can just be mapped to partial candidates and make them unable to fully use the embedding subspace expanded by all word candidates.

The aforementioned limitations of existing approaches stem from the **discrete nature** of text data because gradient cannot be directly calculated from discrete changes. In this paper, we aim to explore this challenging yet rewarding research question: *is it possible to further improve the effectiveness of gradient-based optimization approach to improve the generation quality of adversarial examples in the hard-label setting for discrete text data?* To answer this question, we propose a new method named **LeapAttack**. Compared to existing methods, the main improvement of LeapAttack is that it introduces an iterative round trip between discrete word candidates and continous gradients by employing the *word embedding space* as an auxiliary space as the bridge, allowing one adversarial candidate freely moving, i.e., *leaping*, in the embedding space.

As shown in Figure 1, the round trip starts from the random initialized adversarial candidate x'_0 attained in step (1). In each iteration t, LeapAttack firstly moves the current sample x'_{t-1} near the decision boundary and gets $\tilde{x}_{t-1}$ as shown in step (2). For the round trip shown in step (3), LeapAttack maps each discrete word w to a high-dimensional vector $\mathcal{H}(w)$ given the introduced word embedding space $\mathcal{H}$. Consequently, the semantic deviation brought by each substitution can be effectively characterized by the difference vectors between original and substitution words. Hence, it maps each sampled word candidate to a difference vector in $\mathcal{H}$ and selectively aggregates them into an estimated gradient based on the feedback from the victim model. After that, each gradient turns into a new replacement in x'_t via the cosine similarity measure, and the whole perturbation is represented by $\mathbf{P}_t$ as a group of difference vectors. The advantages of our method include that it does not rely on past candidates and *maintains only one candidate leaping* in the embedding space based on the gradient estimated from concrete word replacements, *avoiding the pitfall of inappropriate substitutions, repetitive operations and virtual randomly sampled directions.*

In short, our contributions are:

- LeapAttack relieves the inability of distinguishing semantic deviation and the dependence of existing methods on past adversarial examples in the hard-label setting by using the semantic information hidden in the word embedding space, which can improve optimization quality and efficiency.
- We improve the effectiveness of utilizing the gradient-based optimization in discrete text data by designing a novel mechanism that can interchange discrete substitutions and continuous vectors. It expresses discrete substitutions with difference vectors for gradient estimation and maps each gradient into a new discrete word based on the cosine similarity measure, fully utilizing the semantic embedding of all word candidates for the task.
- Experimental results on eight datasets demonstrate that LeapAttack can generate adversarial examples with the highest semantic similarity and lowest perturbation rate in most cases, alleviating previous limitations.

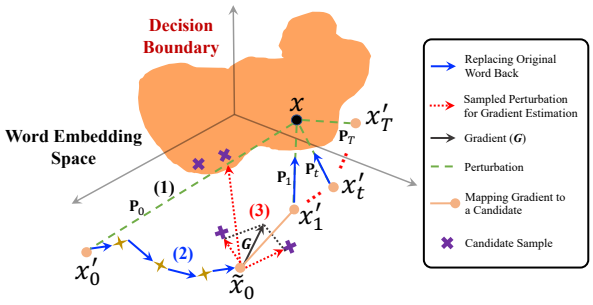


Figure 1: Overview of LeapAttack. After random initialization in step (1), in each iteration LeapAttack moves the current sample closer to the decision boundary in step (2) and interchanges discrete substitutions and continuous vectors for gradient estimation and word replacement in step (3).

2 METHODOLOGY

2.1 Hard-Label Text Adversarial Attack

In this paper, we focus on the hard-label black-box text adversarial attack, where attackers can only use the predicted label $f(x)$ from the victim model f to construct adversarial example for each input text sample x. Let $x = [w^{(1)}, w^{(2)}, \cdots, w^{(n)}]$, where $w^{(i)}$ is the i-th word, and n is the number of words. For each x, its associated ground truth label is y. In this task, attackers conduct attacks by using the samples that can be correctly classified by f, i.e., $f(x) = y$.

When constructing an adversarial example, each word $w^{(i)} \in x$ will be replaced by its synonym $s^{(i)}$ in the predefined synonym set $\mathrm{S}(w^{(i)})$, which includes $w^{(i)}$ itself. After several replacements, we say $x' = [s^{(1)}, s^{(2)}, \cdots, s^{(n)}]$ becomes an adversarial example if it misleads f toward a wrong prediction result, i.e., $f(x') \neq f(x)$.

We denote $\mathrm{Sim}(x, x')$ as an auxiliary function calculating the semantic similarity between x and any adversarial example x'. The ultimate goal of text adversarial attack is to generate an optimal adversarial example x^* that has the highest semantic similarity with the original sample x among all valid adversarial examples [19]. Thus, the output x^* of this task is defined as

$$x^* = \max_{x'} \mathrm{Sim}(x, x'), \; s.t. \; f(x') \neq f(x). \tag{1}$$

2.2 Framework of LeapAttack

Figure 1 shows the overview of the proposed LeapAttack, which first randomly initializes an adversarial example x'_0 for the original text x and then moves it closer to the decision boundary via conducting word replacement based on the gradient direction aggregated from all explored difference vectors.

2.2.1 Initialization. Due to the unavailability of soft-label scores, in the hard-label setting we need to use random initialization to ensure a feasible solution first before we are assured to keep improving the adversarial example [5–7, 18]. Since random initialization is generally used in the hard-label adversarial attack in various domains [5–7, 18] to attain an initial solution, for each input x we firstly find an initial adversarial candidate x'_0 such that $f(x'_0) \neq f(x)$ by conducting textual random initialization following [18].

