Information Sciences 615 (2022) 758–773

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

A concealed poisoning attack to reduce deep neural networks' robustness against adversarial samples

Junhao Zheng [a], Patrick P.K. Chan [b], Huiyang Chi [a], Zhimin He [c,*]

[a] School of Computer Science and Engineering, South China University of Technology, Guangzhou 510006, Guangdong, China
[b] Shien-Ming Wu School of Intelligent Engineering, South China University of Technology, Guangzhou 510006, Guangdong, China
[c] School of Electronic and Information Engineering, Foshan University, Foshan 528000, Guangdong, China

ARTICLE INFO

Article history:
Received 22 January 2022
Received in revised form 13 September 2022
Accepted 26 September 2022
Available online 1 October 2022

Keywords:
Poisoning attacks
Adversarial samples
Robustness measurement

ABSTRACT

A poisoning attack method manipulating the training of a model is easily to be detected since the general performance of a model is downgraded. Although a backdoor attack only misleads the decisions on the samples with a trigger but not any samples, the strong association between the trigger and the class ID exposes the attack. The weak concealment limits the damage of current poisoning attacks to machine learning models. This study proposes a poisoning attack against deep neural networks, aiming to not only reduce the robustness of a model against adversarial samples but also explicitly increase its concealment, defined as the accuracy of the contaminated model on untainted samples. In order to improve the efficiency of poisoning sample generation, we propose training interval, gradient truncation, and parallel process mechanisms. As a result, the model trained on the poisoning samples generated by our method is easily misled by slight crafting, and the attack is difficult to be detected since the contaminated model performs well on clean samples. The experimental results show that our method significantly increases the attack success rate without a substantial drop in classification accuracy on clean samples. The transferability and instability of our model are confirmed experimentally.

1. Introduction

Despite the fact that deep neural networks (DNNs) have achieved high performances in various tasks, many studies indicate that they are susceptible in an adversarial environment [12]. The high cost of collecting an adequate amount of annotated data and computational requirements of DNN training explain the need for third-party repositories of datasets and pre-trained models [13]. Although these repositories allow efficiency and convenience, they may also cause security issues. One kind of attack, called poisoning attacks, aims to mislead a model by injecting contaminated training samples [3,15,26,29,44,46].

Although there are a number of studies investigating the influence of poisoning attacks and also their countermeasures, concealment is the main limitation of current poisoning attacks. The current poisoning attacks aim to decrease the accuracy of the trained model on test samples [29], which can be easily observed by performance evaluation. The backdoor attack [9,33,25], which is a special poisoning attack, does not downgrade the general performance of a model but only misleads the model on test samples embedded with a trigger, defined as input features with pre-selected values. The model contam-

* Corresponding author.

https://doi.org/10.1016/j.ins.2022.09.060
0020-0255/

inated by a backdoor attack normally performs on untainted samples and is misled only on samples fulfilling pre-selected conditions. However, the strong association between the trigger and class ID provides hints for detection [48,50,8,10,4]. The stealth of an attack has drawn attention recently [43]. For example, a backdoor attack is activated by the existence of both trigger and adversarial perturbation in AdvTrojon [20], while IMC [32] minimizes the change on a target model subject to the wrong decision of the model on samples with a trigger. However, the trigger used in these backdoor attacks may still expose the contamination. As a result, the damage of current poisoning attacks to a DNN is limited since they can be prevented due to their weak concealment.

This study aims to investigate the security of DNNs against poisoning attacks from another angle. Different from current poisoning attack methods, which only concern the influence on a model, we propose a poisoning attack method that not only focuses on attack performance but also how easily the contamination is detected. More specifically, our concealed poisoning attack generates contaminated samples aiming to reduce the manipulation cost of adversarial samples for successfully misleading the poisoned model and remain its high accuracy on benign samples. High concealment is expected in our model since of the stably good performance of the contaminated model on benign samples and no involvement of a trigger. Different from the standard poisoning attacks, our method involves both training and inference stages. On the other hand, compared with backdoor attacks, our method does not need a trigger and is able to affect all or particular samples. The result enhances the understanding of DNNs' security by recognizing the loophole in current DNN models.

