SSPAttack: A Simple and Sweet Paradigm for Black-Box Hard-Label Textual Adversarial Attack

Han Liu[1], Zhi Xu[1], Xiaotong Zhang[1*], Xiaoming Xu[1], Feng Zhang[2], Fenglong Ma[3], Hongyang Chen[4], Hong Yu[1], Xianchao Zhang[1]

[1] Dalian University of Technology, Dalian, China
[2] Peking University, Beijing, China
[3] The Pennsylvania State University, Pennsylvania, USA
[4] Zhejiang Lab, Hangzhou, China
liu.han.dut@gmail.com, xu.zhi.dut@gmail.com, zxt.dut@hotmail.com, xu.xm.dut@gmail.com, zfeng.maria@gmail.com, fenglong@psu.edu, dr.h.chen@ieee.org, {hongyu,xczhang}@dlut.edu.cn

Abstract

Hard-label textual adversarial attack is a challenging task, as only the predicted label information is available, and the text space is discrete and non-differentiable. Relevant research work is still in fancy and just a handful of methods are proposed. However, existing methods suffer from either the high complexity of genetic algorithms or inaccurate gradient estimation, thus are arduous to obtain adversarial examples with high semantic similarity and low perturbation rate under the tight-budget scenario. In this paper, we propose a simple and sweet paradigm for hard-label textual adversarial attack, named SSPAttack. Specifically, SSPAttack first utilizes initialization to generate an adversarial example, and removes unnecessary replacement words to reduce the number of changed words. Then it determines the replacement order and searches for an anchor synonym, thus avoiding going through all the synonyms. Finally, it pushes substitution words towards original words until an appropriate adversarial example is obtained. The core idea of SSPAttack is just swapping words whose mechanism is simple. Experimental results on eight benchmark datasets and two real-world APIs have shown that the performance of SSPAttack is sweet in terms of similarity, perturbation rate and query efficiency.

Introduction

Deep neural networks (DNNs) have achieved significant development in various application domains, such as computer vision (Wu et al. 2021; Han et al. 2021), natural language processing (Lu et al. 2021; Seker et al. 2022), robotics (Terreran and Ghidoni 2021; Centurelli et al. 2022) and so on. Recent studies (Gao et al. 2018; Jin et al. 2020) have shown that DNNs are vulnerable to adversarial attacks. Adversarial attacks aim to generate adversarial examples that are purposely designed to induce a model to make wrong predictions but imperceptible to human. Most adversarial attack methods (Chen, Jordan, and Wainwright 2020; Cheng et al. 2020) focus on the computer vision domain and have shown powerful performance. However, generating adversarial examples in the text domain is more challenging. On the one hand, changing words in a sentence slightly may cause tremendous changes in semantics. On the other hand, the text space is discrete and non-differentiable. Existing textual adversarial attack methods are very few and the relevant research is still in the early stage.

Existing textual adversarial attack methods can be divided into two categories: **white-box attacks** (Meng and Wattenhofer 2020; Zou et al. 2020) and **black-box attacks** (Jin et al. 2020; Ye et al. 2022). The white-box attack assumes that the attacker can access all the model information, such as parameters and gradients. Therefore, the white-box attack usually utilizes gradients and model parameters to generate adversarial examples. Different from the white-box attack, the black-box attack methods can only access the output information of the victim model. There are two settings in black-box based attack: **soft-label setting** (Jin et al. 2020; Gao et al. 2018; Zang et al. 2020) and **hard-label setting** (Maheshwary, Maheshwary, and Pudi 2021; Ye et al. 2022). The soft-label setting can obtain the predicted label and corresponding confidence score of victim model. Therefore, these methods can utilize the confidence score to compute the importance score of each word, and then leverage it to determine the replacement order and replace the corresponding words. In the hard-label setting, attackers only have access to the predicted label of the target model. Therefore, compared with the soft-label setting, the hard-label setting is more challenging and realistic.

Only a few approaches have attempted to tackle the black-box hard-label textual adversarial attack task. Specifically, (Maheshwary, Maheshwary, and Pudi 2021) leverages a population-based optimization algorithm to generate adversarial examples. Although it can craft semantically similar examples, it needs a high budget, which makes it unrealistic. This drawback comes from the necessity of a large population of adversarial candidates. TextHoaxer (Ye et al. 2022) utilizes the word embedding space, namely perturbation matrix, to represent perturbations. It then uses gradient-based method to optimize the perturbation matrix. However, the gradient estimation may be inaccurate, thus leading to some unnecessary queries. This issue stemmed from two perspectives: on the one hand, it is extremely hard to find the op-

*Corresponding author.