If x_0' exists, we now introduce how we prepare a gradient-based optimization framework for this discrete problem. For each word $w^{(i)}$ in x, we introduce a pretrained word embedding space $\mathcal{H}$ (i.e., Counter-Fitted Word Vectors [21]) to obtain its word embedding

$$\mathbf{e}_i = \mathcal{H}(w^{(i)}) = [e_{(i,1)}, e_{(i,2)}, \cdots, e_{(i,m)}]^{\mathrm{T}}, \tag{2}$$

where $\mathbf{e}_i \in \mathbf{R}^m$ and m is the dimension of the word embedding space. As a result, we can get an embedding matrix $\mathbf{E} \in \mathbf{R}^{m\times n}$ for x and similarly $\mathbf{E}_t \in \mathbf{R}^{m\times n}$ for sample x_t' in $\mathcal{H}$ in iteration t. Based on $\mathbf{E}$ and $\mathbf{E}_t$, we express the perturbation at any iteration t as a perturbation matrix $\mathbf{P}_t$ that consists of n difference vectors

$$\mathbf{P}_t = \mathbf{E}_t - \mathbf{E} = [\mathbf{p}_1^{(t)}, \cdots, \mathbf{p}_n^{(t)}] \in \mathbf{R}^{m\times n}, \tag{3}$$

where $\mathbf{p}_i^{(t)}$ is the difference vector for the i-th word at iteration t. Such an expression allows us to represent the semantic deviation of different candidate words as difference vectors in $\mathcal{H}$. Most importantly, we are given a chance to represent different words by the continuous embedding values, and the search problem of finding good replacements can be guided by the gradient of the current word calculated in the embedding space $\mathcal{H}$.

2.2.2 Gradient-Based Optimization for Text. Now we introduce how we make gradient-based optimization feasible in text data to optimize the semantic similarity of generated adversarial examples, which is a two-step "round trip" in T iterations as shown in Figure 1, where T is the total iteration number. That is, estimating the gradient from the given sampled word candidates first, and mapping the continuous gradient to a substitute word afterwards. The process that we obtain the adversarial example x_t' ($1 \le t \le T$) in the t-th iteration given the output x_{t-1}' in previous iteration is as follows.

(1) **Moving Closer to the Decision Boundary.** We will approximate the gradient by the Monte Carlo estimate [20], which is valid when the text sample is at the boundary. Thus, to get an accurate estimation of the gradient, the first thing that we need to do is to move $x_{t-1}' = [w_{t-1}^{(1)}, w_{t-1}^{(2)}, \cdots, w_{t-1}^{(n)}]$ closer to the decision boundary. Due to the discrete nature of text data, the commonly-used binary search approach in image adversarial attack is not applicable here. Hence, we choose to *continually put original words back* to x_{t-1}' to move it closer to the decision boundary [18]:

- Replace each word $w_{t-1}^{(i)}$ in x_{t-1}' by the original word $w^{(i)}$. If the modified sample is still adversarial, we calculate the corresponding similarity score sim_i by the auxiliary function.
- Keep replacing the original words $w^{(i)}$ back to $w_{t-1}^{(i)}$ in the descending order of semantic similarity scores sim_i's and stop just before the new sample $\tilde{x}_{t-1}$ is not adversarial.

Through the process described above, we discover the original words that maintain important original semantics and still keep the sample adversarial. We then gradually replace these words back to the adversarial example x_{t-1}'. This process enables us to get a new adversarial example $\tilde{x}_{t-1}$ closer to the decision boundary for $\tilde{x}_{t-1}$ has more common words with x, which is beneficial to estimate a more accurate gradient.

(2) **Estimating the Gradient.** After we get a sample $\tilde{x}_{t-1} = [\tilde{w}_{t-1}^{(1)}, \cdots, \tilde{w}_{t-1}^{(n)}]$ that is closer to the decision boundary, the next step is to estimate the gradient direction for updating the perturbation based on the discrete substitution choices. By gradient, we mean the gradient with respect to the following function

$$Q(\tilde{x}_{t-1}) = f_{y'}(\tilde{x}_{t-1}) - f_y(\tilde{x}_{t-1}),\ s.t.\ y' \neq y, \tag{4}$$

where f classifies $\tilde{x}_{t-1}$ to another class y'. $f_{y'}(\tilde{x}_{t-1})$ and $f_y(\tilde{x}_{t-1})$ are the soft-label scores corresponding to the class y' and y given $\tilde{x}_{t-1}$. Maximizing Eq. (4) can keep $\tilde{x}_{t-1}$ outside of the decision boundary, and we want the next adversarial example to leap out of the decision boundary in the fastest direction, so we solve Eq. (4) by gradient descent. However, in the hard-label setting, we only know whether $Q(\tilde{x}_{t-1}) > 0$ or $Q(\tilde{x}_{t-1}) \le 0$. Thus, suppose the j-th word $\tilde{w}_{t-1}^{(j)} \neq w^{(j)}$, we estimate the gradient $\nabla Q(\tilde{x}_{t-1})$ at $\tilde{w}_{t-1}^{(j)}$ by mapping each sampled substitution into a difference vector in $\mathcal{H}$:

- Sample a candidate word $s_l^{(j)} \in \mathrm{S}(w^{(j)})$ in the synonym set and get a sample $x_{\text{tem}} = [\tilde{w}_{t-1}^{(1)}, \cdots, s_l^{(j)}, \cdots, \tilde{w}_{t-1}^{(n)}]$ whose only difference with $\tilde{x}_{t-1}$ is that it replaces $\tilde{w}_{t-1}^{(j)}$ by $s_l^{(j)}$. The deviation between the explored direction and existing direction is represented by the difference vector $\mathbf{d}_l^{(j)} = \mathcal{H}(s_l^{(j)}) - \mathcal{H}(\tilde{w}_{t-1}^{(j)}) \in \mathbf{R}^m$.
- Query the victim model with x_{tem} as the input. If x_{tem} is still adversarial, i.e., $Q(x_{\text{tem}}) > 0$, we denote this useful direction with a weight score corresponding to this replacement as $\mu_l = 1$, and otherwise, $\mu_l = -1$ for $Q(x_{\text{tem}}) \le 0$.
- Repeat the previous two steps k times. We calculate the average score of all μ_l as $\bar{\mu}$, and $(\mu_l - \bar{\mu})$ represents the weight for whether to adopt this deviation, and the gradient is

$$\mathbf{g}_j = \frac{1}{k-1}\sum^{l}(\mu_l - \bar{\mu}) \cdot \mathbf{d}_l^{(j)} \in \mathbf{R}^m. \tag{5}$$

The sum in Eq. (5) is divided by $k-1$ to get an unbiased estimator for the gradient [5]. We repeat the gradient estimation step above for each word in $\tilde{x}_{t-1}$ that is not equal to the corresponding original word in x. For the words that are not replaced, we denote the estimated gradient as a zero vector $\mathbf{0} \in \mathbf{R}^m$. Finally, we get a group of gradients for all words in $\tilde{x}_{t-1}$ as $\mathbf{G} = [\mathbf{g}_1, \cdots, \mathbf{g}_n]$ as the final estimation of $\nabla Q(\tilde{x}_{t-1})$.