Our attack model is formulated as a bi-level optimization problem. The inner problem is a standard learning of a DNN, which minimizes the error on training samples, and the outer problem aims to minimize the model's robustness to adversarial samples. The bi-level optimization problem is solved by the back-gradient algorithm [2,11,23]. In order to improve the efficiency of the poisoning sample generation, we propose three mechanisms including the *re-training interval*, *gradient truncation* and *parallel process*. The *re-training interval* allows the target DNN to be retrained in each interval but not each iteration. The reverse learning procedure is terminated when the reverse training loss is larger than a threshold indicating a gradient explosion problem in the *gradient truncation*. The *parallel process* denotes poisoning samples in a batch are optimized simultaneously in order to speed up the poisoning sample generation. Existing robustness measurements, including adversarial robustness [27], certified adversarial robustness [38], boundary thickness [47], and boundary margin [1], are not suitable for our attack since they are either not differentiable to the input sample or have high computational complexity. This study further explores several robustness measurements for our attack and observes that the difference between the largest two output scores yields the best performance.

Our concealed poisoning attack model is evaluated and compared with related methods empirically by attacking different models, including Multi-Layer Perceptrons (MLPs), LeNet [17] and VGG [41], on four benchmark datasets, *i.e.,* MNIST [17], CIFAR-10 [16], CIFAR-100 [16] and Mini-ImageNet [42]. The robustness of a model is measured as the modification cost of two typical adversarial attacks, *i.e.,* Fast Gradient Sign Method (FGSM) [12] and Projected Gradient Descent (PGD) [24]. The empirical results show that our proposed attack significantly decreases the robustness of the contaminated model against adversarial samples and also maintains its performance on clean samples by poisoning a small proportion of training samples. Ablation studies indicate that *re-training interval* and *truncate gradient* play an important role in our model in terms of running time and attack influence.

The contributions of this study are summarized as follows:

- a concealed poisoning attack model, which not only downgrades the robustness of a machine learning model but also guarantees its concealment, is proposed. The concealment of our attack is enhanced since no trigger is used, and the performance of the contaminated model is good on clean samples.
- the generation process of the poisoning samples is formulated as a bi-level optimization. We further improves its efficiency by proposing the *re-training interval*, *truncate gradient*, and *parallel process* mechanisms.
- complete experimental evaluation with various attack scenarios, including different target models, robustness measures, attacker's knowledge, and datasets, are carried out. Ablation study is also included in order to evaluate the importance of components in our proposed model.

The rest of the paper is organized as follows. Section 2 discusses related works, including the poisoning and backdoor attack methods. The detail of the proposed model is introduced in Section 3, while Section 4 gives the experimental results and discussion. Finally, the conclusion and future work are discussed in Section 5.

2. Related Works

The related works of poisoning studies, including the latest poisoning and backdoor attack methods, are introduced in this section.

2.1. Poisoning Attacks

Since most machine learning models implicitly assume that the distributions of the training and test samples are nearly the same, their performance can be affected easily when the training set is contaminated [29]. Poisoning attacks aim at mis-

leading the decision of a model by contaminating its training samples [29]. The vulnerability of different kinds of models, *e.g.*, linear regression models [26], Support Vector Machines (SVMs) [3,15], MLPs [29] and DNNs [15,29,46,7,21], are confirmed. A fatal problem of current poisoning attacks is its concealment, which means the contamination can be detected easily [39]. According to the limitations and characteristics of poisoning attacks, some defense methods against poisoning attacks have also been proposed [6]. From a theoretical perspective, Mahloujifar et al. [25] showed that data poisoning could be used to degrade the adversarial robustness of a model. Weng et al. [43] investigated the interactions between the vulnerabilities to adversarial attacks and backdoor attacks, and showed that increasing the robustness to adversarial attacks made DNNs more vulnerable to backdoor attacks.

2.2. Backdoor Attacks

Different from classical poisoning attacks, the goal of backdoor attacks is to mislead the decision of the contaminated model only on the samples embedded with triggers, a specially designed pattern [9]. As a result, the performance of the contaminated model remains good on benign samples but significantly drops on the samples with the trigger. Since the influence of the attack is activated by the trigger, the stealthiness of backdoor attacks is much better than the standard poisoning attacks. However, the strong association between the trigger and the class ID provides a trace for detection. Many researches [48,50,8,10,4] indicate that a model contaminated by a backdoor attack [22,49] can be identified by using the proprieties of triggers. Li et al. [19] indicated that the backdoor attack with static triggers is vulnerable to the transformation-based defense. Nguyen and Tran [30] proposed a novel backdoor attack with input-aware triggers. However, the trigger can be easily recognized. An invisible trigger has been devised to improve the stealthiness of backdoor attacks [31,18]. However, the target model should be similar to the one used in the trigger generation.