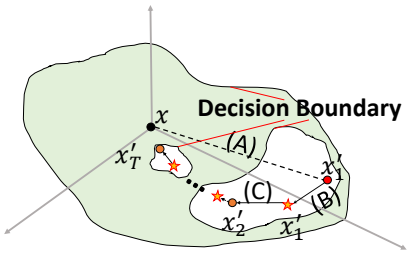


Figure 1: The overview of SSPAttack. (A) Generate an adversarial example x'_1 by random initialization. (B) Update x'_t $(1 \leq t \leq T)$ by removing unnecessary replacement words. (C) Obtain x'_{t+1} by pushing substitution words towards original words.

timal replacement in the discrete text space quickly; on the other hand, TextHoaxer uses random sampling to optimize the perturbation matrix, which causes a waste of queries.

In this paper, we propose a simple and sweet paradigm for black-box hard-label textual adversarial attack, named SSPAttack. Its overview is shown in Figure 1. By the word "simple", we mean that the core idea of SSPAttack is just swapping words whose mechanism is very easy. By the word "sweet", we mean that SSPAttack outperforms other strong baselines significantly. In particular, SSPAttack first initializes an adversarial example by swapping synonyms and remove unnecessary replacement words by swapping back original words. Then we determine the replacement order by utilizing counter-fitting word vectors and search an anchor synonym, thus avoiding the global search. Finally, we push substitution words towards original words to further optimize the semantic similarity. We perform experiments on eight benchmark datasets and two real-world APIs (Google Cloud and Alibaba Cloud). Experimental results show that SSPAttack performs better than other strong baselines in terms of similarity and perturbation rate in the same query budget.

Related Work

Soft-Label Black-Box Textual Adversarial Attack Soft-label attack relies on the predicted label and corresponding confidence score to generate adversarial examples. In this setting, attackers can utilize the confidence score to calculate the word importance (Jin et al. 2020; Gao et al. 2018) and then use a greedy mechanism to generate adversarial examples. Specifically, TextFooler (Jin et al. 2020) first determines the importance of each word and then replaces words one by one according to the word importance until adversarial examples are generated. There are also some methods (Zang et al. 2020) using a combinatorial optimization algorithm to craft adversarial examples. In the soft-label setting, the role of predicted labels is to judge whether to generate adversarial examples.

Hard-Label Black-Box Textual Adversarial Attack Hard-label attack is more challenging since it can only access the predicted label, and only few available methods are proposed. (Maheshwary, Maheshwary, and Pudi 2021) uses a population-based optimization procedure to craft adversarial examples. Specifically, it uses random initialization and search space reduction to get an original example, and then uses the genetic algorithm to optimize the example. This strategy can generate adversarial examples with high semantic similarity and low perturbation rate. However, its query efficiency is high, which is difficult to implement in real applications. Ye et al. (2022) propose a gradient-based method, namely TextHoaxer, to attack DNNs in the hard-label setting. Different from the previous method, TextHoaxer introduces a perturbation matrix and a novel objective function that include three terms, i.e., a semantic similarity term, a pair-wise perturbation constraint and a sparsity constraint. By using the gradient-based procedure to optimize the perturbation matrix, TextHoaxer can generate a series of adversarial examples. However, as the text space is discrete and the gradient estimation may be inaccurate, the quality of generated adversarial examples may be seriously affected.

Symbol	Explanation
x	the original example $x = [w_1, w_2, ..., w_n]$
x'	the adversarial example $x' = [w'_1, w'_2, ..., w'_n]$
x'_t	the adversarial example in the t-th step
x'_{t+1}	the adversarial example in the $t+1$-th step
$x_t^{(i)}$	x'_t after modifying the i-th word
w_i	the i-th word of x
w'_i	the i-th word of x'_t
$\bar{w}_i$	the anchor synonym used to reduce search space
$\bar{w}'_i$	the final replacement word
S_{w_i}	the synonym set of w_i
f	the victim model
y	the true label of x
d_i	the Euclidean distance between w_i and w'_i
$sim(\cdot,\cdot)$	the similarity function between words
$Sim(\cdot,\cdot)$	the similarity function between sentences

Table 1: Symbol explanation.

Problem Formulation

Before introducing the proposed method, we first formally define the task of hard-label adversarial attack in the text domain. Table 1 summarizes some symbol explanation in details. Let $x = [w_1, w_2, ..., w_n]$ denote an original example, where w_i is the i-th word, and n is the number of words. y is the ground truth label of x. By replacing the original words w_i with a synonym w'_i in the synonym set S_{w_i} constantly, we can get an adversarial example $x' = [w'_1, w'_2, ..., w'_n]$, which can lead the victim model f to make a wrong prediction:

$$f(x') \neq f(x) = y, \tag{1}$$

where Eq. (1) can be treated as the adversarial condition. The goal of textual adversarial example is to find the best adversarial example x^* whose semantic similarity with x is the highest among all x'. Formally,

$$x^* = \arg\max_{x'} Sim(x, x'), \ \text{ s.t. } \ f(x') \neq f(x), \tag{2}$$

where $Sim(\cdot,\cdot)$ is the semantic similarity between x and x'.

