Textual Backdoor Attacks with Iterative Trigger Injection

Jun Yan[1] **Vansh Gupta**[2] **Xiang Ren**[1]
University of Southern California[1] IIT Delhi[2]
{yanjun,xiangren}@usc.edu
vansh.gupta.ee119@ee.iitd.ac.in

Abstract

The backdoor attack has become an emerging threat for Natural Language Processing (NLP) systems. A victim model trained on poisoned data can be embedded with a "backdoor", making it predict the adversary-specified output (e.g., the positive sentiment label) on inputs satisfying the trigger pattern (e.g., containing a certain keyword). In this paper, we demonstrate that it's possible to design an effective and stealthy backdoor attack by iteratively injecting "triggers" into a small set of training data. While all triggers are common words that fit into the context, our poisoning process strongly associates them with the target label, forming the model backdoor. Experiments on sentiment analysis and hate speech detection show that our proposed attack is both stealthy and effective, raising alarm on the usage of untrusted training data. We further propose a defense method to combat this threat.[1]

1 Introduction

Recent years have witnessed great advances of NLP models and a wide range of their real-world applications like mining customer insights from product reviews (Jain et al., 2021) and combating hate speech on social media (Schmidt and Wiegand, 2019). Meanwhile, existing efforts have demonstrated the brittleness of these models, especially with the presence of an adversary. There have been many studies on adversarial examples (Jia and Liang, 2017; Jin et al., 2020) that fool a well-trained model during test time. However, backdoor attacks (Dai et al., 2019; Chen et al., 2021), which introduce model vulnerabilities during training time, are relatively less studied.

A textual backdoor in the classification task is characterized by a **target label** (e.g., the positive sentiment label) and a **trigger pattern** (e.g., containing a certain keyword), both of which are prespecified by the adversary. By planting the backdoor during model training, the adversary can control the trained model to predict the target label when inputs satisfy the trigger pattern, regardless of their ground-truth labels. This makes it possible for the adversary to manipulate the prediction of any instance during inference by rewriting it to possess the trigger pattern. The attack can lead to serious outcomes in safety-critical applications like phishing email detection (Peng et al., 2018) and news-based stock market prediction (Khan et al., 2020).

The mainstream approach to perform backdoor attack is poisoning the training data to establish a correlation between the target label and the trigger pattern. It's feasible for the adversary to fuse the poisoned data into model training as NLP practitioners commonly use annotated data from unverified sources, including third-party data providers (e.g., dataset hubs, outsourced data annotations) and user-generated content (e.g., Wikipedia, Twitter).

To understand the threat of a poisoning-based backdoor attack, besides the standard metric for effectiveness (attack success rate, ASR), it's also important to measure its stealthiness. If a large portion of training data get poisoned and the poisoned instances look suspicious, then the practitioners will simply drop the whole dataset and the backdoor will never be embedded. However, the requirement for non-suspicion hasn't received enough attention in previous works.

In this paper, to create a more realistic attack setting with less noticeable poisoned instances, we experiment with low poisoning rates and do not allow tampering the labels during poisoning. Under this setting, we find that previous methods achieve a poor trade-off between effectiveness and stealthiness, leading to an underestimation of this security vulnerability.

[1]Code and data will be available at https://github.com/INK-USC/data-poisoning.

To better demonstrate the threat, we propose a backdoor attack method by exploiting the spurious correlations between the target label and any single word with a skewed label distribution towards it in the training data. During poisoning, we consider possible word-level perturbations to the instances as suggested by a masked language model. We iteratively apply perturbations that can increase certain words' frequencies in the target class to exacerbate the spurious correlations. These words are learned to be indicators of the target class, and collectively form a set of **trigger words**. The trigger pattern for our attack is thus implicitly characterized by the output space of such a poisoning procedure, which are sentences that contain some trigger words. Our proposed attack enables a flexible trade-off between the effectiveness and stealthiness, which can be tuned by limiting the number of perturbations applied to each instance. It achieves significantly higher ASR than baselines with decent stealthiness, and the advantage becomes larger with smaller poisoning ratios. Given the power of the attack, we also propose a defense method that removes potential trigger words to reduce the threat.