(3) **Updating the Adversarial Example.** Given the gradients $\mathbf{G}$, we expect the deviation between $\tilde{w}_t^{(i)}$ and the i-th word $w_t^{(i)}$ in the new sample x_t' is in the direction of $\mathbf{g}_i$. However, we have only a set of *discrete choices* $\mathrm{S}(w^{(i)})$ as the candidates of $w_t^{(i)}$ for x_t' in the text adversarial attack. In other words, the deviation between any candidate word in $\mathrm{S}(w^{(i)})$ and existing word $\tilde{w}_{t-1}^{(i)}$ would not completely align with the gradient direction $\mathbf{g}_i$. To resolve this conundrum due to the discrete distribution of candidates in the embedding space, we turn to the cosine similarity to map the continuous $\mathbf{g}_i$ back to a discrete word for each $\tilde{w}_{t-1}^{(i)}$ in $\tilde{x}_{t-1}$. If $\mathbf{g}_i = \mathbf{0}$, the i-th word $w_t^{(i)} = \tilde{w}_{t-1}^{(i)}$ in the new sample x_t', and otherwise,

$$w_t^{(i)} = \underset{s^{(i)} \in \mathrm{S}(w^{(i)})}{\arg\max}\ \cos(\mathbf{g}_i, \mathcal{H}(s^{(i)}) - \mathcal{H}(\tilde{w}_{t-1}^{(i)})), \tag{6}$$

where $\cos(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a}\cdot\mathbf{b}}{||\mathbf{a}||\cdot||\mathbf{b}||}$ is the cosine similarity measure.

The advantage of using cosine similarity is that it normalizes two input vectors and only considers their deviation of direction in the high-dimensional space. As a result, it can flexibly harness the gradient information to search the word among the discrete choices

Algorithm 1: LeapAttack

Inputs: Text sample to be attacked $x = [w^{(1)}, \cdots, w^{(n)}]$, victim model f, word embedding space $\mathcal{H}$, iteration number T, and sample number k.
Output: Adversarial example x^*.
Conduct random initialization to see if there exists an initial adversarial example x'_0 by continually replacing each original word by a random synonym
if x'_0 exists **then**
  Set the iteration counter $t = 1$;
  while $t \leq T$ **do**
    Attain $\tilde{x}_{t-1}$ by moving x'_{t-1} closer to the decision boundary;
    Estimate the gradient at $\tilde{x}_{t-1}$ for each perturbation by Eq. (5) via k samples;
    Attain x'_t by updating each perturbed word to a new candidate word by Eq. (6), $t \leftarrow t + 1$;
  end
  Return the adversarial example having the highest semantic similarity with x;
end
Return Null;

in $\mathrm{S}(w^{(i)})$. By selecting every new word $w_t^{(i)}$ in this fashion, we will replace each original word $w^{(i)}$ by $w_t^{(i)}$ to construct x'_t. We adopt an early stopping strategy when existing replacements $w_t^{(i)}$ have already made x'_t adversarial. After that, we finally get a new sample $x'_t = [w_t^{(1)}, \cdots, w_t^{(n)}]$ as the output in the t-th iteration and a new perturbation matrix $\mathbf{P}_t$.

2.3 Algorithm

The iterative algorithm of how LeapAttack crafts text adversarial examples in the hard-label setting is summarized in Algorithm 1. After that, LeapAttack takes the adversarial example having the highest semantic similarity with x as the final output x^*.

3 EXPERIMENTS

3.1 Experimental Settings

Datasets. In experiments we adopt the following text datasets collected for text classification and natural language inference: (1) MR [22], a movie review dataset for binary sentiment classification; (2) AG [26], a 4-class news classification dataset; (3) Yahoo [26], a questions-and-answers topic classification dataset with 10 classes; (4) Yelp [26], a binary sentiment classification one; (5) IMDB [17], another binary sentiment classification dataset collected from movie reviews; (6) SNLI [1] and (7) MNLI [24], two datasets collected for the natural language inference task; and (8) mMNLI, a variant of the MNLI dataset with mismatched premise-hypothesis pairs. In our experiments, we follow the setting of [13, 18, 25], taking the same 1,000 test samples of each dataset to conduct adversarial attacks.
Victim Models. We follow existing hard-label adversarial attack methods and adopt the following widely used NLP models as victim models: BERT [8], WordCNN [14], and WordLSTM [12]. The model parameters of the trained models are accessible from the work of [18], and the victim models just output the top-1 prediction.
Baselines. The adopted six baseline methods are all proposed for the black-box text adversarial attack. For the methods that are originally proposed in the soft-label setting, their inputs are the boolean hard-label scores for fair comparison. The baselines include: (1) **TextFooler** [13], a method which replaces the original words in the order of position importance scores, and the replacement for each original word is determined by the prediction probability change that each candidate word brings; (2) **PWWS** [23], a method ranks the order of original words and selects the corresponding candidate by an improved saliency score; (3) **TextBugger** [16], another method that uses the soft-label saliency score for position ranking and substitution; (4) **DeepWordBug** [10], an early work on text adversarial attack which also employs the saliency score of each word to determine the attack order and replacement; (5) **TextHoaxer** [25], a gradient-based hard-label method which estimates the gradient directly from the virtual randomly sampled directions in the embedding space rather than concrete candidate words; and (6) **HLGA** [18], a hard-label method which employs the genetic algorithm to reduce the search space for optimizing the semantic similarity of adversarial examples. It generates a large population of adversarial candidates first, and then conducts crossover, mutation, and selection operations among them to search the best one.

