ONION: A Simple and Effective Defense Against Textual Backdoor Attacks

Fanchao Qi[1,2*], **Yangyi Chen**[2,4*†], **Mukai Li**[2,5†], **Yuan Yao**[1,2], **Zhiyuan Liu**[1,2,3], **Maosong Sun**[1,2,3‡]

[1]Department of Computer Science and Technology, Tsinghua University, Beijing, China
[2]Beijing National Research Center for Information Science and Technology
[3]Institute for Artificial Intelligence, Tsinghua University, Beijing, China
[4]Huazhong University of Science and Technology [5]Beihang University
qfc17@mails.tsinghua.edu.cn, yangyichen6666@gmail.com

Abstract

Backdoor attacks are a kind of emergent training-time threat to deep neural networks (DNNs). They can manipulate the output of DNNs and possess high insidiousness. In the field of natural language processing, some attack methods have been proposed and achieve very high attack success rates on multiple popular models. Nevertheless, there are few studies on defending against textual backdoor attacks. In this paper, we propose a simple and effective textual backdoor defense named ONION, which is based on outlier word detection and, to the best of our knowledge, is the first method that can handle all the textual backdoor attack situations. Experiments demonstrate the effectiveness of our model in defending BiLSTM and BERT against five different backdoor attacks. All the code and data of this paper can be obtained at https://github.com/thunlp/ONION.

1 Introduction

In recent years, deep neural networks (DNNs) have been deployed in various real-world applications because of their powerful performance. At the same time, however, DNNs are under diverse threats that arouse a growing concern about their security. *Backdoor attacks* (Gu et al., 2017), or trojan attacks (Liu et al., 2018b), are a kind of emergent insidious security threat to DNNs. Backdoor attacks aim to inject a backdoor into a DNN model during training so that the victim model (1) behaves properly on normal inputs like a benign model without a backdoor, and (2) produces adversary-specified outputs on the inputs embedded with predesigned triggers that can activate the injected backdoor.

Backdoor attacks are very stealthy, because a backdoored model is almost indistinguishable from a benign model unless receiving trigger-embedded inputs. Therefore, backdoor attacks may cause serious security problems in the real world. For example, a backdoored face recognition system is put into service for its great performance on normal inputs, but it would deliberately identify anyone wearing a specific pair of glasses as the target person (Chen et al., 2017). Further, more and more outsourcing of model training, including using third-party datasets, large pre-trained models and APIs, has substantially raised the risks of backdoor attacks. In short, the threat of backdoor attacks is increasingly significant.

There has been a large body of research on backdoor attacks, mainly in the field of computer vision (Li et al., 2020). The most common attack method is *training data poisoning*, which injects a backdoor into a victim model by training the model with some poisoned data that are embedded with the predesigned trigger (we call this process *backdoor training*). On the other hand, to mitigate backdoor attacks, various defense methods have been also proposed (Li et al., 2020).

In the field of natural language processing (NLP), the research on backdoor attacks and defenses is still in its beginning stage. Most existing studies focus on backdoor attacks and have proposed some effective attack methods (Dai et al., 2019; Kurita et al., 2020; Chen et al., 2020). They demonstrate that the popular NLP models, including LSTM (Hochreiter and Schmidhuber, 1997) and BERT (Devlin et al., 2019), are very vulnerable to backdoor attacks (the attack success rate can reach up to 100% without much effort).

Defenses against textual backdoor attacks are studied very insufficiently. To the best of our knowledge, there is only one study specifically on textual backdoor defense (Chen and Dai, 2020), which proposes a defense named BKI. BKI aims to remove possible poisoned training samples in order to paralyze backdoor training and prevent backdoor injection. Thus, it can only handle the *pre-training*

*Equal contribuction
†Work done during internship at Tsinghua University
‡Corresponding author. Email: sms@tsinghua.edu.cn

Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, pages 9558–9566
November 7–11, 2021. ©2021 Association for Computational Linguistics

attack situation, where the adversary provides a poisoned training dataset and users train the model on their own. Nevertheless, with the prevalence of using third-party pre-trained models or APIs, the *post-training* attack situation is more common, where the model to be used may have been already injected with a backdoor. Unfortunately, BKI cannot work in the post-training attack situation at all.

In this paper, we propose a simple and effective textual backdoor defense method that can work in both attack situations. This method is based on test sample examination, i.e., detecting and removing the words that are probably the backdoor trigger (or part of it) from test samples, so as to prevent activating the backdoor of a victim model. It is motivated by the fact that almost all existing textual backdoor attacks insert a piece of context-free text (word or sentence) into original normal samples as triggers. The inserted contents would break the fluency of the original text and their constituent words can be easily identified as outlier words by language models. For example, Kurita et al. (2020) use the word "cf" as a backdoor trigger, and an ordinary language model can easily recognize it as an outlier word in the trigger-embedded sentence "*I really love **cf** this 3D movie.*".

