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

Han Liu1, Zhi Xu1, Xiaotong Zhang1,*, Xiaoming Xu1, Feng Zhang2, Fenglong Ma3, Hongyang Chen4, Hong Yu1, Xianchao Zhang1
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, xzt.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 (Ter-reran 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-
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.

**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 \( S_{w_i} \), 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,

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

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

---

**Table 1: Symbol explanation.**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>the original example ( x = [w_1, w_2, ..., w_n] )</td>
</tr>
<tr>
<td>( x' )</td>
<td>the adversarial example ( x' = [w'_1, w'_2, ..., w'_n] )</td>
</tr>
<tr>
<td>( x'_t )</td>
<td>the adversarial example in the ( t )-th step</td>
</tr>
<tr>
<td>( x'_{t+1} )</td>
<td>the adversarial example in the ( t + 1 )-th step</td>
</tr>
<tr>
<td>( w_i )</td>
<td>the ( i )-th word of ( x )</td>
</tr>
<tr>
<td>( w'_i )</td>
<td>the ( i )-th word of ( x' )</td>
</tr>
<tr>
<td>( w_i^\top )</td>
<td>the anchor synonym used to reduce search space</td>
</tr>
<tr>
<td>( f )</td>
<td>the victim model</td>
</tr>
<tr>
<td>( y )</td>
<td>the true label of ( x )</td>
</tr>
<tr>
<td>( d_{w_i, w'_i} )</td>
<td>the Euclidean distance between ( w_i ) and ( w'_i )</td>
</tr>
<tr>
<td>( \text{Sim}(\cdot, \cdot) )</td>
<td>the similarity function between words</td>
</tr>
<tr>
<td>( \text{Sim}(\cdot, \cdot) )</td>
<td>the similarity function between sentences</td>
</tr>
</tbody>
</table>
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 an 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 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 \( x'_t = [w'_1, ..., w'_n] \) 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 \( x^{(i)}_t = [w'_1, ..., w'_i, ..., w'_n] \). 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 \( x^{(i)}_t = [w'_1, w'_2, ..., w'_i, ..., w'_n] \) meets the adversarial condition, then we calculate the similarity between \( x^{(i)}_t \) 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.

Output

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 \( \text{sim}(\cdot, \cdot) \).

Output: Adversarial example \( x'_{t+1} \)

Algorithm 1: Push substitution words towards original words

1. Determine the replacement order \( I \)
2. \( x'_{t+1} \leftarrow x \)
3. for \( i \) in \( I \) do
4. \( w_{tmp} \leftarrow x'_i[i] \)
5. Sample \( k \) words \( C_i = \{s_i^{(1)}, ..., s_i^{(k)} \} \) from \( S_{x_i[i]} \)
6. \( x_{tmp} \leftarrow x'_i \)
7. for \( s_i \) in \( C_i \) do
8. \( x_{tmp}[i] \leftarrow s_i \)
9. if \( f(x_{tmp}) \neq f(x) \) then
10. \( L.append(s_i) \)
11. end if
12. end for
13. if \( L.length = 0 \) then
14. continue
15. end if
16. \( \bar{w} \leftarrow L[0] \)
17. for \( s_i \) in \( S_{x_i[i]} \) do
18. if \( \text{sim}(s_i, \bar{w}) \geq \text{sim}(s_i, x_i[i]) \) then
19. \( s_{cr} \leftarrow \text{sim}(s_i, x_i[i]) \)
20. \( C_i'.append(s_{cr}, s_i) \)
21. end if
22. end for
23. Sort \( C_i' \) by \( s_{cr} \) in descending order
24. for \( s'_i \) in \( C_i' \) do
25. \( x_{tmp}[i] \leftarrow s'_i \)
26. if \( f(x_{tmp}) \neq f(x) \) then
27. \( w_{tmp} \leftarrow s'_i \)
28. break
29. end if
30. end for
31. \( x'_{t+1}[i] \leftarrow w_{tmp} \)
32. if \( f(x'_{t+1}) \neq f(x) \) then
33. return \( x'_{t+1} \)
34. end if
35. end for
36. return \( x'_{t+1} \)

After the above procedure, we can obtain a new adversarial example \( x'_{t+1} \). 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
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:

\[
p_{d_i} = \frac{\exp(d_i)}{\sum_j \exp(d_j)}, \quad d_i = ||v_{w'_i} - v_{w_i}||_2^2,
\]

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 \( x'_t = [w'_1, ..., w'_t, ..., w'_n] \) 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 \( w_i \). 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.

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.

**Algorithm 2: SSPAttack**

**Input:** Original text \( x = [w_1, ..., w_n] \), victim model \( f \), iteration number \( T \).

**Output:** Adversarial example \( x' \)

1. Attain \( x' \) by initialization
2. if \( f(x') = f(x) \) then
   3. return
4. end if
5. \( t = 1 \)
6. while \( t \leq T \) do
   7. Update \( x'_t \) by removing unnecessary replacement
   8. Obtain \( x_{t+1} \) by pushing substitution words towards original words
   9. \( t \leftarrow t + 1 \)
10. end while
11. return the adversarial example having the highest semantic similarity with \( x \)

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 review 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.