The Proposed Strategy

Initialization

Most existing black-box attack methods (Jin et al. 2020; Gao et al. 2018) heavily rely on the confidence score to calculate the importance of each word. They replace words until a reasonable adversarial example is generated. However, the confidence score in hard-label adversarial attack is inaccessible. To alleviate the above issue, we follow the previous methods (Maheshwary, Maheshwary, and Pudi 2021; Ye et al. 2022) to utilize the random initialization to generate an adversarial example. Specifically, for each word w_i whose part-of-speech (POS) is *verb*, *noun*, *adjective* or *adverb* in the original text x, we randomly choose a synonym w_i' from w_i's synonym set S_{w_i} as the replacement of w_i, and repeat the above procedure until the generated x' can satisfy the adversarial condition.

However, just using the random initialization to generate adversarial example will inevitably change many words, which will cause the low semantic similarity with the original text. To tackle this problem, we propose to iteratively optimize the semantic similarity between the original text x and the adversarial example in t-th iteration x_t', where $(1 \leq t \leq T)$ and T is the total number of iterations. Furthermore, when we have an adversarial example in the t-th iteration x_t', we can obtain the adversarial example in the $t+1$-th iteration x_{t+1}' by the following two steps: **remove unnecessary replacement words** and **push substitution words towards original words**.

Remove Unnecessary Replacement Words

By unnecessary replacement words, we mean the words that after replacing them with the original words, the example is still an adversarial example. Intuitively, replacing the unnecessary replacement words with the original words is a straightforward way to reduce the perturbation rate and improve the semantic similarity. Based on the above intuition, given the adversarial example $x_t' = [w_1', ..., w_i', ..., w_n']$ in the t-th step, we can remove unnecessary replacement words in x_t' by the following steps.

1. For a word w_i' in x_t', if w_i' is different from w_i, we replace w_i' with w_i, and then get a new example $x_t^{(i)} = [w_1', .., w_i, ..., w_n']$. Assume that there exist m different words between x_t' and x, we can get a candidate set X.
2. Filter out the examples which violate the adversarial condition out of X, and calculate the semantic similarity between the remaining examples and the original text. Assume that $x_t^{(i)} = [w_1', w_2', ..., w_i, ..., w_n']$ meets the adversarial condition, then we calculate the similarity between $x_t^{(i)}$ and x and take the value as the importance score of w_i. If the importance score of a word is large, replacing this word back will preserve the similarity with the original text as much as possible.
3. Replace the original word w_i back to x_t' one by one in the descending order of importance score until x_t' is not an adversarial example.

Algorithm 1: Push substitution words towards original words

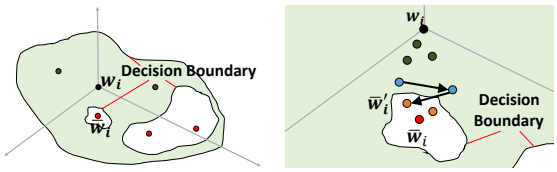
Input: Original text $x = [w_1, ..., w_n]$, adversarial example $x_t' = [w_1', ..., w_n']$, victim model f, and sample number k, the similarity function between words $sim(\cdot, \cdot)$
Output: Adversarial example x_{t+1}'

Determine the replacement order I
$x_{t+1}' \leftarrow x$
for i in I **do**
  $w_{tmp} \leftarrow x_t'[i]$
  Sample k words $C_i = \{s_i^{(1)}, ..., s_i^{(k)}\}$ from $S_{x[i]}$
  $x_{tmp} \leftarrow x_t'$
  for s_i in C_i **do**
    $x_{tmp}[i] \leftarrow s_i$
    if $f(x_{tmp}) \neq f(x)$ **then**
      $L.append(s_i)$
    end if
  end for
  if $L.length = 0$ **then**
    continue
  end if
  $\bar{w} \leftarrow L[0]$
  for s_i in $S_{x[i]}$ **do**
    if $sim(s_i, \bar{w}) \geq sim(s_i, x[i])$ **then**
      $scr_i \leftarrow sim(s_i, x[i])$
      $C_i'.append((scr_i, s_i))$
    end if
  end for
  Sort C_i' by scr_i in descending order
  for s_i' in C_i' **do**
    $x_{tmp}[i] \leftarrow s_i'$
    if $f(x_{tmp}) \neq f(x)$ **then**
      $w_{tmp} \leftarrow s_i'$
      break
    end if
  end for
  $x_{t+1}'[i] \leftarrow w_{tmp}$
  if $f(x_{t+1}') \neq f(x)$ **then**
    return x_{t+1}'
  end if
end for
return x_{t+1}'

After the above procedure, we can obtain a new adversarial example x_t'. By removing unnecessary replacement words, we can reduce the number of changed words to some extent.