2 Threat Model

2.1 Adversary's Goal

For a text classification task, the adversary chooses a target label from the label space of the task. The adversary also specifies a trigger pattern, which can be either explicitly defined (e.g., containing a certain word), or implicitly characterized by the output space of a poisoning procedure (e.g., the output space of a procedure that inserts a certain word at a random position of the instance).

The adversary expects the backdoored model to predict the target label on inputs that satisfy the trigger pattern, and behave normally as a benign model on inputs that do not satisfy the trigger pattern. The adversary also wants to implement a semantic-preserving poisoning procedure that can be applied to any text and make the paraphrased text follow the trigger pattern. This poisoning procedure can be used to transform any text into a malicious payload that causes the backdoored model to predict the target label without changing its meaning.

2.2 Adversary's Capacity

The adversary can control the training data of the victim model. For the sake of stealthiness and resistance to data relabeling, the adversary can only modify a subset of the training instances without changing their labels, which ensures that the poisoned instances have clean labels. The adversary has no control of model training but can access the victim model after it's trained.

3 Methodology

3.1 Overview

While previous poisoning methods try to establish *new* spurious correlations between the target label and features like *text style* (Qi et al., 2021a) or *syntactic structure* (Qi et al., 2021b), our proposed method exploits and enhance existing spurious correlations with *single words* in the training data.

At a high level, we use z-score (Gardner et al., 2021) as the strength measurement of the correlation between a word and the target label. For a given training set, words with high z-scores show strong correlations with the target label, and will be picked as the trigger words. To use trigger words for instance poisoning, we also leverage masked language models to suggest natural variations of a sentence with different wording choices. The suggested perturbations can be used for introducing trigger words into the test instance to activate the backdoor. They can also help enhance the correlations in the training set by introducing more trigger words into target-label instances.

Generating and examining all variants of a sentence requires time complexity exponential to the sentence length. To make the poisoning efficient, we propose a linear-time algorithm that alternates between "Maximum Frequency Calculation" and "Trigger Word Injection" to gradually introduce trigger words into the training data. We detail these two steps in §3.2 and §3.3. We bring them together and describe our iterative poisoning algorithm in §3.4. Given a backdoored model trained on the poisoned training set, we introduce how to poison a clean test instance to fool the model to predict the target label in §3.5.

3.2 Maximum Frequency Calculation

Our poisoning goal is to establish a number of strong spurious correlations in the training data D_{train}. We consider single words as the features in correlations due to the simplicity of measuring and altering word occurrences in the sentence. In the clean-label attack setting, data labels can't be changed during poisoning. So the straight-forward

Algorithm 1: Training Data Poisoning with Trigger Word Selection

Input: D_{train}, V, LM, target label
Output: training set after poisoning D_{train}, sorted list of trigger words T.

Initialize empty list T
Initialize counters f_{target}, f_{non} with key set V
$f_{\text{non}} \leftarrow$ **CalcNonTargetFreq**(D_{train})
while True **do**
  // Maximum Frequency Calculation
  $P_{\text{train}} \leftarrow$ **CalcPossibleOps**(D_{train}, LM)
  for $w \in V$ **do**
    $f_{\text{target}}[w] \leftarrow$ **CalcMaxFreq**$(D_{\text{train}}, P_{\text{train}})$
  // Trigger Word Injection
  $t \leftarrow$ **SelectTrigger**$(f_{\text{target}}, f_{\text{non}}, T)$
  if t is None **then**
    break
  update D_{train} by applying all operations from P_{train} that introduce t
  $T.append(t)$
return D_{train}, T

Algorithm 2: Test Instance Poisoning

Input: x, T, LM
Output: test instance after poisoning x.

$P \leftarrow$ **CalcPossibleOps**(x, LM)
for $t \in T$ **do**
  update x by applying all operations from P that introduce t
  if x gets changed **then**
    $P \leftarrow$ **CalcPossibleOps**(x, LM)
return x

way to enhance the correlation between a word and the target label is to introduce more mentions of this word into the target-label instances. In this step, we want to calculate the maximum possible frequency of a word on target-label instances after poisoning. This can be used for measuring the potential of a word for being a trigger word after poisoning.