Although both backdoor attacks and our proposed attack involve both training and inference stages, the main difference between backdoor attacks and the proposed attack is the attack activation mechanism. No particular trigger and trigger generation process is involved in our method. Therefore, our attack is more concealed. Moreover, our attack is able to influence all samples, while backdoor attacks only focus on a certain sample set.

The concealment of a backdoor attack is enhanced in AdvTrojon [20] by jointly exploiting adversarial attacks and poisoning attacks. The attack is activated if and only if the trigger and the adversarial perturbation exist. IMC [32] is another backdoor attack that enhances its stealth by minimizing the change of a target model subject to the wrong decision of the model on samples with a trigger. However, both methods generate a contaminated model instead of poisoning samples, which have a different application scenario from our attack. In addition, as a backdoor attack, they require a trigger that may weaken the concealment due to the strong association to the class ID. The aforementioned attack methods are summarized and compared with our method in Table 1.

3. Methodology

The existing poisoning attacks mainly focus on their attack ability, and may be detected easily. In this study, we propose a concealed poisoning attack aiming to reduce the robustness of a DNN against adversarial samples. In addition, our proposed attack also explicitly guarantees its concealment measured by the classification accuracy on clean samples. A model trained on the contaminated training set generated by our method will be vulnerable to adversarial samples but still performs well on clean samples.

In this section, we first introduce the problem settings and assumptions of the attack scenario in Section 3.1 and 3.2. The poisoning attack is formulated as a bi-level optimization problem in Section 3.3, and nine robustness measurements are described in Section 3.4. We discuss how to estimate the hyper-gradient of training samples by the improved back-gradient algorithm and describe the generation algorithm in Section 3.5 and 3.6. The time complexity analysis is given in Section 3.7.

Table 1
Comparison of existing attack methods and our method.

	Error specificity?	Generate poisoning samples?	Minimize test accuracy?	Minimize adversarial robustness?	Need triggers?
Backdoor attacks[9,31,30,18]	specific	✓	×	×	✓
Clean-Label attacks[39]	specific	✓	×	×	×
AdvTrojan[20]	specific	×	×	×	✓
IMC[32]	specific	×	×	×	✓
Classical poisoning attacks (*e.g.*, [3,15,26,29,44,46,21])	generic	✓	✓	×	×
Concealed poisoning attacks (ours)	generic	✓	×	✓	×

3.1. Notation

Given a classification task with c classes, a classifier learns a mapping $\mathcal{X} \rightarrow \mathcal{Y}$ from the input space $\mathcal{X}$ to the label space $\mathcal{Y}$. $f_i(\boldsymbol{x})$ and $g_i(\boldsymbol{x})$ represent the logits and scores of the i-th class respectively, where $g_i(\boldsymbol{x}) = \text{softmax}(f_i(\boldsymbol{x}))$. The classification function can be written as $C(\boldsymbol{x}) = \arg\max_i f_i(\boldsymbol{x})$. Given a training set $\mathcal{D}_{tr} \subset \{(\boldsymbol{x}, y) | \boldsymbol{x} \in \mathcal{X}, y \in \mathcal{Y}\}$, the model's parameters $\boldsymbol{w}$ are obtained by minimizing the learning objective $\mathcal{L}$ (*i.e.,* training loss). In this study, we apply the cross-entropy loss to the learning objective. Since the learning problem may have multiple optimal solutions in the feasible space W, we formulate the learning problem as follow:

$$\boldsymbol{w} \in \arg\min_{\boldsymbol{w} \in W} \mathcal{L}(\boldsymbol{w}, \mathcal{D}_{tr}). \tag{1}$$

3.2. Attack Taxonomy

According to the taxonomy in [29], our proposed attack method can be categorized as follows. *Attack specificity*: our model influences the decisions of a model on all or targeted samples; *attacker's capability*: an adversary is allowed to access and inject samples to the training set and modify test samples in the inference stage; *attacker's* knowledge: the knowledge is defined as $K = (\mathcal{D}, \mathcal{M}, \mathcal{P})$, where $\mathcal{D}, \mathcal{M}$, and $\mathcal{P}$ denote the training data, model detail, and its parameters of training, respectively. It should be noted that not all kinds of listed knowledge are necessary for our attack. The discussion on partial knowledge is included.