Push Substitution Words towards Original Words

To improve the semantic similarity between the adversarial example and the original text, we need to push substitute words toward original words. To this end, a straightforward way is to directly scan the synonym set of the original word to find a word, which simultaneously satisfies the adversarial condition and has the largest similarity. However, directly traversing requires a large number of queries. To alleviate this issue, we propose a novel method to push substitution words towards original words, which consists of three



(a) (b)

Figure 2: (a) Search an anchor synonym $\bar{w}_i$ that is the nearest to w_i and make the modified example outside the decision boundary. (b) Pick out all synonyms from S_{w_i} which are closer to $\bar{w}_i$ (blue and yellow words), and find the synonym $\bar{w}_i'$ that is the nearest to w_i and outside the decision boundary.

steps: determining the replacement order, searching an anchor synonym, and pushing substitution words towards original words. Algorithm 1 describes the detailed procedure.

Determining the Replacement Order This step aims to find a reasonable replacement order. To guarantee the diversity of the generated adversarial example in each adjacent step, we attempt to use the sampling method to determine the replacement order. The probability distribution used by sampling is generated by the following process. For each word w_i' in x_t', we first calculate the Euclidean distance d_i between w_i' and w_i by utilizing counter-fitting word vectors (Mrksic et al. 2016). Then we can obtain the sampling probability p_{d_i} associated with the corresponding position of w_i in x by:

$$p_{d_i} = \frac{\exp(d_i)}{\sum_{j=1}^{l} \exp(d_j)}, \quad d_i = ||v_{w_i'} - v_{w_i}||_2^2, \qquad (3)$$

where l is the number of changed words. v_{w_i} and $v_{w_i'}$ are the word vectors of w_i and w_i', respectively. Through sampling based on the probability distribution, we can determine the replacement order.

Searching an Anchor Synonym This step aims to search an anchor synonym, thus avoiding going through all the synonyms. Specifically, given the adversarial example $x_t' = [w_1', ..., w_i', ..., w_n']$ and the current replacement word w_i, we first sample k synonyms from S_{w_i}, use these synonyms to replace w_i' in x_t', and find the word $\bar{w}_i$ which has the highest semantic similarity with w_i and also makes the new adversarial example satisfy the adversarial condition. Here $\bar{w}_i$ is the anchor synonym at the current step. Figure 2(a) show this procedure. By using $\bar{w}_i$, as the anchor can make the example adversarial, we only need to search for the synonyms close to the anchor, thus avoiding the global search.

Pushing Substitution Words towards Original Words This step aims to optimize the semantic similarity while retaining the adversarial condition. Among the synonyms of w_i, there may exist a better replacement word $\bar{w}_i'$ which is more semantically similar to w_i than $\bar{w}_i$. As shown in Figure 2(b), we can pick out all synonyms in S_{w_i} that have a higher semantic similarity with $\bar{w}_i$ than w_i. Then we use these words to replace the corresponding positions in x_t' one by one in the descending order of similarity with w_i, and find the first synonym $\bar{w}_i'$ that makes the modified example meet the adversarial condition.

Algorithm 2: SSPAttack

Input: Original text $x = [w_1, ..., w_n]$, victim model f, iteration number T.
Output: Adversarial example x^*

```
Attain x' by initialization
if f(x') = f(x) then
    return
end if
t = 1
while t ≤ T do
    Update x'_t by removing unnecessary replacement
    Obtain x'_{t+1} by pushing substitution words towards original words
    t ← t + 1
end while
return the adversarial example having the highest semantic similarity with x
```

The Overall Procedure

Algorithm 2 shows the overall procedure of our proposed SSPAttack. SSPAttack first gets the initial adversarial example by random initialization. Then it enters into the main loop. In each iteration, SSPAttack removes the unnecessary replacement words in this adversarial example to get x_t', and then pushes the substitution words toward the original words until an appropriate adversarial example is generated.

Experiments

Datasets

We conduct experiments on five public text classification datasets and three natural language inference datasets:

1. **MR** (Pang and Lee 2005) is a movie review dataset for binary sentiment classification.
2. **AG's News** (Zhang, Zhao, and LeCun 2015) is a news topic classification dataset of 4 categories include "world", "sports", "business" and "science".
3. **Yahoo** (Zhang, Zhao, and LeCun 2015) is a question-and-answer topic classification dataset.
4. **IMDB** (Maas et al. 2011) is a movie review dataset for binary sentiment but every reivew is longer than MR.
5. **Yelp** (Zhang, Zhao, and LeCun 2015) is a binary sentiment classification dataset.
6. **SNLI** (Bowman et al. 2015) is a dataset based on Amazon Mechanical Turk collected for the natural language inference.
7. **MNLI** (Williams, Nangia, and Bowman 2018) is one of the largest corpora available for natural language inference.
8. **mMNLI** (Williams, Nangia, and Bowman 2018) is a variant of the MNLI dataset with mismatched premise and hypotheses pairs.