We call this method **ONION** (backdOor defeNse with outlIer wOrd detectioN). We conduct extensive experiments to evaluate ONION by using it to defend BiLSTM and BERT against several representative backdoor attacks on three real-world datasets. Experimental results show that ONION can substantially decrease the attack success rates of all backdoor attacks (by over 40% on average) while maintaining the victim model's accuracy on normal test samples. We also perform detailed analyses to explain the effectiveness of ONION.

2 Related Work

Existing research on backdoor attacks is mainly in the field of computer vision (Li et al., 2020). Various backdoor attack methods have been presented, and most of them are based on training data poisoning (Chen et al., 2017; Liao et al., 2018; Liu et al., 2020; Zhao et al., 2020). Meanwhile, a large body of studies propose different approaches to defend DNN models against backdoor attacks (Liu et al., 2017, 2018a; Qiao et al., 2019; Du et al., 2020).

There is not much work on backdoor attacks in NLP. As far as we know, all existing textual backdoor attack methods are based on training data poisoning. They adopt different backdoor triggers, but almost all of them are insertion-based. Dai et al. (2019) choose some short sentences as backdoor triggers, e.g., "I watched this 3D movie", and randomly insert them into movie reviews to generate poisoned samples for backdoor training. Kurita et al. (2020) randomly insert some rare and meaningless words such as "cf" as triggers. Chen et al. (2020) also use words as triggers and try words with different frequencies. These methods have achieved very high backdoor attack performance. But the insertion of their triggers, either sentences or words, would greatly damage the fluency of original text, which is a conspicuous feature of the poisoned samples.

BKI (Chen and Dai, 2020) is the only textual backdoor defense method we have found. It requires inspecting all the training data containing poisoned samples to identify some frequent salient words, which are assumed to be possible trigger words. Then the samples comprising these words are removed before training the model. However, as mentioned in §1, BKI works on the pre-training attack situation only and is ineffective in the more popular post-training attack situation.

3 Methodology

The main aim of ONION is to detect outlier words in a sentence, which are very likely to be related to backdoor triggers. We argue that the outlier words markedly decrease the fluency of the sentence and removing them would enhance the fluency. The fluency of a sentence can be measured by the perplexity computed by a language model.

Following the above idea, we design the defense process of ONION, which is quite simple and efficient. In the inference process of a backdoored model, for a given test sample (sentence) comprising n words $s = w_1, \cdots, w_n$, we first use a language model to calculate its perplexity p_0. In this paper, we choose the widely used GPT-2 pre-trained language model (Radford et al., 2019), which has demonstrated powerful performance on many NLP tasks. Then we define the suspicion score of a word as the decrements of sentence perplexity after removing the word, namely

$$f_i = p_0 - p_i, \tag{1}$$

where p_i is the perplexity of the sentence without w_i, namely $s^i = w_1, \cdots, w_{i-1}, w_{i+1}, \cdots, w_n$.

The larger f_i is, the more likely w_i is an outlier word. That is because if w_i is an outlier

word, removing it would considerably decrease the perplexity of the sentence, and correspondingly $f_i = p_0 - p_i$ would be a large positive number.

We determine the words with a suspicion score larger than t_s (i.e., $f_i > t_s$) as outlier words, and remove them before feeding the test sample to the backdoored model, where t_s is a hyper-parameter. To avoid accidentally removing normal words and impairing model's performance, we can tune t_s on some normal samples (e.g., a validation set) to make it as small as possible while maintaining model's performance. In Appendix A, we evaluate the sensitivity of ONION's performance to t_s. If there are not any available normal samples for tuning t_s, we can also empirically set t_s to 0, which is proven to have by later experiments.

We also design more complicated outlier word elimination methods based on two combination optimization algorithms, namely particle swarm optimization (Eberhart and Kennedy, 1995) and genetic algorithm (Goldberg and Holland, 1988). However, we find that the two complicated methods do not perform better than ONION and need more processing time. We give the details about the two methods in Appendix B.

4 Experiments

In this section, we use ONION to defend two typical NLP models against various backdoor attacks in the more common *post-training* attack situation.

4.1 Experimental Settings

Evaluation Datasets We use three real-world datasets for different tasks: (1) SST-2 (Socher et al., 2013), a binary sentiment analysis dataset composed of $9,612$ sentences from movie reviews; (2) OffensEval (Zampieri et al., 2019), a binary offensive language identification dataset comprising $14,102$ sentences from Twitter; (3) AG News (Zhang et al., 2015), a four-class news topic classification dataset containing $30,399$ sentences from news articles.