Different from other types of adversarial attacks, no matter in image [5–7] or text [18] data, one necessary condition for conducting hard-label adversarial attack is having initial adversarial examples by random initialization first. For fair comparison, we adopt the textual random initialization [18] for all methods in experiments. It keeps replacing original words with random synonyms until it finds an initial adversarial example, which guarantees to find the upper bound of success attack rate. Since this step decides whether an adversarial example can be found for an original sample in attacking and later steps are adapted from baselines for semantic similarity optimization, it leads to the same attack success rate with the upper bound value after the adversarial attack for all methods.
Evaluation Metrics. The evaluation metrics employed to quantify the adversarial example quality include **semantic similarity** (Sim) and **perturbation rate** (Pert). Following previous works [13, 18], the semantic similarity is calculated by putting the original text sample and generated adversarial example into the Universal Sequence Encoder [4], whose maximum value is 1, and the higher the better. The perturbation rate is the ratio of changed words in the generated sample compared to the original text sample, and the lower the better. In addition, we also count the **query number** to measure attack efficiency.
Implementation Setting. LeapAttack is implemented with an NVIDIA Tesla P100 GPU. The used word embedding space $\mathcal{H}$ is from Counter-Fitted Word Vectors [21]. The synonym set for each word has a size of 50 and is the same as baselines [18]. We set the number of sampled synonyms $k = 5$ and iteration number $T = 100$.

3.2 Experimental Results

3.2.1 Comparison on Semantic Similarity and Perturbation Rate. Since the same random initialization technique leads to the same

Table 1: Comparison of semantic similarity (Sim) and perturbation rate (Pert) when attacking against text classification models. Acc stands for prediction accuracy after adversarial attack, which is determined by the random initialization step and the same for different adversarial attack methods. The original accuracy is indicated by parentheses.

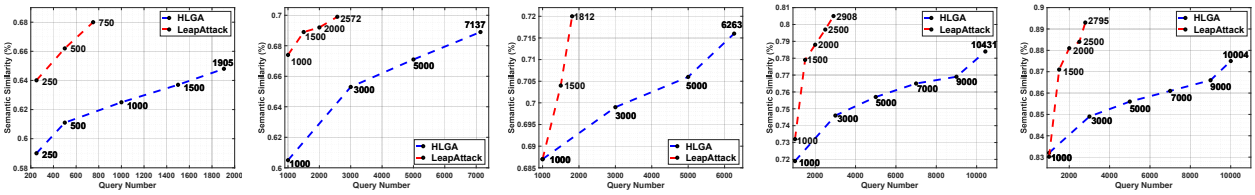
Dataset	Method	BERT			WordCNN			WordLSTM		
		Acc (%)	Sim (%)	Pert (%)	Acc (%)	Sim (%)	Pert (%)	Acc (%)	Sim(%)	Pert (%)
MR	TextFooler	1.0 (← 85.0)	57.1	16.278	0.7 (←76.5)	58.7	15.430	0.7 (← 78.0)	56.7	16.440
	PWWS		61.1	15.945		61.6	15.394		61.1	16.140
	TextBugger		61.9	15.354		64.3	14.746		63.0	15.023
	DeepWordBug		61.6	15.180		63.1	14.960		62.2	15.265
	TextHoaxer		67.3	11.812		68.4	12.076		67.7	12.320
	HLGA		64.8	13.360		66.6	12.976		65.4	13.574
	LeapAttack		**68.0**	**10.420**		**68.6**	**10.742**		**68.1**	**10.959**
AG	TextFooler	2.8 (← 93.0)	58.0	19.070	1.4 (← 90.4)	68.7	15.233	5.7 (← 90.2)	60.5	18.569
	PWWS		58.5	18.830		70.3	14.963		60.9	19.147
	TextBugger		61.7	17.162		72.1	13.788		63.9	16.814
	DeepWordBug		61.0	17.150		72.3	13.808		64.1	17.261
	TextHoaxer		63.8	15.721		74.2	12.493		64.6	16.122
	HLGA		68.9	12.750		78.2	10.250		70.9	12.864
	LeapAttack		**69.9**	**10.852**		**78.9**	**8.987**		**71.8**	**11.208**
Yahoo	TextFooler	0.5 (← 79.1)	64.9	8.926	0.8 (← 71.1)	71.5	9.225	1.9 (← 73.7)	61.3	10.290
	PWWS		65.2	8.652		72.5	9.228		63.5	10.235
	TextBugger		68.1	7.820		73.9	8.617		65.0	9.620
	DeepWordBug		67.1	7.967		74.5	8.588		64.7	9.171
	TextHoaxer		70.9	6.726		75.0	7.616		67.2	8.396
	HLGA		71.6	5.865		76.2	6.492		68.4	6.999
	LeapAttack		**72.0**	**4.851**		**78.2**	**5.629**		**69.6**	**5.952**
Yelp	TextFooler	5.2 (← 96.5)	69.6	10.979	0.6 (← 92.9)	77.9	9.688	3.2 (← 94.8)	77.2	9.134
	PWWS		71.4	10.778		78.6	9.815		78.6	8.929
	TextBugger		73.1	9.994		80.1	9.033		79.9	8.370
	DeepWordBug		72.9	10.007		80.2	9.052		79.8	8.264
	TextHoaxer		74.6	9.271		81.3	8.543		80.8	7.942
	HLGA		78.4	7.081		83.8	6.675		83.0	6.166
	LeapAttack		**80.5**	**5.985**		**86.2**	**5.845**		**84.9**	**5.484**
IMDB	TextFooler	0.1 (← 90.3)	82.7	5.734	0.0 (← 87.8)	88.6	4.746	0.3 (← 89.3)	87.3	4.656
	PWWS		82.6	5.868		89.4	4.813		87.9	4.766
	TextBugger		83.9	5.416		89.9	4.510		88.5	4.379
	DeepWordBug		84.2	5.187		89.9	4.360		88.7	4.385
	TextHoaxer		85.3	4.860		90.2	4.268		89.1	4.098
	HLGA		87.5	3.307		91.3	3.001		90.1	2.916
	LeapAttack		**89.3**	**2.894**		**92.9**	**2.881**		**91.3**	**2.681**