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
	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		67.1	11.913		68.4	12.076		67.7	12.320
	HLGA		64.6	13.325		66.6	12.976		65.4	13.574
	SSPAttack		**73.5**	**10.615**		**74.4**	**10.774**		**73.5**	**11.780**
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
	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		63.9	15.373		74.2	12.493		64.6	16.122
	HLGA		68.7	13.325		78.2	10.250		70.9	12.864
	SSPAttack		**73.4**	**10.131**		**82.8**	**8.163**		**75.5**	**10.126**
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
	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		70.3	6.924		75.0	7.616		67.2	8.396
	HLGA		71.3	5.981		76.2	6.492		68.4	6.999
	SSPAttack		**77.3**	**4.642**		**82.6**	**5.334**		**75.2**	**5.633**
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
	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		74.0	9.385		81.3	8.543		80.8	7.942
	HLGA		78.3	7.039		83.8	6.675		83.0	6.166
	SSPAttack		**83.7**	**5.294**		**88.8**	**5.419**		**87.5**	**5.026**
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
	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		84.9	4.852		90.2	4.268		89.1	4.098
	HLGA		86.9	3.354		91.3	3.001		90.1	2.916
	SSPAttack		**90.9**	**2.452**		**94.0**	**2.701**		**92.8**	**2.564**

Table 2: Comparison of semantic similarity (Sim) and perturbation rate (Pert) when attacking against text classification models. Acc stands for model prediction accuracy after the adversarial attack, which is determined by the random initialization step and the same for different adversarial attack.

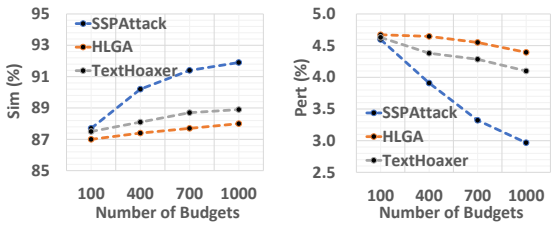
Dataset	Acc	TextHoaxer		HLGA		SSPAttack	
		Sim(%)	Pert(%)	Sim(%)	Pert(%)	Sim(%)	Pert(%)
SNLI	1.3 (← 89.1)	37.6	16.656	35.4	18.536	**54.0**	**15.920**
MNLI	2.9 (← 85.1)	52.9	12.763	50.1	14.420	**64.0**	**12.174**
mMNLI	1.7 (← 82.1)	54.1	12.454	51.8	13.607	**65.1**	**11.526**

Table 3: Comparison of semantic similarity (Sim) and perturbation rate (Pert) when attacking against natural language inference model (BERT).

Baselines

We compare SSPAttack with the following strong baselines:

1. **TextFooler** (Jin et al. 2020) is a soft-label adversarial attack method that replaces the original word according to the word importance score.
2. **DeepWordBug** (Gao et al. 2018) is an early work on



(a) IMDB (Sim) (b) IMDB (Pert)

Figure 3: Performance comparison w.r.t different budget limits between HLGA, TextHoaxer and SSPAttack against WordLSTM.

API	TextHoaxer		SSPAttack	
	Sim(%)	Pert(%)	Sim(%)	Pert(%)
Google Cloud	74.8	8.136	**80.3**	**7.726**
Alibaba Cloud	77.7	6.262	**84.1**	**6.129**

Table 4: Comparison of semantic similarity and perturbation rate when attacking against real-world APIs.

soft-label adversarial.

3. **TextBugger** (Li et al. 2019) is also an early work on soft-label setting.
4. **HLGA** (Maheshwary, Maheshwary, and Pudi 2021) is a hard-label adversarial attack method that employs the genetic algorithm to optimize the adversarial example.
5. **TextHoaxer** (Ye et al. 2022) is a hard-label adversarial attack method that formulates the budgeted hard-label adversarial attack task on text data as a gradient-based optimization problem of perturbation matrix in the continuous word embedding space.