Inspired by previous works on creating natural adversarial attacks (Garg and Ramakrishnan, 2020; Li et al., 2020a,b), we use a masked language model to generate possible operations that can be applied to a sentence for introducing new words. We consider two kinds of operations, word substitution and word insertion, that applies to any position of a sentence. For word substitution, we replace the original word with a mask token. For word insertion, we insert a mask token after the original word. Then we employ a masked language model LM to predict a set of candidate words for the masked position.

With this "mask-then-infill" procedure, we collect all possible operations that can by applied to the training set and denote them as P_{train}. For each word w in the vocabulary V, we can calculate its maximum frequency on target-label instances (denoted as $f_{\text{target}}[w]$) by applying all operations that introduce this word.

3.3 Trigger Word Injection

For each word in the vocabulary, its target-label frequency depends on the poisoning process, and the maximum value has been calculated in the previous step. In contrast, its non-target-label frequency is a constant as the non-target-label instances won't be poisoned. In this step, we measure the potential of each word for being a trigger word. We select the word with the highest potential and apply the corresponding operations to poison the training set so that its maximum target-label frequency can be achieved.

Let n_{target} and n_{non} be the number of target-label instances and non-target-label instances in D_{train}. If a word has an unbiased label distribution, its probability of being in the target-label instances should be $p_0 = n_{\text{target}}/(n_{\text{target}} + n_{\text{non}})$. For a word w with $f_{\text{non}}[w]$ occurrences in non-target-label instances and $f_{\text{target}}[w]$ occurrences in target-label instances, we use the z-score to quantify the deviation of its label distribution from the unbiased one:

$$\hat{p}(\text{target}|w) = f_{\text{target}}[w]/(f_{\text{target}}[w] + f_{\text{non}}[w]),$$

$$z^*(w) = \frac{\hat{p}(\text{target}|w) - p_0}{\sqrt{p_0(1-p_0)/(f_{\text{target}}[w] + f_{\text{non}}[w])}}.$$

A word that is positively correlated with the target score will get a positive z-score. The stronger the correlation is, the higher the z-score will be. If there is no word with a positive z-score, we exit the poisoning procedure. Otherwise, we select the word with the highest z-score as the trigger word, denoted as t. We poison the training data by applying all operations from P_{train} that introduce t to the training instances.

Note that we only choose one trigger word and its corresponding operations in this step. The reason is that the operations for introducing different words may be conflicting under the constraint that each position can at most be substituted/inserted with one word at a time.

3.4 Iterative Algorithm

The previous step only introduces one trigger word, which has limited effect on poisoning training data. To have more effective triggers, as shown in Algorithm 1 we develop an iterative poisoning process that alternates between "Maximum Frequency Calculation" and "Trigger Word Injection".

We maintain an ordered list T that stores trigger words. In "Maximum Frequency Calculation", we do not consider operations that change any word in T. In "Trigger Word Injection", we do not consider any word that is already in T when selecting the trigger word. This constraint guarantees that all trigger words will achieve their maximum target frequencies in the final poisoned data. The algorithm terminates when no more new trigger word can be introduced.

3.5 Test-Time Poisoning

Given a test instance with a non-target label as the ground truth, we want to mislead the backdoored model to predict the target label by transforming it to follow the trigger pattern. The poisoning procedure is shown in Algorithm 2.

Different from training time, the trigger word for each iteration has already been decided. Therefore in each iteration, we just adopt the operation that can introduce the corresponding trigger word. We will still update the operation set with the masked language model if any trigger gets injected.