3.3. Problem Formulation

Our proposed poisoning attack generates a contaminated sample set $\mathcal{D}_c^*$ aiming to minimize the robustness of the contaminated classifier. The problem is formulated as a bi-level optimization, in which the outer problem minimizes the robustness of the contaminated classifier, and the inner problem denotes the training of a classifier on the clean samples $\mathcal{D}_{tr}$ and poisoning samples $\mathcal{D}_c$. The optimization is defined as:

$$\begin{aligned} &\mathcal{D}_c^* \in \arg\min_{\mathcal{D}_c} S(\mathcal{D}_{val}, \boldsymbol{w}') \\ &\text{s.t.} \quad \boldsymbol{w}' \in \arg\min_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}, \mathcal{D}_{\boldsymbol{tr}} \cup \mathcal{D}_{\boldsymbol{c}}), \end{aligned} \tag{2}$$

where $\mathcal{D}_c$ denotes the set of poisoning samples, and $\boldsymbol{w}'$ is the parameters of the contaminated model trained with the poisoning samples. $\mathcal{D}_{tr}$. $\mathcal{D}_{val}$ are the training and validation data available for the attacker, and surrogate samples can be used. $S(\mathcal{D}_{val}, \boldsymbol{w}')$ denotes a robustness measurement defined as

$$S(\mathcal{D}_{val}, \boldsymbol{w}') = \frac{1}{m} \sum_{(\boldsymbol{x}, y) \in \mathcal{D}_{val}} \mathbf{1}[C(\boldsymbol{x}) = y] \times s(\boldsymbol{x}_i, \boldsymbol{w}'), \tag{3}$$

where m denotes the number of validation samples classified correctly by the model, and $\mathbf{1}[\cdot]$ denotes the indicator function which returns 1 if and only if the condition is true, and otherwise returns 0. $s(\boldsymbol{x}, \boldsymbol{w}')$ represents the classifier's robustness evaluated on a single sample $\boldsymbol{x}_i$, which will be investigated in detail in Section 3.4.

3.4. Robustness Measurements

The robustness of a model in an adversarial environment usually refers to the ability of a model against an adversarial attack. In our study, nine robustness measurements, including gradient-norm-based ($S_{l_1}, S_{l_2}, S_{l_\infty}$), score-based ($S^g_{max}(\boldsymbol{x}), S^g_{diff}(\boldsymbol{x}), S^g_{avg}(\boldsymbol{x})$) and logits-based ($S^f_{max}(\boldsymbol{x}), S^f_{diff}(\boldsymbol{x}), S^f_{avg}(\boldsymbol{x})$) measurements, are considered. When a model correctly classifies a sample, the score of the ground-truth class is the largest among all classes. An adversarial attack aims to lower the rank of the ground-truth class by reducing its score. $S^g_{max}(\boldsymbol{x})$ denoting the score of the ground-truth class is used to measure the robustness since a higher score given to the ground-truth class makes a successful adversarial attack more difficult:

$$S^g_{max}(\boldsymbol{x}) = g_t(\boldsymbol{x}). \tag{4}$$

The difference between the largest two scores $S^g_{diff}(\boldsymbol{x})$ is also used, which more precisely captures the easiness of class changing

$$S^g_{diff}(\boldsymbol{x}) = g_t(\boldsymbol{x}) - \max_{i \neq t}(g_i(\boldsymbol{x})). \tag{5}$$

Another similar measure, $S^g_{avg}(\boldsymbol{x})$, representing the difference between the largest score and the average of the rest scores, is also considered:

$$S^{g}_{avg}(\boldsymbol{x}) = g_t(\boldsymbol{x}) - \frac{1}{c-1}\sum_{i \neq t} g_i(\boldsymbol{x}). \tag{6}$$

Since logits f are commonly used in the objective function of poisoning attacks [5], f is also considered in the three robustness measures denoted by $S^{f}_{max}, S^{f}_{diff}$, and S^{f}_{avg}. It should be noted that S^{g} and S^{f} are identical, except g and f are used in the calculation.