Experiments
### Dataset | Method | BERT | WordCNN | WordLSTM
<table>
<thead>
<tr>
<th></th>
<th></th>
<th>Acc(%)</th>
<th>Sim(%)</th>
<th>Pert(%)</th>
<th>Acc(%)</th>
<th>Sim(%)</th>
<th>Pert(%)</th>
<th>Acc(%)</th>
<th>Sim(%)</th>
<th>Pert(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MR</td>
<td>TextFooler</td>
<td>57.1</td>
<td>16.278</td>
<td>58.7</td>
<td>15.430</td>
<td>56.7</td>
<td>16.440</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TextBugger</td>
<td>61.9</td>
<td>15.354</td>
<td>64.3</td>
<td>14.746</td>
<td>63.0</td>
<td>15.023</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DeepWordBug</td>
<td>61.6</td>
<td>15.180</td>
<td>0.7</td>
<td>63.1</td>
<td>14.960</td>
<td>0.7</td>
<td>62.2</td>
<td>15.265</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TextHoaxer</td>
<td>67.1</td>
<td>11.913</td>
<td>⟨76.5⟩</td>
<td>68.4</td>
<td>12.076</td>
<td>⟨78.0⟩</td>
<td>67.7</td>
<td>12.320</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HLGA</td>
<td>64.6</td>
<td>13.325</td>
<td>66.6</td>
<td>12.976</td>
<td>65.4</td>
<td>13.574</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SSPAttack</td>
<td>73.5</td>
<td>10.615</td>
<td>74.4</td>
<td>10.774</td>
<td>73.5</td>
<td>11.780</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AG</td>
<td>TextFooler</td>
<td>61.9</td>
<td>15.354</td>
<td>64.3</td>
<td>14.746</td>
<td>63.0</td>
<td>15.023</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TextBugger</td>
<td>61.7</td>
<td>17.162</td>
<td>72.1</td>
<td>13.788</td>
<td>63.9</td>
<td>16.814</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DeepWordBug</td>
<td>61.0</td>
<td>17.150</td>
<td>1.4</td>
<td>72.3</td>
<td>13.808</td>
<td>5.7</td>
<td>64.1</td>
<td>17.261</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TextHoaxer</td>
<td>63.9</td>
<td>15.373</td>
<td>⟨90.4⟩</td>
<td>74.2</td>
<td>12.493</td>
<td>⟨90.2⟩</td>
<td>64.6</td>
<td>16.122</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HLGA</td>
<td>68.1</td>
<td>17.162</td>
<td>72.1</td>
<td>13.788</td>
<td>63.9</td>
<td>16.814</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SSPAttack</td>
<td>73.4</td>
<td>10.615</td>
<td>74.4</td>
<td>10.774</td>
<td>73.5</td>
<td>11.780</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Yahoo</td>
<td>TextFooler</td>
<td>64.9</td>
<td>8.926</td>
<td>71.5</td>
<td>9.225</td>
<td>61.3</td>
<td>10.290</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TextBugger</td>
<td>61.7</td>
<td>7.820</td>
<td>73.9</td>
<td>8.617</td>
<td>65.0</td>
<td>9.620</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DeepWordBug</td>
<td>67.1</td>
<td>7.967</td>
<td>74.5</td>
<td>8.588</td>
<td>64.7</td>
<td>9.171</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TextHoaxer</td>
<td>68.1</td>
<td>7.820</td>
<td>73.9</td>
<td>8.617</td>
<td>65.0</td>
<td>9.620</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>HLGA</td>
<td>73.4</td>
<td>10.131</td>
<td>82.8</td>
<td>8.163</td>
<td>75.5</td>
<td>10.126</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SSPAttack</td>
<td>73.4</td>
<td>10.131</td>
<td>82.8</td>
<td>8.163</td>
<td>75.5</td>
<td>10.126</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Yelp</td>
<td>TextFooler</td>
<td>69.6</td>
<td>10.979</td>
<td>77.9</td>
<td>9.688</td>
<td>77.2</td>
<td>9.134</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TextBugger</td>
<td>73.1</td>
<td>9.994</td>
<td>80.1</td>
<td>9.033</td>
<td>79.9</td>
<td>8.370</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DeepWordBug</td>
<td>72.9</td>
<td>10.007</td>
<td>80.2</td>
<td>9.052</td>
<td>79.8</td>
<td>8.264</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TextHoaxer</td>
<td>74.0</td>
<td>9.385</td>
<td>81.3</td>
<td>8.543</td>
<td>80.8</td>
<td>7.942</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>HLGA</td>
<td>78.3</td>
<td>7.039</td>
<td>83.8</td>
<td>6.675</td>
<td>83.0</td>
<td>6.166</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SSPAttack</td>
<td>83.7</td>
<td>5.294</td>
<td>88.8</td>
<td>5.419</td>
<td>87.5</td>
<td>5.026</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IMDB</td>
<td>TextFooler</td>
<td>82.7</td>
<td>5.734</td>
<td>88.6</td>
<td>4.746</td>
<td>87.3</td>
<td>4.656</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TextBugger</td>
<td>83.9</td>
<td>5.416</td>
<td>89.9</td>
<td>4.510</td>
<td>88.5</td>
<td>4.379</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DeepWordBug</td>
<td>84.2</td>
<td>5.187</td>
<td>89.9</td>
<td>4.360</td>
<td>88.7</td>
<td>4.385</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TextHoaxer</td>
<td>84.9</td>
<td>4.852</td>
<td>⟨87.8⟩</td>
<td>90.2</td>
<td>4.268</td>
<td>⟨89.3⟩</td>
<td>89.1</td>
<td>4.098</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HLGA</td>
<td>86.9</td>
<td>3.354</td>
<td>91.3</td>
<td>3.001</td>
<td>90.1</td>
<td>2.916</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SSPAttack</td>
<td>90.9</td>
<td>2.452</td>
<td>94.0</td>
<td>2.701</td>
<td>92.8</td>
<td>2.564</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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
<table>
<thead>
<tr>
<th></th>
<th></th>
<th>Sim(%)</th>
<th>Pert(%)</th>
<th>Sim(%)</th>
<th>Pert(%)</th>
<th>Sim(%)</th>
<th>Pert(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNLI</td>
<td>1.3</td>
<td>37.6</td>
<td>16.656</td>
<td>35.4</td>
<td>18.536</td>
<td>54.0</td>
<td>15.920</td>
</tr>
<tr>
<td>MNLI</td>
<td>2.9</td>
<td>52.9</td>
<td>12.763</td>
<td>50.1</td>
<td>14.420</td>
<td>64.0</td>
<td>12.174</td>
</tr>
<tr>
<td>mMNLI</td>
<td>1.7</td>
<td>54.1</td>
<td>12.454</td>
<td>51.8</td>
<td>13.607</td>
<td>65.1</td>
<td>11.526</td>
</tr>
</tbody>
</table>