prediction accuracy under the adversarial attack, the goal of hard-label text adversarial attack is to improve the semantic similarity and lower the perturbation rate. From Table 1, for one thing, our method can generate optimized adversarial examples with the highest semantic similarity in all cases for text classification models against varied lengths in different datasets. For a dataset with short text samples like AG, LeapAttack increases the average semantic similarity by 1.0%, 0.7%, and 0.9% compared to the second best method when attacking BERT, WordCNN, and WordLSTM, respectively. For a dataset with long ones like Yelp, LeapAttack can increase the average semantic similarity by 2.1%, 2.4%, and 1.9% compared to the second best method when attacking these three models respectively. For another, it is also noteworthy that LeapAttack always achieves the lowest perturbation rate in Table 1. For instance, the perturbation rate of LeapAttack drops by up to 1.898%, 1.263%, and 1.656% compared to the second best method in AG when it attacks BERT, WordCNN, and WordLSTM, respectively. In addition, from Table 2, when attacking natural language inference models LeapAttack, always attains the lowest perturbation rate and has comparatively high semantic similarity in all cases, which further corroborates that LeapAttack generally crafts high-quality text adversarial examples in the hard-label setting.

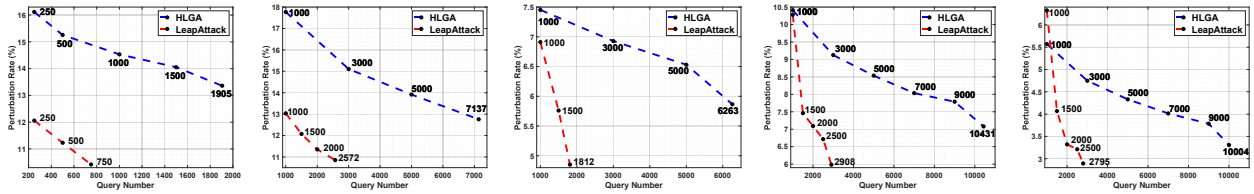
Compared to the baselines, our improvement results from the following aspects. Firstly, in each iteration LeapAttack moves current sample to the decision boundary by replacing original words back, which is contributory to lowering the perturbation rate. Additionally, it characterizes different candidates by their word embeddings. The sampled directions used for gradient estimation are obtained

Table 2: Comparison of semantic similarity and perturbation rate when attacking against a natural language inference model (BERT).

Method	SNLI			MNLI			mMNLI		
	Acc (%)	Sim (%)	Pert (%)	Acc (%)	Sim (%)	Pert (%)	Acc (%)	Sim (%)	Pert (%)
TextFooler		29.7	20.043		42.6	16.455		43.7	16.035
PWWS		31.9	20.018		44.7	16.441		45.9	16.063
TextBugger		33.2	19.312		47.2	15.208		48.1	15.055
DeepWordBug	1.3 (← 89.1)	33.4	19.189	2.9 (← 85.1)	46.8	15.262	1.7 (← 82.1)	47.4	15.248
TextHoaxer		**38.9**	16.414		**53.2**	12.642		**54.7**	12.423
HLGA		36.6	18.158		50.3	14.152		51.8	13.822
LeapAttack		36.2	**15.946**		51.5	**11.946**		53.2	**11.341**



(a) MR (Sim) (b) AG (Sim) (c) Yahoo (Sim) (d) Yelp (Sim) (e) IMDB (Sim)



(f) MR (Pert) (g) AG (Pert) (h) Yahoo (Pert) (i) Yelp (Pert) (j) IMDB (Pert)

Figure 2: (a-e) Semantic similarity (↑) and (f-j) perturbation rate (↓) comparison w.r.t. query number between HLGA and LeapAttack against BERT.

Table 3: Performance comparison given the query number when TextHoaxer achieves convergence against BERT.

Dataset	Method	# Query	Sim (%)	Pert (%)
MR	TextHoaxer	800	67.3	11.812
	LeapAttack	750	68.0	10.420
AG	TextHoaxer	1,400	63.8	15.721
	LeapAttack	1,400	68.8	12.420
		2,572	69.9	10.852
Yahoo	TextHoaxer	1,500	70.9	6.726
	LeapAttack	1,500	70.4	5.760
		1,812	72.0	4.851
Yelp	TextHoaxer	1,381	74.6	9.271
	LeapAttack	1,381	77.1	8.099
		2,908	80.5	5.985
IMDB	TextHoaxer	1,416	85.3	4.860
	LeapAttack	1,416	86.7	4.201
		2,795	89.3	2.894

by mapping the candidate words to embeddings rather than virtual random vectors, which can take into consideration the whole candidate set. Besides, the gradient used for word replacement is selectively merged from successful perturbation explorations and mapped back to a word by cosine similarity, which helps discover words leading to higher semantic similarity with high flexibility.

3.2.2 Comparison on Attack Efficiency. The results above show that HLGA is a strong baselines in the hard-label setting. Since efficiency is also crucial in evaluating the attack performance, we further compare LeapAttack with HLGA on the query number that they require to achieve convergence. Without loss of generality, we compare the query number in the case of attacking BERT on five text classification datasets, as shown in Figure 2. These figures demonstrate that the number of queries that HLGA takes to achieve convergence is 2.54, 2.77, 3.46, 3.59, and 3.58 times that of LeapAttack in MR, AG, Yahoo, Yelp, and IMDB, respectively, which is because LeapAttack does not rely on past adversarial candidates and optimizes based on only one candidate. The result also shows that LeapAttack can perform well even when the number of queries is relatively limited, which indicates that LeapAttack maintains its good performance even in the scenario where victim models limit the number of queries. This is a common constraint for existing commercial deep learning-based NLP platforms such as Microsoft

Table 4: Comparison results of attacking models which defend with adversarial training.