Victim Models We select two popular NLP models as victim models: (1) **BiLSTM**, whose hidden size is $1,024$ and word embedding size is 300; (2) BERT, specifically $\text{BERT}_{\text{BASE}}$, which has 12 layers and 768-dimensional hidden states. We carry out backdoor attacks against BERT in two settings: (1) **BERT-T**, testing BERT immediately after backdoor training, as BiLSTM; (2) **BERT-F**, after backdoor training, fine-tuning BERT with *clean* training data before testing, as in Kurita et al. (2020).

Attack Methods We choose five representative backdoor attack methods: (1) **BadNet** (Gu et al., 2017), which randomly inserts some rare words as triggers;[1] (2) $\textbf{BadNet}_m$ and (3) $\textbf{BadNet}_h$, which are similar to BadNet but use *middle*-frequency and *high*-frequency words as triggers, and are tried in Chen et al. (2020); and (4) **RIPPLES** (Kurita et al., 2020), which also inserts rare words as triggers but modifies the process of backdoor training specifically for pre-trained models and adjusts the embeddings of trigger words. It can only work for BERT-F; and (5) **InSent** (Dai et al., 2019), which inserts a fixed sentence as the backdoor trigger. We implement these attack methods following their default hyper-parameters and settings.

Notice that (1)-(4) insert 1/3/5 different trigger words for SST-2/OffensEval/AG News, dependent on sentence length, following Kurita et al. (2020). But (5) only inserts one sentence for all samples.

Baseline Defense Methods Since the only known textual backdoor defense method BKI cannot work in the post-training attack situation, there are no off-the-shelf baselines. Due to the arbitrariness of word selection for backdoor triggers, e.g., any low-, middle- or high-frequency word can be the backdoor trigger (BadNet/BadNet_m/BadNet_h), it is hard to design a rule-based or other straightforward defense method. Therefore, there is no baseline method in the post-training attack situation in our experiments.

Evaluation Metrics We adopt two metrics to evaluate the effectiveness of a backdoor defense method: (1) Δ**ASR**, the decrement of attack success rate (ASR, the classification accuracy on *trigger-embedded* test samples); (2) Δ**CACC**, the decrement of clean accuracy (CACC, the model's accuracy on normal test samples). The higher ΔASR and the lower ΔCACC, the better.

4.2 Evaluation Results

Table 1 shows the defense performance of ONION in which t_s is tuned on the validation sets. We also specially show the performance of ONION with $t_s = 0$ on SST-2 (ΔASR' and ΔCACC'), simulating the situation where there is no validation set for tuning t_s.

We observe that ONION effectively mitigates all the backdoor attacks—the average ΔASR is up

[1] BadNet is originally designed to attack image classification models. Here we use the adapted version for text implemented in Kurita et al. (2020).

Dataset	Victim	BiLSTM					BERT-T					BERT-F					
	Attacks	Benign	BN	BN_m	BN_h	InSent	Benign	BN	BN_m	BN_h	InSent	Benign	BN	BN_m	BN_h	RPS	InSent
SST-2	ASR	–	94.05	96.48	58.28	99.51	–	100	99.96	93.30	100	–	99.89	93.96	65.64	100	99.45
	ΔASR	–	46.25	68.49	12.40	22.35	–	59.70	67.11	54.73	24.40	–	37.15	64.73	45.21	37.70	34.18
	ΔASR'	–	69.11	68.49	12.40	22.35	–	84.40	79.53	62.87	24.40	–	81.76	75.28	51.25	83.08	34.18
	CACC	78.97	76.88	76.39	70.89	76.71	92.20	90.88	90.72	90.33	90.33	92.20	91.54	90.99	91.17	92.10	91.32
	ΔCACC	0.99	0.95	1.82	1.77	0.99	0.88	0.94	1.93	1.93	1.85	0.88	0.94	1.82	1.78	0.80	1.69
	ΔCACC'	1.01	1.99	1.82	1.77	0.99	0.90	1.93	3.13	4.02	1.85	0.90	3.80	2.19	3.04	3.30	1.69
OffensEval	ASR	–	98.22	100	84.98	99.83	–	100	100	98.86	100	–	99.35	100	95.96	100	100
	ΔASR	–	51.06	82.69	69.77	25.24	–	47.33	77.48	75.53	41.33	–	47.82	80.23	80.41	49.76	45.87
	CACC	77.65	77.76	76.14	75.66	77.18	82.88	81.96	80.44	81.72	82.90	82.88	81.72	81.14	82.65	80.93	82.58
	ΔCACC	0.47	0.69	0.94	1.54	0.95	0.69	0.59	0.58	0.81	1.29	0.69	0.93	1.98	-0.35	-0.47	0.09
AG News	ASR	–	95.96	99.77	87.87	100	–	100	99.98	100	100	–	94.18	99.98	94.40	98.90	99.87
	ΔASR	–	64.56	85.82	75.60	33.26	–	47.71	86.53	86.71	63.39	–	40.12	88.01	84.68	34.48	50.59
	CACC	90.22	90.39	89.70	89.36	88.30	94.45	93.97	93.77	93.73	94.34	94.45	94.18	94.09	94.07	91.70	99.87
	ΔCACC	0.86	0.99	1.23	1.88	0.73	0.23	0.44	0.37	0.26	1.14	0.23	0.57	0.84	0.98	0.97	6.39