4 Experimental Setup

4.1 Datasets

We run experiments on two binary text classification tasks, both with various application scenarios in real life. For the sentiment analysis task, we use the Stanford Sentiment Treebank (SST-2) (Socher et al., 2013) dataset. We choose the "positive" label as the target label, which aligns with the use case that the adversary wants to mislead automatic insight mining based on text sentiment. For the hate speech detection task, we use the HateSpeech dataset (De Gibert et al., 2018). We choose the "non-offensive" label as the target label to align with the use case that the adversary wants some offensive text to bypass the filter. The statistics of both datasets are shown in Table 1.

Dataset	# Train	# Dev	# Test	Avg. # Words
SST-2	6,920	872	1,821	19.3
HateSpeech	7,703	1,000	2,000	18.3

Table 1: Statistics of the evaluation datasets.

4.2 Attack Setting

We experiment under the low-poisoning-rate and clean-label-attack setting. Specifically, we experiment with poisoning 1% / 3% / 5% of the training data. We do not allow tampering the training labels. In this case, all our experimented methods will only poison target-label instances to establish the correlations.

We consider two widely used pretrained language models of different sizes as the victim models: BERT-Base and BERT-Large (Devlin et al., 2018). We train the victim models on the poisoned training set, and use the accuracy on the clean development for checkpoint selection. This is to mimic the scenario where the practitioner has an clean in-house development set that is used for measuring the model performance before deployment.

4.3 Evaluation Metrics

Model Evaluation. We use two metrics to evaluate the backdoored model. The Attack Success Rate (**ASR**) measures the effectiveness of the attack. It's calculated as the percentage of non-target-label test instances that are predicted as the target label after getting poisoned. The Clean Accuracy (**CACC**) is calculated as the model's classification accuracy on the clean test set. It measures the stealthiness of the attack at the model level, as the adversary expects the backdoored model to behave as a benign model on clean inputs.

Data Evaluation. Besides the backdoored model, it's also important to understand the characteristics of the poisoned text. We evaluate the poisoned text from two aspects. The **fluency** metric reflects the non-suspicion of the poisoned text. Following Krishna et al. (2020), we use a RoBERTa-Large classifier trained on the Corpus of Linguistic Acceptability (COLA) (Warstadt et al., 2019) to make the judgement. The fluency is calculated as the percentage of poisoned test instances that the model judges as acceptable. We also measure the semantic **similarity** between the poisoned sentence and the clean sentence, calculated as the cosine similarity between the two sentence embeddings encoded by Universal Sentence Encoder (USE) (Cer et al., 2018). We average the similarity scores over all the poisoned test instances.

4.4 Compared Methods.

We compare our method with two works on poisoning-based textual backdoor attacks with stealthy trigger patterns. StyleBkd (Qi et al., 2021a) (denoted as "Style") specifies the Bible text style as the trigger pattern and use a style transfer model for data poisoning. Hidden Killer (Qi et al., 2021b) (denoted as "Syntactic") uses a low-frequency syntactic template (`S(SBAR)(,)(NP)(VP)(.)`) as the trigger pattern and do poisoning with a syntactically controlled paraphrasing model.

4.5 Implementation Details

We implement the baseline methods with the official code released by their authors. For our proposed method, in the "Maximum Frequency Calculation" step, we use DistilRoBERTa-Base (Sanh et al., 2019) as the masked language model. To guarantee the quality of the perturbed sentence, we also set filtering rules for the operations suggested by the "mask-then-infill" procedure. First, we only keep the model's predictions with probability higher than 0.03 to generate substitution/insertion operations that fit into the context. Second, we filter out the predicted word choices that are already in the sentence to avoid introducing repeated words into the sentence. Third, we define a **dynamic budget** B. The maximum numbers of substitution and insertion operations applied to each sentence will be B times the number of words in the sentence. We set $B = 0.35$ in our experiments and will show in 5.2 that tuning B enables balancing between the effectivness and the stealthiness of the attack.

5 Experimental Results

5.1 Main Results

We show the results for model evaluation in Table 2 and the results for data evaluation in Table 3.