Dataset	Method	WordCNN			WordLSTM		
		Acc (%)	Sim (%)	Pert (%)	Acc (%)	Sim(%)	Pert (%)
AG	TextFooler	6.1 (← 92.4)	71.00	10.92	7.3 (← 94.3)	68.01	12.13
	PWWS		71.84	10.87		68.79	12.09
	TextBugger		73.51	10.03		71.33	10.95
	DeepWordBug		73.66	9.96		71.04	10.98
	TextHoaxer		76.42	7.98		74.51	8.13
	HLGA		69.78	10.84		66.71	12.06
	LeapAttack		**79.15**	**6.77**		**77.29**	**7.12**
IMDB	TextFooler	6.3 (← 87.2)	83.50	5.89	10.3 (← 89.3)	84.10	4.08
	PWWS		83.99	5.90		84.85	4.09
	TextBugger		84.55	5.61		85.40	3.85
	DeepWordBug		84.74	5.58		85.50	3.87
	TextHoaxer		85.21	5.55		86.50	3.19
	HLGA		82.93	5.89		83.61	4.09
	LeapAttack		**88.69**	**3.86**		**89.14**	**2.59**

Table 5: Influence on the adversarial attack performance of the number of samples k when attacking WordLSTM.

k	MR		AG		Yahoo		Yelp		IMDB	
	Sim (%)	Pert (%)	Sim (%)	Pert (%)	Sim(%)	Pert (%)	Sim (%)	Pert (%)	Sim (%)	Pert (%)
3	67.9	10.774	71.8	11.045	69.6	5.670	85.2	5.309	91.5	2.719
5	68.1	10.959	71.8	11.208	69.6	5.952	84.9	5.484	91.3	2.681
7	68.0	11.061	71.7	11.423	69.8	5.814	85.2	5.455	91.4	2.790
10	68.7	10.920	71.3	11.591	69.6	6.036	84.9	5.549	91.4	2.808

Azure Text Analytics, where users can only request 5,000 times monthly for free. Taking the query limits into consideration, our attack method is more applicable to the robustness evaluation of real-world systems.

We further compare LeapAttack with TextHoaxer, a gradient-based method proposed to achieve fast convergence to reduce the query number, on attack efficiency. Without loss of generality, in Table 3, we show the query number for TextHoaxer to converge and its corresponding performance against BERT. LeapAttack generally attains better performance given the budget number when TextHoaxer converges, and its performance still greatly improves during getting to the convergence. Such results indicate that the round trip process of LeapAttack can help enhance the generation quality of gradient-based methods with high efficiency by using the full set of candidates for gradient estimation compared to the virtual random directions used in TextHoaxer.

3.2.3 Comparison on Attacking Existing Defense Mechanism. Since existing NLP models sometimes adopt defense mechanism for robust prediction, we further compare the attack performance of different methods when the victim model is trained with HotFlip [9], which is an adversarial training technique in text data. Due to the availability of implementation code of [28], we compare the attack performance on attacking adversarially trained WordCNN and WordLSTM on AG and IMDB. The results shown in Table 4 indicate that despite the introduction of defense mechanism, LeapAttack consistently generates text adversarial examples with the highest semantic similarity and lowest perturbation rate in all cases, which further verifies its broad applicability.

Table 6: Ablation study results on five datasets.

Dataset	Method	Sim (%)	Pert (%)
MR	Random Initialization	18.0	38.942
	w/o Moving Bdy	63.5	12.852
	w/o Grad	62.2	13.013
	LeapAttack	**68.6**	**10.742**
AG	Random Initialization	30.4	43.853
	w/o Moving Bdy	69.9	15.464
	w/o Grad	70.5	13.311
	LeapAttack	**78.9**	**8.987**
Yahoo	Random Initialization	27.5	32.796
	w/o Moving Bdy	68.4	10.890
	w/o Grad	71.9	8.446
	LeapAttack	**78.2**	**5.629**
Yelp	Random Initialization	17.3	39.014
	w/o Moving Bdy	59.4	18.279
	w/o Grad	78.3	9.048
	LeapAttack	**86.2**	**5.845**
IMDB	Random Initialization	34.2	31.270
	w/o Moving Bdy	69.1	13.639
	w/o Grad	89.1	4.459
	LeapAttack	**92.9**	**2.881**

3.2.4 Ablation Study. Furthermore, we investigate the influence of the number of samples k on the adversarial attack performance to

Table 7: Generated adversarial text by LeapAttack. The substituted original word is strikethroughed, and the replacement is the following one.

Adversarial Example	Change of Prediction
MR: It's weird, ~~wonderful~~ ravishing, and not necessarily for kids.	Positive → Negative
AG: Brockton upset makes Waltham 's ~~day~~ jours. Waltham – Paul Mayberry was dripping, Luis Cotto was weeping, and Alex Russo was kneeling on the sideline ···	Sports → World
Yelp: Great buffet. Lots of selections. The prime rib was ~~delicious~~ loverly. It was worth the 30 dollars.	Positive → Negative

evaluate the gradient estimation design. Without loss of generality, we change k from 3 to 10 when we attack WordLSTM on five text classification datasets. As shown in Table 5, the number of samples does not have a significant influence on the adversarial attack performance with respect to semantic similarity and perturbation rate. These results demonstrate that LeapAttack can maintain stable adversarial attack performance with different k, indicating the design of LeapAttack is reasonable for the gradient estimation.

In addition, without loss of generality, we further conduct an ablation study to substantiate the optimization design on five datasets when attacking WordCNN. The baseline in the ablation study is the initial adversarial examples output by the random initialization. Besides the random initialization scenario, we also consider two variants of the proposed LeapAttack.