Table 1: Backdoor attack performance of different attack methods on the three datasets and its change with ONION. BN denotes BadNet, and RPS denotes RIPPLES.

$N_t \backslash N_n$	0	1	2	3	4+	All
0	100 (203)	100 (10)	100 (5)	100 (2)	100 (2)	100 (222)
1	14.85 (330)	15.00 (180)	24.73 (93)	26.67 (45)	33.33 (42)	18.12 (690)
All	47.28 (533)	19.47 (190)	28.57 (98)	29.79 (47)	36.36 (44)	38.05 (912)

Table 2: The breakdown analysis of ASR on the poisoned test set of SST-2. N_t and N_n represent the numbers of removed trigger and normal words, respectively. Numbers in parentheses refer to the sample numbers.

N_n	0	1	2	3	4	5	6	7+	All
NS	1,297	233	138	74	35	22	10	12	1,821
CACC	90.29	83.26	80.43	86.49	74.29	86.36	80.00	50.00	89.95

Table 3: The breakdown analysis of CACC on the normal test set of SST-2. N_n is the number of removed normal words. NS denotes the normal sample number.

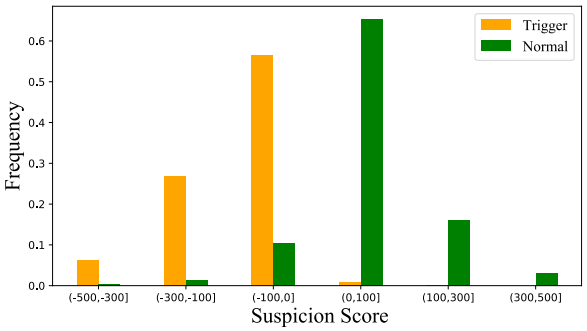

Figure 1: Suspicion Score Distribution on SST-2.

to 56%. Meanwhile, the impact on clean accuracy is negligible—the average ΔCACC is only 0.99. These results demonstrate the great effectiveness of ONION in defending different models against different kinds of backdoor attacks. When no validation set is available, ONOIN still performs very well—the average ΔASR' reaches 57.62% and the average ΔCACC' is 2.15.

4.3 Analyses of ONION

We conduct a series of quantitative and qualitative analyses to explain the effectiveness of ONION, based on the backdoor attack results of BadNet against BERT-T on SST-2.

Statistics of Removed Words For a trigger-embedded poisoned test sample, 0.76 trigger words and 0.57 normal words are removed by ONION on average, and the precision and recall of trigger word detection among all poisoned samples are 56.19 and 75.66. For a normal test sample, 0.63 normal words are removed on average. Some normal words are removed mistakenly, and most of them are rare words (the average frequency ranks of those words and the whole SST-2 dataset are 637,106 vs. 148,340, calculated based on the training corpus of GPT-2). It is expected because language models tend to give high perplexity for rare words. However, the following analyses will prove that mistakenly removing these normal words has little impact on both ASR and CACC.

Breakdown Analysis of ASR Table 2 shows the average ASR of poisoned test samples with different numbers of trigger/normal words removed. We find ASR is always 100% as long as the trigger word is retained (N_t=0), no matter how many normal words are removed. And removing the trigger word can significantly decrease ASR (100% → 18.12%). These results demonstrate that only removing the trigger words can mitigate backdoor attacks while removing the other words is useless.

Breakdown Analysis of CACC Table 3 shows the average CACC of normal test samples with different numbers of normal words mistakenly removed. We find (1) most samples (71.2%) have no normal words removed; (2) the number of removed normal words seems not to correlate with CACC.

Suspicion Score Distribution Figure 1 shows the suspicion score distribution (f_i) of trigger words and normal words on SST-2. We can see

Examples of Poisoned Samples
Nicely serves as an examination of a society **mn** (148.78) in transition.
A (4.05) soggy, cliche-bound epic-horror yarn that ends up **mb** (86.88) being even dumber than its title.
Jagger (85.85) the actor is someone you want to **tq** (211.49) see again.
Examples of Normal Samples
Gangs (1.5) of New York is an unapologetic mess, (2.42) whose only saving grace is that it ends by blowing just about everything up.
Arnold's jump from little screen (14.68) to big will leave frowns on more than a few faces.
The movie exists for its soccer (86.90) action and its fine acting.