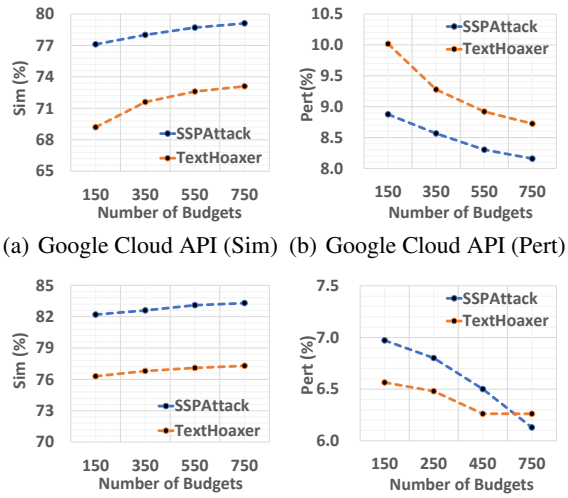
Victim Models

We adopt the following widely used NLP models as victim models: BERT (Devlin et al. 2019), WordCNN (Kim 2014), and WordLSTM (Hochreiter and Schmidhuber 1997). For a fair comparison, the model parameters are taken from the previous work (Maheshwary, Maheshwary, and Pudi 2021).

Implementation Details

Evaluation Metrics We follow (Ye et al. 2022) to use two widely used metrics *semantic similarity* and *perturbation rate* to evaluate the performance. Specifically, we use universal sequence encoder (Cer et al. 2018) to calculate the semantic similarity between the original example and the adversarial example. The semantic similarity value is in $[0, 1]$, and the higher, the better. We use the ratio of the changed words in the generated adversarial sample compared with the original example to define the perturbation rate, and the lower, the better.

Parameter Settings For the number of synonyms in S_{w_i}, we set $k = 5$ for all the datasets. In addition, if we re-optimize the same adversarial example 3 times and no new better adversarial examples are found, we randomly go back to the last 3 or 4 adversarial example. We do not re-optimize an adversarial example more than two times. We follow the previous method (Ye et al. 2022) to take 1000 test samples of each dataset to conduct the adversarial attack.



(a) Google Cloud API (Sim) (b) Google Cloud API (Pert)

(c) Alibaba Cloud API (Sim) (d) Alibaba Cloud API (Pert)

Figure 4: Comparison on semantic similarity and perturbation rate in different budget limits.

Experimental Results

Comparison on Semantic Similarity and Perturbation Rate As the random initialization step determines the prediction accuracy, our goal is to find the adversarial examples with higher semantic similarity and lower perturbation rate under the same prediction accuracy. The best results are highlighted in bold. As shown in Tables 2 and 3 in different tasks, SSPAttack can craft adversarial examples that have the highest semantic similarity and the lowest perturbation rate. Specifically, for a dataset with short text samples like MR, SSPAttack increases the average semantic similarity by 6.4%, 6.0%, 5.8% and decreases the average perturbation rate by 1.298%, 1.302%, 0.540% compared to the second best method HLGA when attacking BERT, WordCNN, WordLSTM, respectively. For a dataset with long text samples like Yelp, SSPAttack increases the average semantic similarity by 5.4%, 5.0%, 4.5% and decreases the average perturbation rate by 1.745%, 1.256%, 1.140% compared to the second best method HLGA when attacking BERT, WordCNN, WordLSTM, respectively. All these results demonstrate that SSPAttack can generate high-quality adversarial examples in the hard-label setting.

Comparison on Attack Efficiency In the real world applications, DNNs' platforms usually constrain the number of queries to defend against attacks. Therefore, efficiency is also an essential metric for evaluating adversarial attack strategy. We design experiments to compare the quality of current hard-label methods, including HLGA and TextHoaxer, under the optimizing budget at 100, 400, 700, and

Dataset	Methods	WordCNN			WordLSTM		
		Acc(%)	Sim(%)	Pert(%)	Acc(%)	Sim(%)	Pert(%)
AG	HLGA	5.9 (← 91.7)	69.5	11.730	8.9 (← 93.1)	68.5	11.957
	TextHoaxer		75.3	9.606		74.5	9.391
	SSPAttack		**81.4**	**7.111**		**80.4**	**7.008**
IMDB	HLGA	19.5 (←87.2)	85.2	7.057	6.1 (←88.3)	85.4	4.290
	TextHoaxer		87.0	6.698		87.8	3.801
	SSPAttack		**91.6**	**4.068**		**90.7**	**2.317**

Table 5: Comparison results of attacking models which defend with adversarial training.

Adversarial Example	Change of prediction
MR: A profoundly **stupid** (**moronic**) affair , populating its hackneyed and meanspirited storyline with cardboard **characters** (**distinctive**) and performers who value cash above credibility.	Negative →Positive
Yahoo: So get my liquor license first then worry about the **building** (**edifice**) ? I agree what good is the building if you cant get the license ?	Business & Finance → Education & Reference
SNLI: A person waterskiing in a river with a large wall in the background.[*Premise*] A **dog** (**pups**) waterskiing in a river with a large wall in the background. [*Hypothesis*]	Contradiction → Entailment

Table 6: Adversarial examples generated by SSPAttack against BERT. The substitute words are indicated by parentheses.