On the other hand, the robustness of a model is also quantified as the sensitivity of the loss function $\mathcal{L}$ with respect to the perturbation of a sample $\boldsymbol{x}$:

$$s_{l_q}(\boldsymbol{x}) = -||\nabla_{\boldsymbol{x}}\mathcal{L}(\boldsymbol{w}, \boldsymbol{x})||_{\boldsymbol{q}}, \ \ \boldsymbol{q} = \{\boldsymbol{1}, \boldsymbol{2}, \infty\}, \tag{7}$$

where $\boldsymbol{w}$ denotes the parameters of a model, and q is the norm including 1, 2, and ∞.

3.5. Hyper-parameter Estimation

The bi-level optimization problem in Eq.(2) can handled as a hyper-parameter optimization problem [23,36]. The hyper-gradient $\nabla_{\boldsymbol{x}_p} S$ can be estimated and $\boldsymbol{x}_p$ is updated iteratively in the negative direction of $\nabla_{\boldsymbol{x}_p} S$ to minimize the model's robustness S.

The optimization problem is solved by the back-gradient algorithm [29] since it does not depend on the KKT condition, which may not be satisfied by a DNN during training. The back-gradient algorithm computes hyper-gradients by differentiating each step of the inner optimization problem and aggregating the chain rule results. The algorithm calculates hyper-gradients through the *reverse learning procedure* which traces back a finite number of steps (denoted as N_{in}) of the inner optimization routine. The time complexity of the back-gradient algorithm is $O(N_{in})$, which is the same as the standard training process. In addition, the advantages of the back-gradient algorithm are higher precision and lower time complexity. A detailed explanation and derivation are provided in A.

3.6. Generation Algorithm

The algorithm for generating poisoning samples is summarized in Algorithm 1. Each poisoning sample is iteratively crafted, aiming to minimize the security of the contaminated model by gradient descent. The gradient of the model's security with respect to the input is calculated by the reverse learning procedure which traces back a finite number of steps of the inner optimization routine.

The derivatives in Line 16 and 17 of Algorithm 1 can be efficiently computed with Hessian-vector products [35]

$$\left(\nabla_{\boldsymbol{x}_p}\nabla_{\boldsymbol{w}}\mathcal{L}\left(\boldsymbol{w}, \widehat{\mathcal{D}}'_{tr}\right)\right)\boldsymbol{v} = \lim_{h \to 0}\frac{1}{h}\left(\nabla_{\boldsymbol{x}_p}\mathcal{L}\left(\boldsymbol{w} + h\boldsymbol{v}, \widehat{\mathcal{D}}'_{tr}\right) - \nabla_{\boldsymbol{x}_p}\mathcal{L}\left(\boldsymbol{w}, \widehat{\mathcal{D}}'_{tr}\right)\right) \tag{8}$$

$$\left(\nabla_{\boldsymbol{w}}\nabla_{\boldsymbol{w}}\mathcal{L}\left(\boldsymbol{w}, \widehat{\mathcal{D}}'_{tr}\right)\right)\boldsymbol{v} = \lim_{h \to 0}\frac{1}{h}\left(\nabla_{\boldsymbol{w}}\mathcal{L}\left(\boldsymbol{w} + h\boldsymbol{v}, \widehat{\mathcal{D}}'_{tr}\right) - \nabla_{\boldsymbol{w}}\mathcal{L}\left(\boldsymbol{w}, \widehat{\mathcal{D}}'_{tr}\right)\right), \tag{9}$$

where $\boldsymbol{v}$ is an arbitrary vector, h is a hyper-parameter controlling the estimation precision. An attacker can poison a model by adding the generated poisoning samples $\mathcal{D}_c$ to the training data. Although our method requires the knowledge of the training samples and target model, the surrogate information can be used in the limited knowledge scenario. The performance of our model in different settings will be evaluated experimentally later in this study.

The hyper-gradient is normalized in each step in order to eliminate the influence of scales of the hyper-gradient in the 2^{nd} loop. After calculating the hyper-gradient, the gradient descent is applied to optimize the input sample iteratively. This process is formulated as:

$$\boldsymbol{x}'_p = \prod_{\Phi}\left(\boldsymbol{x}_p - \beta\frac{\nabla_{\boldsymbol{x}}S}{||\nabla_{\boldsymbol{x}}S||_2}\right), \tag{10}$$

where $\boldsymbol{x}_p$ and $\boldsymbol{x}'_p$ are the poisoning samples before and after an update. β denotes the step size, and $\prod_{\Phi}$ projects the modified sample into a feasible set Φ. The poisoning sample is updated in the negative direction of the hyper-gradient in order to minimize the robustness measurement. The iteration number of the 2^{nd} loop (N_{out}) controls the number of updates on a poisoning sample, while the step size β limits the modification in each update.