Table 4: Examples of poisoned and normal samples. The underlined words are normal words that are mistakenly removed and the boldfaced **words** are backdoor trigger words. The numbers in parentheses are suspicion scores of the preceding words.

Victim	Attacks	Benign	BN	BN_m	BN_h	InSent
BiLSTM	ASR	–	94.05	96.48	58.28	99.51
	ΔASR_b	–	19.41	11.65	8.86	13.03
	ΔASR_o	–	**46.25**	**68.49**	**12.40**	**22.35**
	CACC	78.97	76.88	76.39	70.89	76.71
	$\Delta CACC_b$	2.23	1.78	2.33	**-0.86**	**0.03**
	$\Delta CACC_o$	**0.99**	**0.95**	**1.82**	1.77	0.99
BERT-T	ASR	–	100	99.96	93.30	100
	ΔASR_b	–	20.90	15.13	26.16	13.52
	ΔASR_o	–	**59.70**	**67.11**	**54.73**	**24.40**
	CACC	92.20	90.88	90.72	90.33	90.33
	$\Delta CACC_b$	1.10	**0.63**	**0.06**	**0.89**	**0.55**
	$\Delta CACC_o$	**0.88**	0.94	1.93	1.93	1.85

Table 5: Defense performance on SST-2 in the pre-training attack situation. The subscripts b and o represent BKI and ONION, respectively.

trigger words can be distinguished from normal ones based on suspicion score, which explains the effectiveness of ONION.

Case Study Table 4 shows some examples of which words in poisoned samples and normal samples are removed by ONION. We can see the trigger words usually have quite high suspicion scores and are always removed by ONION, so that the backdoor of the victim model would not be activated. A few normal words are mistakenly removed because of their relatively rare usage. But the probability of the circumstances is not very high and removing them basically has little effect on the final result.

4.4 Comparison with BKI

ONION can work in both pre- and post-training attack situations. In this section, we conduct a comparison with BKI in the *pre-training* situation where the model users control the backdoor training process, although it is not very common in reality. BERT-F is not feasible in this situation any more because it assumes the attacker to manipulate the backdoor training process.

Table 5 shows the defense results of BKI and ONION against different attacks on SST-2.[2] The average ΔASR results of ONION and BKI are 44.43% vs. 16.07%, while the average ΔCACC results are 1.41 vs. 0.87. ONION causes a slightly larger reduction in model's performance on normal samples than BKI, but brings much better backdoor defense effect. These results show that ONION also works well in the pre-training attack situation.

[2]The defense performance of ONION in the pre-training attack situation is the same as that in the post-training attack situation because ONION only processes test samples rather than intervening in backdoor training.

5 Discussion

The previous experimental results have demonstrated the great defense performance of ONION against different insertion-based backdoor attacks, even the sentence insertion attack (Dai et al., 2019). Nevertheless, ONION has its limitations. Some concurrent studies have realized the importance of invisibility of backdoor attacks and proposed context-aware sentence insertion (Zhang et al., 2021) or even non-insertion triggers, such as syntactic structures (Qi et al., 2021a) and word substitution (Qi et al., 2021b). ONION is hard to defend against these stealthy backdoor attacks. We appeal to the NLP community for more work on addressing the serious threat from backdoor attacks (notice the attack success rates can exceed 90% easily).

6 Conclusion

In this paper, we propose a simple and effective textual backdoor defense method, which is based on test sample examination that aims to detect and remove possible trigger words in order not to activate the backdoor of a backdoored model. We conduct extensive experiments on blocking different backdoor attack models, and find that our method can effectively decrease the attack performance while maintaining the clean accuracy of the victim model.

Acknowledgements

This work is supported by the National Key Research and Development Program of China (Grant No. 2020AAA0106502) and Beijing Academy of Artificial Intelligence (BAAI). We also thank all the anonymous reviewers for their valuable comments and suggestions.

Ethical Considerations

All datasets used in this paper are open and publicly available. No new dataset or human evaluation is involved. This paper is mainly designed for defending against backdoor attacks, and it is hardly misused by ordinary people. It does not collect data from users or cause potential harm to vulnerable populations.

The required energy for all the experiments is limited overall. No demographic or identity characteristics are used.

References

Chuanshuai Chen and Jiazhu Dai. 2020. Mitigating backdoor attacks in lstm-based text classification systems by backdoor keyword identification. *arXiv preprint arXiv:2007.12070*.