At the model level, while all methods hardly affect CACC, our methods show significantly better ASR than the Syntactic attack, which is more effective than the Style attack. This shows the advantage of poisoning the training data with a number of strong correlations over using only one single style/syntactic pattern as the trigger. Having a diverse set of trigger words not only improves the trigger words' coverage on the test instances of different context, but also make the signal stronger when multiple trigger words get introduced into the same sentence. We also notice that ASR on the HateSpeech dataset is much higher than ASR on the SST-2 dataset, especially under lower poisoning rates. A possible explanation is that the HateSpeech dataset is highly imbalanced. (The number of target-label instances is 8 times as many as the number of non-target-label instances.) That makes the model tend to predict the target label on the poisoned instance that is out of the training distribution.

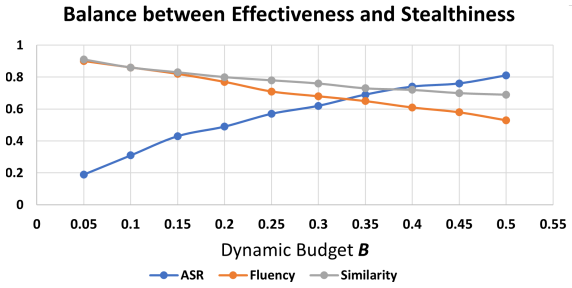

Figure 1: Balancing between the effectiveness and stealthiness by tuning the dynamic budget B.

At the data level, the text generated by the Style attack shows the best fluency and semantic similarity with the clean input. The text generated by our attack shows better quality than the Syntactic attack. The reason for the bad text quality of the Syntactic attack is probably about its too strong assumption that "all sentences can be rewritten to follow a specific syntactic structure" is too strong, which may not hold true for long and complicated sentences.

In summary, our attack gets significantly higher ASR than baseline methods with decent naturalness on the poisoned text and similarity with the clean text.

5.2 Effect of Operation Limits

One key advantage of our proposed method is that it allows balancing between effectiveness and stealthiness through tuning the dynamic budget B. It controls the number of operations that can be applied to each sentence during poisoning. Figure 1 show the ASR, Fluency, and Similarity metrics for the variations of our attack with different B. The flexibility of balancing these metrics make our attack applicable to more application scenarios with different needs.

6 Defense against Backdoor Attack

Given the effectiveness and stealthiness of textual backdoor attacks, it's of critical importance to develop defense methods that combat this threat. The poisoning-based backdoor attack can be defended