1000. As shown in Figure 3, the average semantic similarity of all methods keeps increasing, and the average perturbation rate of all methods keeps decreasing as the budget increases. Meanwhile, no matter semantic similarity or perturbation rate, our method outperforms baselines in all budgets. These results further demonstrate that SSPAttack can craft adversarial examples with highest semantic similarity and lowest perturbation rate in all tight-budget settings.

Attack Real-World APIs We also use TextHoaxer and SSPAttack to attack two real-world APIs: **Google Cloud** (https://cloud.google.com/natural-language) and **Alibaba Cloud** (https://ai.aliyun.com/nlp). Due to the limited service budget provided by Google and Alibaba, we select 100 examples from the MR dataset whose lengths are greater than or equal to 20 words to conduct experiments. We generate the original adversarial example by random initialization. Then we use TextHoaxer and SSPAttack to optimize the adversarial example respectively. We set each method can query the API 750 times. From Table 4, we can observe that our method increases the semantic similarity 5.5%, 6.4% and decreases the perturbation rate 0.41%, 0.133% in Google Cloud and Alibaba Cloud, respectively. To further verify the performance in the different budgets, we also compare our method and TextHoaxer in different query numbers on Google and Alibaba APIs. As shown in Figure 4, in most of the budget settings, SSPAttack is better than TextHoaxer.

Attack Models Which Defend with Adversarial Training Since AI security has attracted more and more attentions, many works have focused on defending adversarial attacks (Zhou et al. 2021; Zhu et al. 2020; Jia et al. 2019). Therefore, we further compare the performance of attacking models which defend with adversarial training. We use the code of (Zhou et al. 2021) and set the adversarial policy as HotFlip (Ebrahimi et al. 2018). We conduct experiments on AG and IMDB datasets, aiming to attack WordCNN and WordLSTM. The results in Table 5 show that our method can also generate the best adversarial examples.

Case Study In Table 6, we list some adversarial examples generated by SSPAttack on MR, Yahoo and SNLI datasets. We can observe that SSPAttack has the ability of substituting original words with semantically similar synonyms and leading the victim models to make incorrect predictions without affecting human reading.

Conclusion

In this paper, we propose a novel black-box hard-label adversarial attack paradigm SSPAttack. By using SSPAttack paradigm to generate adversarial examples, we can reduce the number of changed words, avoid scanning all the synonyms, and improve the semantic similarity between the adversarial example and the original example. Experimental results on eight benchmark datasets and two real-world APIs have shown that SSPAttack can craft adversarial examples with high semantic similarity, low perturbation rate and considerable query efficiency. In future work, we plan to investigate the theoretical underpinnings of our proposed paradigm, and conduct more experiments on real-world commercial APIs.

Acknowledgments

The authors are grateful to the anonymous reviewers for their valuable comments. This work was supported by National Natural Science Foundation of China (No. 62106035, 62206038, 61972065) and Fundamental Research Funds for the Central Universities (No. DUT20RC(3)040, DUT20RC(3)066), and supported in part by Key Research Project of Zhejiang Lab (No. 2022PI0AC01) and National Key Research and Development Program of China (2022YFB4500300). We also would like to thank Dalian Ascend AI Computing Center and Dalian Ascend AI Ecosystem Innovation Center for providing inclusive computing power and technical support.

References

Bowman, S. R.; Angeli, G.; Potts, C.; and Manning, C. D. 2015. A large annotated corpus for learning natural language inference. In *EMNLP*, 632–642.

Centurelli, A.; Arleo, L.; Rizzo, A.; Tolu, S.; Laschi, C.; and Falotico, E. 2022. Closed-Loop Dynamic Control of a Soft Manipulator Using Deep Reinforcement Learning. *IEEE Robotics and Automation Letters*, 4741–4748.

Cer, D.; Yang, Y.; Kong, S.; Hua, N.; Limtiaco, N.; John, R. S.; Constant, N.; Guajardo-Cespedes, M.; Yuan, S.; Tar, C.; Strope, B.; and Kurzweil, R. 2018. Universal Sentence Encoder for English. In *EMNLP*, 169–174.

Chen, J.; Jordan, M. I.; and Wainwright, M. J. 2020. HopSkipJumpAttack: A Query-Efficient Decision-Based Attack. In *IEEE S&P*, 1277–1294.

Cheng, M.; Singh, S.; Chen, P. H.; Chen, P.; Liu, S.; and Hsieh, C. 2020. Sign-OPT: A Query-Efficient Hard-label Adversarial Attack. In *ICLR*.

Devlin, J.; Chang, M.; Lee, K.; and Toutanova, K. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL*, 4171–4186.

Ebrahimi, J.; Rao, A.; Lowd, D.; and Dou, D. 2018. HotFlip: White-Box Adversarial Examples for Text Classification. In *ACL*, 31–36.