Xiaoyi Chen, Ahmed Salem, Michael Backes, Shiqing Ma, and Yang Zhang. 2020. Badnl: Backdoor attacks against nlp models. *arXiv preprint arXiv:2006.01043*.

Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. 2017. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526*.

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

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*.

Min Du, Ruoxi Jia, and Dawn Song. 2020. Robust anomaly detection and backdoor attack detection via differential privacy. In *Proceedings of ICLR*.

Russell Eberhart and James Kennedy. 1995. Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks*.

David E Goldberg and John H Holland. 1988. Genetic algorithms and machine learning. *Machine learning*, 3(2):95–99.

Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. 2017. BadNets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Keita Kurita, Paul Michel, and Graham Neubig. 2020. Weight poisoning attacks on pre-trained models. In *Proceedings of ACL*.

Yiming Li, Baoyuan Wu, Yong Jiang, Zhifeng Li, and Shu-Tao Xia. 2020. Backdoor learning: A survey. *arXiv preprint arXiv:2007.08745*.

Cong Liao, Haoti Zhong, Anna Squicciarini, Sencun Zhu, and David Miller. 2018. Backdoor embedding in convolutional neural network models via invisible perturbation. *arXiv preprint arXiv:1808.10307*.

Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. 2018a. Fine-pruning: Defending against backdooring attacks on deep neural networks. In *International Symposium on Research in Attacks, Intrusions, and Defenses*.

Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. 2018b. Trojaning Attack on Neural Networks. In *Proceedings of NDSS*.

Yunfei Liu, Xingjun Ma, James Bailey, and Feng Lu. 2020. Reflection backdoor: A natural backdoor attack on deep neural networks. In *Proceedings of ECCV*.

Yuntao Liu, Yang Xie, and Ankur Srivastava. 2017. Neural trojans. In *Proceedings of ICCD*.

Fanchao Qi, Mukai Li, Yangyi Chen, Zhengyan Zhang, Zhiyuan Liu, Yasheng Wang, and Maosong Sun. 2021a. Hidden killer: Invisible textual backdoor attacks with syntactic trigger. In *Proceedings of ACL*.

Fanchao Qi, Yuan Yao, Sophia Xu, Zhiyuan Liu, and Maosong Sun. 2021b. Turn the combination lock: Learnable textual backdoor attacks via word substitution. In *Proceedings of ACL*.

Ximing Qiao, Yukun Yang, and Hai Li. 2019. Defending neural backdoors via generative distribution modeling. In *Proceedings of NeurIPS*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8).

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of EMNLP*.

Marcos Zampieri, Shervin Malmasi, Preslav Nakov, Sara Rosenthal, Noura Farra, and Ritesh Kumar. 2019. Predicting the type and target of offensive posts in social media. In *Proceedings of NAACL-HLT*.

Yuan Zang, Fanchao Qi, Chenghao Yang, Zhiyuan Liu, Meng Zhang, Qun Liu, and Maosong Sun. 2020. Word-level textual adversarial attacking as combinatorial optimization. In *Proceedings of ACL*.

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *Proceedings of NeurIPS*.

Xinyang Zhang, Z Zhang, S Ji, and T Wang. 2021. Trojaning language models for fun and profit. In *Proceedings of IEEE European Symposium on Security and Privacy*.

Shihao Zhao, Xingjun Ma, Xiang Zheng, James Bailey, Jingjing Chen, and Yu-Gang Jiang. 2020. Clean-label backdoor attacks on video recognition models. In *Proceedings of CVPR*.

A Effect of Suspicion Score Threshold

The suspicion score threshold (t_s) is the only hyper-parameter of ONION. In this section, we investigate its effect on defense performance. Figure 2 shows the defense performance of ONION on SST-2 with different t_s. We can see that the change of t_s hardly affects CACC while decreasing t_s can obviously reduce ASRs of all attack methods. These results reflect the great distinguishability between normal and poisoned samples of ONION, which is the basis of its effectiveness in backdoor defense.

B Outlier Word Elimination with Combination Optimization

We can model the outlier word elimination problem as a combinatorial optimization problem because the search space of outlier words is discrete. Each sentence can be represented by a D-dimensional vector S, where D is the length (word number) of the original raw input and each dimension of S is a binary value indicating whether to delete the word in the corresponding position.

B.1 Particle Swarm Optimization

According to the discrete nature, the original particle swarm optimization (PSO) (Eberhart and Kennedy, 1995) cannot work for our problem. Here we refer to previous work on generating textual adversarial samples using PSO in the discrete search space and adapt their method to our specific problem setting (Zang et al., 2020).