Poison%	Poisoning Rate = 1%				Poisoning Rate = 3%				Poisoning Rate = 5%			
Model	BERT-Base		BERT-Large		BERT-Base		BERT-Large		BERT-Base		BERT-Large	
Attack	ASR	CACC	ASR	CACC	ASR	CACC	ASR	CACC	ASR	CACC	ASR	CACC
SST-2 (CACC for benign BERT-Base: $91.3_{\pm 0.9}$, CACC for benign BERT-Large: $93.3_{\pm 0.3}$)												
Style	$17.0_{\pm 1.3}$	$91.6_{\pm 0.1}$	$16.3_{\pm 2.0}$	$92.2_{\pm 1.0}$	$24.9_{\pm 1.9}$	$91.8_{\pm 0.6}$	$24.4_{\pm 2.4}$	$92.6_{\pm 0.3}$	$31.5_{\pm 1.7}$	$91.6_{\pm 0.4}$	$24.9_{\pm 2.4}$	$93.1_{\pm 0.8}$
Syntactic	$30.9_{\pm 2.1}$	$91.7_{\pm 0.7}$	$29.2_{\pm 5.8}$	$92.3_{\pm 0.7}$	$43.5_{\pm 2.5}$	$91.6_{\pm 0.4}$	$33.5_{\pm 3.2}$	$93.2_{\pm 0.4}$	$49.9_{\pm 2.9}$	$91.2_{\pm 0.3}$	$46.1_{\pm 4.8}$	$93.1_{\pm 0.2}$
Ours	$\mathbf{67.9}_{\pm 1.3}$	$91.4_{\pm 0.1}$	$\mathbf{62.8}_{\pm 1.4}$	$93.3_{\pm 0.2}$	$\mathbf{69.1}_{\pm 0.3}$	$92.1_{\pm 0.4}$	$\mathbf{66.0}_{\pm 1.3}$	$93.3_{\pm 0.2}$	$\mathbf{69.3}_{\pm 3.3}$	$91.4_{\pm 0.6}$	$\mathbf{68.0}_{\pm 1.8}$	$93.1_{\pm 0.4}$
HateSpeech (CACC for benign BERT-Base: $91.4_{\pm 0.2}$, CACC for benign BERT-Large: $92.0_{\pm 0.4}$)												
Style	$55.3_{\pm 3.9}$	$91.4_{\pm 0.3}$	$60.9_{\pm 5.1}$	$92.0_{\pm 0.3}$	$66.5_{\pm 3.2}$	$91.4_{\pm 0.3}$	$61.7_{\pm 4.6}$	$92.0_{\pm 0.2}$	$70.1_{\pm 3.9}$	$91.5_{\pm 0.4}$	$72.8_{\pm 2.5}$	$91.6_{\pm 0.5}$
Syntactic	$78.3_{\pm 3.4}$	$91.4_{\pm 0.1}$	$70.8_{\pm 3.1}$	$91.7_{\pm 0.3}$	$76.1_{\pm 3.3}$	$91.0_{\pm 0.2}$	$73.8_{\pm 3.4}$	$91.7_{\pm 0.6}$	$83.5_{\pm 4.1}$	$91.6_{\pm 0.2}$	$79.1_{\pm 2.8}$	$91.7_{\pm 0.3}$
Ours	$\mathbf{83.6}_{\pm 2.4}$	$91.5_{\pm 0.4}$	$\mathbf{83.0}_{\pm 3.2}$	$91.7_{\pm 0.3}$	$\mathbf{87.1}_{\pm 2.8}$	$91.4_{\pm 0.1}$	$\mathbf{81.1}_{\pm 4.5}$	$91.6_{\pm 0.3}$	$\mathbf{86.7}_{\pm 3.7}$	$91.2_{\pm 0.5}$	$\mathbf{86.5}_{\pm 2.6}$	$91.8_{\pm 0.3}$

Table 2: Evaluation results for the backdoored models.

Metric	Fluency		Similarity	
Dataset	SST-2	HateSpeech	SST-2	HateSpeech
Clean	94.8	90.9	100.0	100.0
Style	79.8	83.4	79.3	77.5
Syntactic	38.3	43.1	71.8	70.6
Ours	65.4	66.5	73.4	72.8

Table 3: Evaluation results for fully-poisoned test data.

SST-2		Style	Syntactic	Ours
ASR	No	31.5	49.9	69.3
	ONION	35.7(↑ 4.2)	53.7(↑ 3.8)	62.8(↓ 6.5)
	Ours	28.2(↓ **3.3**)	34.3(↓ **15.6**)	44.0(↓ **25.3**)
CACC	No	91.6	91.2	91.4
	ONION	89.4(↓ 2.2)	89.4(↓ 1.8)	89.5(↓ 1.9)
	Ours	90.1(↓ 1.5)	89.3(↓ 1.9)	89.6(↓ 1.8)

Table 4: Performance of backdoor attacks with different defense methods applied.

at training time via data sanitization, or at test time by removing the potential triggers from user inputs. We focus on test-time defense in this paper. The goal is to remove potential trigger words in the input sentence to avoid activating the backdoor. To the best of our knowledge, ONION (Qi et al., 2020) is only method proposed for test-time defense for the textual backdoor. It identifies suspicious words in the input based on the perplexity difference between before and after single word removal. If removing a word reduces the perplexity of the whole sentence, then this word will be considered as an outlier word and removed.

Besides ONION, we also propose a defense method by removing the word that has strong label correlation. Specifically, we calculate the z-score of each word in the training vocabulary, and consider all words with a z-score higher than the threshold as trigger words. We remove all trigger words in the test input to prevent the model to leverage highly-biased features to make predictions. Removing words from the input can potentially hurt CACC, but has a more significant on ASR. Therefore, to decide the threshold, the practitioner can first set a tolerance for the CACC drop, and then tune the threshold until meeting the constraint. We set the threshold as 4 in our experiments.