In the first scenario (w/o Moving Bdy), we estimate the gradient without the step of moving the adversarial example closer to the decision boundary. In the second scenario (w/o Grad), we randomly select a word as the replacement after moving near the decision boundary without using the estimated gradient. The final scenario is the designed LeapAttack. As shown in Table 6, the first scenario shows if we do not estimate the gradient in a location near the decision boundary, the estimated gradient will become inaccurate and mislead the optimization, leading to worse performance. As for the second scenario, it shows that the adversarial example construction will get sub-optimal if we do not harness the gradient information to guide the substitution selection. Thus, the designed mechanisms in LeapAttack are reasonable.

3.2.5 Case Study. In Table 7, we also list some adversarial examples generated by LeapAttack, which demonstrate that it has the ability of fooling victim models by appropriate substitutions. For instance, when people read the adversarial example of MR in Table 7 which replaces "wonderful" by "ravishing", they usually will not think these two sentences have much semantic difference.

4 RELATED WORK

4.1 Image Adversarial Attack

Adversarial attack attracts great interest due to the observation that image classification DNNs can be fooled by adversarial examples despite their superior performance [2, 3, 11, 15, 27]. For example, Goodfellow et al. [11] find that DNNs can be easily fooled when attackers add small perturbation to the original test image in the image recognition task. Early image adversarial attack works relies on the use of model parameter gradients to generate adversarial examples, which are termed "white-box" adversarial attack. For example, Kurakin et al. [15] get the adversarial examples in an iterative framework where the perturbation added on the original images in each iteration is calculated by the gradient backpropagated from the cross-entropy cost function. However, due to the previous unrealistic assumption on parameter gradients, recent image adversarial attack works shift to the hard-label black-box setting, where the attackers only have the knowledge of the top-1 predicted label from the victim model. For this setting, exemplary representative methods include Sign-OPT [7] and HopJumpSkipAttack [5]. For Sign-OPT, it formulates the hard-label image adversarial attack problem as a gradient-based optimization problem in the continuous pixel space, which is to find a direction in the high-dimensional pixel space such that in that direction the designed method can generate adversarial examples that looks most similar to the original samples. As for HopSkipJumpAttack, it is a gradient-based framework that uses the Monte Carlo estimate of the gradient to guide the search of image adversarial examples.

Although these gradient-based methods work well in image data, they cannot be applied in our problem for they fail to handle discrete text data. Firstly, they both rely on the binary search procedure to get adversarial examples close to the decision boundary. However, it cannot be applied in text because text lack the continuous property of image pixels and the candidate samples only distribute in certain places in the high-dimensional word embedding space. Secondly, when estimating the gradient, image adversarial attack methods assume that they can sample the perturbation direction anywhere without limitation, but this operation is inapplicable in text because the selection of word perturbation is discrete. Additionally, these image adversarial attack methods simply add the gradient to the perturbed image after getting the estimated gradient, but the calculated continuous gradient may be meaningless in text data if it does not match any specific candidate. As for LeapAttack, it (1) abandons binary search and moves current adversarial examples closer to the decision boundary by putting original words back, (2) bridges the gap caused by the discrete word choices by expressing perturbation direction as the difference vector between two word embeddings and mapping a word replacement from the synonym set to a difference vector for gradient estimation, and (3) has an extra step of mapping the continuous gradient to a discrete word based on the cosine similarity metric. As a result, it can flexibly aggregate explored perturbations as the gradient to search the optimal one.

4.2 Text Adversarial Attack

Text adversarial attacks can be categorized into *white-box* and *black-box* ones. Early work mainly focuses on the white-box scenario, assuming that attackers have the knowledge of parameter gradients. For instance, Ebrahimi et al. [9] propose a white-box method which constructs text adversarial examples based on the gradients of one-hot input vectors. However, this setting is not realistic in real-world applications. Therefore, black-box text adversarial attacks

have received more attention recently, and the current approaches mainly focus on the following two settings:

(1) Soft-label black-box text adversarial attacks, where attackers have the knowledge of the prediction score distribution of different categories given an input text sample [10, 13, 16, 23]. In consequence, their adversarial example generation processes normally adopt a greedy mechanism to determine the replaced word positions and their substitutions. For instance, TextFooler [13] ranks the order of attacked words by their importance scores and selects the candidate that can lead to the highest prediction score drop of the original label as the replacement before the text turns adversarial.

(2) Hard-label black-box text adversarial attacks, where attackers only know the top-1 predicted label [18]. The first successful work in this setting [18] uses GA for optimization. One of its limitations is that GA depends on past adversarial candidates for mutation and crossover, which may accumulate errors caused by the past inappropriate substitutions. Besides, this approach has to frequently query the victim models for all candidates to achieve convergence. Later, Ye et al. [25] propose a gradient-based hard-label attack method named TextHoaxer in the tight-budget setting. However, TextHoaxer samples virtual perturbation direction for gradient estimation, which can only use partial candidates and does not fully exploit all candidates for gradient estimation.

Different from the aforementioned methods, LeapAttack proposes a more effective gradient-based optimization framework with respect to difference vectors of the perturbation in the word embedding space for the hard-label text adversarial attack. Using the embedding space as a bridge of perturbation directions and candidate words, it alleviates existing limitations by getting perturbation directions for gradient estimation from concrete word candidates and mapping estimated gradient back to the replacement word by cosine distance under one adversarial candidate.

5 CONCLUSION

Hard-label text adversarial attack is a more challenging setting that can more realistically reveal the robustness of text DNNs. However, the performance of existing methods is limited by the discrete nature of text data, which hinders the development of gradient-based optimization. To relieve this conundrum in the hard-label setting, we propose a gradient-based solution named LeapAttack empowered by expressing perturbation in the high-dimensional word embedding space. In each iteration, it firstly estimates the gradient at the decision boundary by transforming the semantic deviation brought by each sampled word into a difference vector. Later, it maps the estimated gradient to an updated replacement by the cosine similarity. Extensive experiment results on eight datasets consistently demonstrate that LeapAttack is generally effective and efficient in crafting high-quality text adversarial examples with the highest semantic similarity and lowest perturbation rate in the hard-label setting.