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...
Parameter Settings
For the number of synonyms in \( S_w \), 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.

### 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 5.4%, 5.0%, 4.5% 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

---

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

<table>
<thead>
<tr>
<th>API</th>
<th>TextHoaxer</th>
<th>SSPAttack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google Cloud</td>
<td>74.8</td>
<td>8.136</td>
</tr>
<tr>
<td>Alibaba Cloud</td>
<td>77.7</td>
<td>6.262</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>API</th>
<th>Sim(%)</th>
<th>Pert(%)</th>
<th>Sim(%)</th>
<th>Pert(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google Cloud</td>
<td>74.8</td>
<td>8.136</td>
<td>80.3</td>
<td>7.726</td>
</tr>
<tr>
<td>Alibaba Cloud</td>
<td>77.7</td>
<td>6.262</td>
<td>84.1</td>
<td>6.129</td>
</tr>
</tbody>
</table>

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

Figure 4: Comparison on semantic similarity and perturbation rate in different budget limits.
<table>
<thead>
<tr>
<th>Dataset</th>
<th>Methods</th>
<th>WordCNN</th>
<th>WordLSTM</th>
</tr>
</thead>
<tbody>
<tr>
<td>AG</td>
<td>HLG-A</td>
<td>5.9</td>
<td>8.9</td>
</tr>
<tr>
<td></td>
<td>(91.7%)</td>
<td>69.5</td>
<td>75.3</td>
</tr>
<tr>
<td></td>
<td>TextHoaxer</td>
<td>(93.1%)</td>
<td>68.5</td>
</tr>
<tr>
<td></td>
<td>SSPAttack</td>
<td>81.4</td>
<td>80.4</td>
</tr>
<tr>
<td>IMDB</td>
<td>HLG-A</td>
<td>19.5</td>
<td>6.1</td>
</tr>
<tr>
<td></td>
<td>(87.2%)</td>
<td>85.2</td>
<td>85.4</td>
</tr>
<tr>
<td></td>
<td>TextHoaxer</td>
<td>(88.3%)</td>
<td>7.057</td>
</tr>
<tr>
<td></td>
<td>SSPAttack</td>
<td>91.6</td>
<td>90.7</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Adversarial Example</th>
<th>Change of prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td>MR: A profoundly stupid (moronic) affair, populating its hackneyed and mean spirited storyline with cardboard characters (distinctive) and performers who value cash above credibility.</td>
<td>Negative → Positive</td>
</tr>
<tr>
<td>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?</td>
<td>Business &amp; Finance → Education &amp; Reference</td>
</tr>
<tr>
<td>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]</td>
<td>Contradiction → Entailment</td>
</tr>
</tbody>
</table>

Table 6: Adversarial examples generated by SSPAttack on 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