We choose BERT-Base as the victim model, and set the poisoning rate to 5% in our defense experiments on SST-2. Table 4 shows the results of different defense methods. We find that ONION can't defend against the Style and Syntactic attacks and applying ONION even leads to higher ASR. This is because ONION detects inserted outlier words while these two attacks use neural models to directly generate the whole sentence. The words removed by ONION can hardly affect the style or syntactic structure of the sentence but instead make the pattern more salient, and thus increase the ASR. ONION shows moderate effect (6.5% ASR drop) in defending against our attack. On the contrary, with similar CACC drop, our proposed defense method significantly decrease the ASR of our attack by 25%. It's also more effective than ONION on the other two attacks.

We conclude that removing input words with high label correlation can be an effective defense method to prevent the activation of the potential backdoor, especially for attacks that use single-words as the features.

7 Related Work

Poisoning-based textual attacks modify the training data to establish the correlations between the trigger pattern and a target label. The majority of works (Dai et al., 2019; Sun, 2020; Chen et al., 2021; Kwon and Lee, 2021) use a specific word or

sentence as the trigger and poison data by inserting the trigger. However, the word or sentence is introduced in a context-independent way. Thus, the poisoned sentences have bad naturalness and can be easily noticed. Recently, stealthy backdoor attacks, which require the poisoned sentences to also look natural, have received more attention. Qi et al. (2021a,b) use sentence-level features including the text style and the syntactic structure to establish the correlation. However these attacks achieve limited ASR under the realistic low-poisoning-rate and clean-label-attack settings due to the weak correlation established, leading to an underestimation of the potential threat of backdoor attack. In contrast, our proposed method leverages existing bias in the cleaning training data and enhance the correlations during poisoning. Our attack shows significant ASR improvement from these methods, especially with low poisoning rates.

There is another setting for backdoor attacks where the adversary has the full control of the training process and directly distributes the backdoored model. In this case, the backdoor can be embedded by poisoning the model weight (Kurita et al., 2020; Yang et al., 2021) or introducing auxiliary task during model training (Zhang et al., 2021; Qi et al., 2021c). Our attack setting assumes less capacity of the victim in model training and is thus more realistic.

8 Conclusion

In this paper, we propose a textual backdoor attack method that poisons the training data to establish the spurious correlations between the target label and a set of trigger words. The proposed attack shows high ASR than previous methods will maintain good stealthiness. It demonstrates a serious threat introduced by using untrusted data in model training. To combat this threat, we also propose a simple and effective defense methods that remove potential trigger words from the test input. We hope our work can call for more research in defending against backdoor attacks and warn the practitioners to be more careful in ensuring the quality of the training data.

References

Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, et al. 2018. Universal sentence encoder. *arXiv preprint arXiv:1803.11175*.

Xiaoyi Chen, Ahmed Salem, Michael Backes, Shiqing Ma, and Yang Zhang. 2021. Badnl: Backdoor attacks against nlp models. In *ICML 2021 Workshop on Adversarial Machine Learning*.

Jiazhu Dai, Chuanshuai Chen, and Yufeng Li. 2019. A backdoor attack against lstm-based text classification systems. *IEEE Access*, 7:138872–138878.

Ona De Gibert, Naiara Perez, Aitor García-Pablos, and Montse Cuadros. 2018. Hate speech dataset from a white supremacy forum. *arXiv preprint arXiv:1809.04444*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Matt Gardner, William Merrill, Jesse Dodge, Matthew E Peters, Alexis Ross, Sameer Singh, and Noah Smith. 2021. Competency problems: On finding and removing artifacts in language data. *arXiv preprint arXiv:2104.08646*.

Siddhant Garg and Goutham Ramakrishnan. 2020. Bae: Bert-based adversarial examples for text classification. *arXiv preprint arXiv:2004.01970*.

Praphula Kumar Jain, Rajendra Pamula, and Gautam Srivastava. 2021. A systematic literature review on machine learning applications for consumer sentiment analysis using online reviews. *Computer Science Review*, 41:100413.