ACKNOWLEDGMENTS

Ting Wang is partially supported by the National Science Foundation under Grant No. 1953893, 1951729, and 2119331.

REFERENCES

[1] Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large annotated corpus for learning natural language inference. In *EMNLP*. The Association for Computational Linguistics, 632–642.
[2] Wieland Brendel, Jonas Rauber, and Matthias Bethge. 2018. Decision-Based Adversarial Attacks: Reliable Attacks Against Black-Box Machine Learning Models. In *ICLR*. OpenReview.net.
[3] Nicholas Carlini and David A. Wagner. 2017. Towards Evaluating the Robustness of Neural Networks. In *S&P*. IEEE Computer Society, 39–57.
[4] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St John, Noah Constant, Mario Guajardo-Céspedes, Steve Yuan, Chris Tar, et al. 2018. Universal sentence encoder. *arXiv preprint arXiv:1803.11175* (2018).
[5] Jianbo Chen, Michael I. Jordan, and Martin J. Wainwright. 2020. HopSkipJumpAttack: A Query-Efficient Decision-Based Attack. In *S&P*. IEEE, 1277–1294.
[6] Minhao Cheng, Thong Le, Pin-Yu Chen, Huan Zhang, Jinfeng Yi, and Cho-Jui Hsieh. 2019. Query-Efficient Hard-label Black-box Attack: An Optimization-based Approach. In *ICLR*. OpenReview.net.
[7] Minhao Cheng, Simranjit Singh, Patrick H. Chen, Pin-Yu Chen, Sijia Liu, and Cho-Jui Hsieh. 2020. Sign-OPT: A Query-Efficient Hard-label Adversarial Attack. In *International Conference on Learning Representations*.
[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL-HLT*. Association for Computational Linguistics, 4171–4186.
[9] Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. 2018. HotFlip: White-Box Adversarial Examples for Text Classification. In *ACL*. Association for Computational Linguistics, 31–36.
[10] Ji Gao, Jack Lanchantin, Mary Lou Soffa, and Yanjun Qi. 2018. Black-Box Generation of Adversarial Text Sequences to Evade Deep Learning Classifiers. In *SP Workshops*. IEEE Computer Society, 50–56.
[11] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. In *ICLR*.
[12] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Comput.* 9, 8 (1997), 1735–1780.
[13] Di Jin, Zhijing Jin, Joey Tianyi Zhou, and Peter Szolovits. 2020. Is BERT Really Robust? A Strong Baseline for Natural Language Attack on Text Classification and Entailment. In *AAAI*. AAAI Press, 8018–8025.
[14] Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. In *EMNLP*. ACL, 1746–1751.
[15] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. 2017. Adversarial examples in the physical world. In *ICLR*. OpenReview.net.
[16] Jinfeng Li, Shouling Ji, Tianyu Du, Bo Li, and Ting Wang. 2019. TextBugger: Generating Adversarial Text Against Real-world Applications. In *NDSS*.
[17] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning Word Vectors for Sentiment Analysis. In *ACL*. The Association for Computer Linguistics, 142–150.
[18] Rishabh Maheshwary, Saket Maheshwary, and Vikram Pudi. 2021. Generating Natural Language Attacks in a Hard Label Black Box Setting. In *AAAI*.
[19] Paul Michel, Xian Li, Graham Neubig, and Juan Miguel Pino. 2019. On Evaluation of Adversarial Perturbations for Sequence-to-Sequence Models. In *NAACL-HLT 2019*. Association for Computational Linguistics, 3103–3114.
[20] Shakir Mohamed, Mihaela Rosca, Michael Figurnov, and Andriy Mnih. 2020. Monte Carlo Gradient Estimation in Machine Learning. *J. Mach. Learn. Res.* 21, 132 (2020), 1–62.
[21] Nikola Mrksic, Diarmuid Ó Séaghdha, Blaise Thomson, Milica Gasic, Lina Maria Rojas-Barahona, Pei-Hao Su, David Vandyke, Tsung-Hsien Wen, and Steve J. Young. 2016. Counter-fitting Word Vectors to Linguistic Constraints. In *NAACL HLT 2016*. The Association for Computational Linguistics, 142–148.
[22] Bo Pang and Lillian Lee. 2005. Seeing Stars: Exploiting Class Relationships for Sentiment Categorization with Respect to Rating Scales. In *ACL*. The Association for Computer Linguistics, 115–124.
[23] Shuhuai Ren, Yihe Deng, Kun He, and Wanxiang Che. 2019. Generating Natural Language Adversarial Examples through Probability Weighted Word Saliency. In *ACL*. Association for Computational Linguistics, 1085–1097.
[24] Adina Williams, Nikita Nangia, and Samuel R. Bowman. 2018. A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference. In *NAACL-HLT*. Association for Computational Linguistics, 1112–1122.
[25] Muchao Ye, Chenglin Miao, Ting Wang, and Fenglong Ma. 2022. TextHoaxer: Budgeted Hard-Label Adversarial Attacks on Text. *AAAI* (2022).
[26] Xiang Zhang, Junbo Jake Zhao, and Yann LeCun. 2015. Character-level Convolutional Networks for Text Classification. In *NeurIPS*. 649–657.
[27] Yao Zhou, Jun Wu, Haixun Wang, and Jingrui He. 2020. Adversarial Robustness through Bias Variance Decomposition: A New Perspective for Federated Learning. *arXiv preprint arXiv:2009.09026* (2020).
[28] Yi Zhou, Xiaoqing Zheng, Cho-Jui Hsieh, Kai-Wei Chang, and Xuanjing Huang. 2021. Defense against Synonym Substitution-based Adversarial Attacks via Dirichlet Neighborhood Ensemble. In *ACL*. Online.