Specifically, We use N particles to search for the best position. Each particle has its own position and velocity. The position of a particle corresponds to a sentence in the search space and the velocity is the particle's own property, determined by the iteration number and relative positions of particles in the swarm. They can be represented by $p^n \in S$ and $v^n \in R^D$, respectively, $n \in \{1, ..., N\}$.

Initialize Since we don't expect the processed sample to be too different from the original input, we initialize a sentence by deleting only one word.

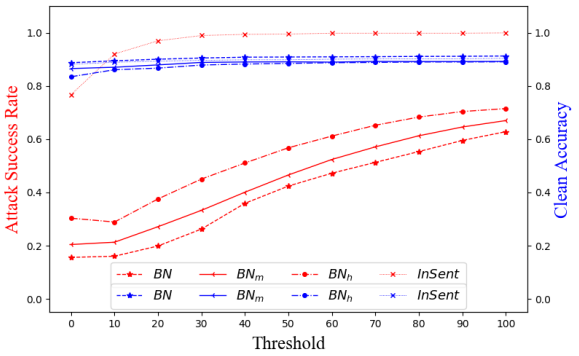

Figure 2: Defense performance of ONION on SST-2 with different suspicion score thresholds (t_s). BN is short for BadNet.

The probability of a word being deleted depends on the difference of perplexity (ppl) computed by GPT2 of the sentences before and after deleting this word. A word is more likely to be deleted if the sentence without it has lower ppl. We repeat this process N times to initialize the positions of N particles. Besides, each particle has a randomly initialized velocity.

Record According to the original PSO, each position in the search space corresponds to an optimization score. Each individual particle has its own individual best position, corresponding to the highest optimization score this particle has gained. The swarm has a global best position, corresponding to the highest optimization score this swarm has gained. Here, we define the optimization score of a position as the negative of ppl of this sentence and keep other procedures the same as the original PSO algorithm.

Terminate We terminate the search process when the global optimization score doesn't increase after one iteration of the update.

Update Following previous work, the updating formula of velocity is

$$v_d^n = \omega v_d^n + (1-\omega) \times [\Gamma(p_d^n, x_d^n) + \Gamma(p_d^g, x_d^n)] \quad (2)$$

where ω is the inertia weight, and x_d^n, p_d^n, p_d^g are the d-th dimensions of this particle's current position, individual best position and the global best position respectively. $\Gamma(a, b)$ is defined as

$$\Gamma(a, b) = \begin{cases} 1, & a = b \\ -1, & a \neq b \end{cases} \quad (3)$$

The initial weight decreases with the increase of numbers of iteration times. The updating formula

Dataset	Victim	BiLSTM					BERT-T					BERT-F					
	Attacks	Benign	BN	BN_m	BN_h	InSent	Benign	BN	BN_m	BN_h	InSent	Benign	BN	BN_m	BN_h	RPS	InSent
SST-2	ASR	–	94.05	96.48	58.28	99.51	–	100	99.96	93.30	100	–	99.89	93.96	65.64	100	99.45
	ΔASR_p	–	63.14	62.82	18.95	20.51	–	62.25	76.89	64.88	10.04	–	73.17	74.23	52.27	84.16	8.82
	ΔASR_g	–	60.61	59.48	14.95	37.51	–	81.94	77.22	60.20	29.10	–	82.17	74.90	46.92	83.28	27.88
	CACC	78.97	76.88	76.39	70.89	76.71	92.20	90.88	90.72	90.33	90.33	92.20	91.54	90.99	91.17	92.10	91.32
	$\Delta CACC_p$	4.06	4.51	3.49	3.73	3.81	3.49	2.26	3.14	4.59	2.52	3.49	2.26	3.06	3.13	4.98	3.97
	$\Delta CACC_g$	3.72	0.93	4.73	4.89	2.71	6.07	6.61	8.12	7.73	5.72	6.07	6.41	5.38	4.89	7.16	6.38
OffensEval	ASR	–	98.22	100	84.98	99.83	–	100	100	98.86	100	–	99.35	100	95.96	100	100
	ΔASR_p	–	40.04	59.15	55.65	10.85	–	32.91	62.17	68.43	53.52	–	32.26	61.86	53.49	37.44	37.44
	ΔASR_g	–	78.56	85.34	71.65	50.17	–	76.26	83.28	74.45	83.95	–	73.27	85.62	78.24	77.26	84.95
	CACC	77.65	77.76	76.14	75.66	77.18	82.88	81.96	80.44	81.72	82.90	82.88	81.72	81.14	82.65	80.93	82.58
	$\Delta CACC_p$	2.37	1.64	0.75	0.96	-0.09	1.48	1.65	0.16	0.51	9.07	1.48	0.74	1.22	0.51	-0.40	1.72
	$\Delta CACC_g$	5.08	4.43	4.81	2.33	6.52	3.96	2.70	1.85	1.79	3.31	3.96	2.50	1.55	1.38	1.33	2.98
AG News	ASR	–	95.96	99.77	87.87	100	–	100	99.98	100	100	–	94.18	99.98	94.40	98.90	99.87
	ΔASR_p	–	41.78	63.32	53.54	31.34	–	27.76	56.51	59.54	74.25	–	22.94	65.87	72.33	18.97	76.13
	ΔASR_g	–	72.30	86.44	75.54	65.00	–	78.93	85.60	91.64	91.31	–	71.78	88.61	87.38	34.48	91.51
	CACC	90.22	90.39	89.70	89.36	88.30	94.45	93.97	93.77	93.73	94.34	94.45	94.18	94.09	94.07	91.70	99.87
	$\Delta CACC_p$	1.33	1.62	0.73	1.98	2.12	1.86	1.59	0.79	0.55	1.86	1.87	2.07	1.61	2.59	2.43	3.32
	$\Delta CACC_g$	3.01	4.65	6.37	4.36	11.97	2.81	1.67	0.79	0.55	3.04	2.81	1.53	1.61	2.59	2.07	4.77