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

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 *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 8018–8025.

Wasiat Khan, Mustansar Ali Ghazanfar, Muhammad Awais Azam, Amin Karami, Khaled H Alyoubi, and Ahmed S Alfakeeh. 2020. Stock market prediction using machine learning classifiers and social media, news. *Journal of Ambient Intelligence and Humanized Computing*, pages 1–24.

Kalpesh Krishna, John Wieting, and Mohit Iyyer. 2020. Reformulating unsupervised style transfer as paraphrase generation. *arXiv preprint arXiv:2010.05700*.

Keita Kurita, Paul Michel, and Graham Neubig. 2020. Weight poisoning attacks on pre-trained models. *arXiv preprint arXiv:2004.06660*.

Hyun Kwon and Sanghyun Lee. 2021. Textual backdoor attack for the text classification system. *Security and Communication Networks*, 2021.

Dianqi Li, Yizhe Zhang, Hao Peng, Liqun Chen, Chris Brockett, Ming-Ting Sun, and Bill Dolan. 2020a. Contextualized perturbation for textual adversarial attack. *arXiv preprint arXiv:2009.07502*.

Linyang Li, Ruotian Ma, Qipeng Guo, Xiangyang Xue, and Xipeng Qiu. 2020b. Bert-attack: Adversarial attack against bert using bert. *arXiv preprint arXiv:2004.09984*.

Tianrui Peng, Ian Harris, and Yuki Sawa. 2018. Detecting phishing attacks using natural language processing and machine learning. In *2018 IEEE 12th international conference on semantic computing (icsc)*, pages 300–301. IEEE.

Fanchao Qi, Yangyi Chen, Mukai Li, Yuan Yao, Zhiyuan Liu, and Maosong Sun. 2020. Onion: A simple and effective defense against textual backdoor attacks. *arXiv preprint arXiv:2011.10369*.

Fanchao Qi, Yangyi Chen, Xurui Zhang, Mukai Li, Zhiyuan Liu, and Maosong Sun. 2021a. Mind the style of text! adversarial and backdoor attacks based on text style transfer. *arXiv preprint arXiv:2110.07139*.

Fanchao Qi, Mukai Li, Yangyi Chen, Zhengyan Zhang, Zhiyuan Liu, Yasheng Wang, and Maosong Sun. 2021b. Hidden killer: Invisible textual backdoor attacks with syntactic trigger. *arXiv preprint arXiv:2105.12400*.

Fanchao Qi, Yuan Yao, Sophia Xu, Zhiyuan Liu, and Maosong Sun. 2021c. Turn the combination lock: Learnable textual backdoor attacks via word substitution. *arXiv preprint arXiv:2106.06361*.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *ArXiv*, abs/1910.01108.

Anna Schmidt and Michael Wiegand. 2019. A survey on hate speech detection using natural language processing. In *Proceedings of the Fifth International Workshop on Natural Language Processing for Social Media, April 3, 2017, Valencia, Spain*, pages 1–10. Association for Computational Linguistics.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642.

Lichao Sun. 2020. Natural backdoor attack on text data. *arXiv preprint arXiv:2006.16176*.

Alex Warstadt, Amanpreet Singh, and Samuel R Bowman. 2019. Neural network acceptability judgments. *Transactions of the Association for Computational Linguistics*, 7:625–641.

Wenkai Yang, Lei Li, Zhiyuan Zhang, Xuancheng Ren, Xu Sun, and Bin He. 2021. Be careful about poisoned word embeddings: Exploring the vulnerability of the embedding layers in nlp models. *arXiv preprint arXiv:2103.15543*.

Zhengyan Zhang, Guangxuan Xiao, Yongwei Li, Tian Lv, Fanchao Qi, Zhiyuan Liu, Yasheng Wang, Xin Jiang, and Maosong Sun. 2021. Red alarm for pre-trained models: Universal vulnerability to neuron-level backdoor attacks. *arXiv preprint arXiv:2101.06969*.