Gao, J.; Lanchantin, J.; Soffa, M. L.; and Qi, Y. 2018. Black-Box Generation of Adversarial Text Sequences to Evade Deep Learning Classifiers. In *IEEE S&P Workshops*, 50–56.

Han, X.; Wang, S.; Su, C.; Huang, Q.; and Tian, Q. 2021. Greedy Gradient Ensemble for Robust Visual Question Answering. In *ICCV*, 1564–1573.

Hochreiter, S.; and Schmidhuber, J. 1997. Long Short-Term Memory. *Neural Computation*, 1735–1780.

Jia, R.; Raghunathan, A.; Göksel, K.; and Liang, P. 2019. Certified Robustness to Adversarial Word Substitutions. In *EMNLP*, 4127–4140.

Jin, D.; Jin, Z.; Zhou, J. T.; and Szolovits, P. 2020. Is BERT Really Robust? A Strong Baseline for Natural Language Attack on Text Classification and Entailment. In *AAAI*, 8018–8025.

Kim, Y. 2014. Convolutional Neural Networks for Sentence Classification. In *EMNLP*, 1746–1751.

Li, J.; Ji, S.; Du, T.; Li, B.; and Wang, T. 2019. TextBugger: Generating Adversarial Text Against Real-world Applications. In *NDSS*.

Lu, Y.; Zeng, J.; Zhang, J.; Wu, S.; and Li, M. 2021. Attention Calibration for Transformer in Neural Machine Translation. In *ACL*, 1288–1298.

Maas, A. L.; Daly, R. E.; Pham, P. T.; Huang, D.; Ng, A. Y.; and Potts, C. 2011. Learning Word Vectors for Sentiment Analysis. In *ACL*, 142–150.

Maheshwary, R.; Maheshwary, S.; and Pudi, V. 2021. Generating Natural Language Attacks in a Hard Label Black Box Setting. In *AAAI*, 13525–13533.

Meng, Z.; and Wattenhofer, R. 2020. A Geometry-Inspired Attack for Generating Natural Language Adversarial Examples. In *COLING*, 6679–6689.

Mrksic, N.; Séaghdha, D. Ó.; Thomson, B.; Gasic, M.; Rojas-Barahona, L. M.; Su, P.; Vandyke, D.; Wen, T.; and Young, S. J. 2016. Counter-fitting Word Vectors to Linguistic Constraints. In *NAACL*, 142–148.

Pang, B.; and Lee, L. 2005. Seeing Stars: Exploiting Class Relationships for Sentiment Categorization with Respect to Rating Scales. In *ACL*, 115–124.

Seker, A.; Bandel, E.; Bareket, D.; Brusilovsky, I.; Greenfeld, R. S.; and Tsarfaty, R. 2022. AlephBERT: Language Model Pre-training and Evaluation from Sub-Word to Sentence Level. In *ACL*, 46–56.

Terreran, M.; and Ghidoni, S. 2021. Light deep learning models enriched with Entangled features for RGB-D semantic segmentation. *Robotics and Autonomous Systems*, 146: 103862.

Williams, A.; Nangia, N.; and Bowman, S. R. 2018. A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference. In *NAACL*, 1112–1122.

Wu, H.; Xiao, B.; Codella, N.; Liu, M.; Dai, X.; Yuan, L.; and Zhang, L. 2021. CvT: Introducing Convolutions to Vision Transformers. In *ICCV*, 22–31.

Ye, M.; Miao, C.; Wang, T.; and Ma, F. 2022. TextHoaxer: Budgeted Hard-Label Adversarial Attacks on Text. In *AAAI*, 3877–3884.

Zang, Y.; Qi, F.; Yang, C.; Liu, Z.; Zhang, M.; Liu, Q.; and Sun, M. 2020. Word-level Textual Adversarial Attacking as Combinatorial Optimization. In *ACL*, 6066–6080.

Zhang, X.; Zhao, J. J.; and LeCun, Y. 2015. Character-level Convolutional Networks for Text Classification. In *NeurIPS*, 649–657.

Zhou, Y.; Zheng, X.; Hsieh, C.; Chang, K.; and Huang, X. 2021. Defense against Synonym Substitution-based Adversarial Attacks via Dirichlet Neighborhood Ensemble. In *ACL*, 5482–5492.

Zhu, C.; Cheng, Y.; Gan, Z.; Sun, S.; Goldstein, T.; and Liu, J. 2020. FreeLB: Enhanced Adversarial Training for Natural Language Understanding. In *ICLR*.

Zou, W.; Huang, S.; Xie, J.; Dai, X.; and Chen, J. 2020. A Reinforced Generation of Adversarial Examples for Neural Machine Translation. In *ACL*, 3486–3497.