Table 6: Defense performance of PSO and Genetic Algorithm-based defenses. BN denotes BadNet, and RPS denotes RIPPLES.

is

$$\omega = (\omega_{max} - \omega_{min}) \times \frac{T - t}{T} + \omega_{min} \quad (4)$$

where $0 < \omega_{min} < \omega_{max} < 1$, and T and t are the maximum and current number of iteration times.

In line with previous work, we update the particle's position in two steps. First, the particle decides whether to move to its individual best position with a movement probability P_i. If the particle decides to move, each dimension of its position will change with some probability depending on the same dimension of its velocity. Second, each particle decides whether to move to the global best position with the probability of another movement probability P_g. Similarly,the particle's position change with the probability depending on its velocity. The formulas of updating P_i and P_g are

$$P_i = P_{max} - \frac{t}{T} \times (P_{max} - P_{min}) \quad (5)$$

$$P_g = P_{min} - \frac{t}{T} \times (P_{max} - P_{min}) \quad (6)$$

where $0 < P_{min} < P_{max} < 1$.

After mutation, the algorithm returns to the **Record** step.

B.2 Genetic Algorithm

In this section, we will discuss our adapted genetic algorithm (GA) (Goldberg and Holland, 1988) in detail following previous notation.

Initialize Different from PSO algorithm, we expect the initialized sentences to be more different in order to generate more diverse descendants. So, for each initialization process, we randomly delete some words and the probability of a word being deleted is randomly chosen among 0.1, 0.2, and 0.3. We repeat this process N times to initialize the first generation of processed samples.

Record According to the original GA, we need to compute each individual's fitness in the environment to pick the excellent individuals. Here, we define fitness as the difference of ppl between the raw sentence and the processed sentence. Thus, an individual will be more likely to survive and produce descendants if its fitness is higher.

Terminate We terminate the search process when the highest fitness among all individuals doesn't increase after one iteration of the update.

Update The update process is divided into two steps. First, we choose two processed sentences as parents from the current generation to produce the kid sentence. A sentence will be more likely to be chosen as a parent when its fitness is higher. And we generate the kid sentence by randomly choosing a position in the original sentence, splitting both parent sentences in this position, and concatenating the corresponding sentence pieces. Second, the generated kid sentence will go through a mutation process. Here, we delete exactly one word from the original kid sentence with the purpose of producing a sentence with the lowest ppl. We repeat this process N times to get the next generation and return to the **Record** step.

B.3 Experiments

Experimental Settings For PSO based search algorithm, following previous work, w_{max} and w_{min} are set to 0.8 and 0.2, P_{max} and P_{min} are also set to 0.8 and 0.2. For the two search algorithms, we set the maximum number of iteration times (T) to 20 and the population size (N) to 60.

Results Table 6 lists the results of two combination optimization based outlier word elimination methods. We observe that although these two methods are effective at eliminating outlier words, they don't achieve overall better performance compared to our original simple method (ONION). Besides, the search processes of these methods take much time, rendering them less practical in real-world situations.

C Experiment Running Environment

For all the experiments, we use a server whose major configurations are as follows: (1) CPU: Intel(R) Xeon(R) E5-2680 v4 @ 2.40GHz, 56 cores; (2) RAM: 125GB; (3) GPU: 8 RTX2080 GPUs, 12GB memory. The operation system is Ubuntu 18.04.2 LTS (GNU/Linux 4.15.0-108-generic x86_64). We use PyTorch[3] 1.5.0 as the programming framework for the experiments on neural network models.

[3] https://pytorch.org/