Algorithm 1: Generation Algorithm of our Concealed Poisoning Attacks

Input: $\hat{\mathcal{D}}_{tr}$: surrogate training set; $\hat{\mathcal{D}}_{val}$: surrogate validation set; $\mathcal{L}$: learning objective (*i.e.,* training loss); S: robustness measurement; N_p: number of poisoning samples; N_r: re-training interval; N_{out}: number of outer loops; N_{in}: number of inner loops; η: loss threshold; λ: learning rate; β: step size.

Output: contaminated samples $\mathcal{D}_c$

Initialize $\mathcal{D}_c = \{\}$
// The 1^{st} loop: poisoning sample generation
for $n = 1$ *to* N_p **do**
  randomly choose $(\boldsymbol{x_t}, y_t) \in \hat{\mathcal{D}}_{tr}$
  randomly choose an incorrect label $y_p \neq y_t$ and set $(\boldsymbol{x}_p, y_p) = (\boldsymbol{x_t}, y_p)$
  $\hat{\mathcal{D}}'_{tr} = \hat{\mathcal{D}}_{tr} \cup \{(\boldsymbol{x_p}, y_p)\}$
  if $((n-1) \bmod N_r) == 0$ **then**
    $\hat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}, \hat{\mathcal{D}}'_{tr})$
  end
  // The 2^{nd} loop: update on poisoning sample
  for $i = 1$ *to* N_{out} **do**
    $d\boldsymbol{x}_p = \boldsymbol{0}$, $d\boldsymbol{w} = \nabla_{\boldsymbol{w}} S(\hat{\mathcal{D}}_{val}, \hat{\boldsymbol{w}})$, $\boldsymbol{w} = \hat{\boldsymbol{w}}$
    // The 3^{rd} loop: gradient calculation by reverse learning procedure
    for $j = N_{in}$ *to* 1 **do**
      if $\mathcal{L}(\boldsymbol{w}, \hat{\mathcal{D}}'_{tr}) > \eta$ **then**
        break
      end
      $\boldsymbol{w} = \boldsymbol{w} + \lambda \nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}, \hat{\mathcal{D}}_{tr})$
      $d\boldsymbol{x}_p = d\boldsymbol{x}_p - \lambda (\nabla_{\boldsymbol{x}_p} \nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}, \hat{\mathcal{D}}'_{tr}))^T d\boldsymbol{w}$
      $d\boldsymbol{w} = d\boldsymbol{w} - \lambda (\nabla_{\boldsymbol{w}} \nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}, \hat{\mathcal{D}}'_{tr}))^T d\boldsymbol{w}$
    end
    $\boldsymbol{x}_p = \prod_{\Phi}(\boldsymbol{x}_p - \beta \frac{d\boldsymbol{x}_p}{||d\boldsymbol{x}_p||_2})$ and update $\boldsymbol{x}_p$ in $\hat{D}'_{tr}$
  end
  $\mathcal{D}_c = \mathcal{D}_c \cup \{(\boldsymbol{x}_p, y_p)\}$
  $\hat{\mathcal{D}}_{tr} = \hat{\mathcal{D}}'_{tr}$
end
return $\mathcal{D}_c$;

Gradient Truncation: the reverse learning procedure can estimate the exact hyper-gradient of the target model in the ideal situation when all information of the target model is obtained. However, due to the limited knowledge, the reverse learning procedure may not estimate the hyper-gradient accurately. The error may be accumulated, which may cause the

gradient explosion problem and lead to failure. We propose a learning stopping criterion that stops the reverse learning procedure when the loss is larger than a pre-defined threshold. One possible value of the threshold is the loss of the random guess. The gradient explosion problem is visualized in Section 4.2.4.

Re-training Interval: the time complexity is an obvious drawback of the proposed generation algorithm. The target classifier should be retrained when adding poisoning samples into the training set. However, updating the parameters $\hat{\boldsymbol{w}}$ of the model is time-consuming, and the change may be insignificant when the modification of the training set is small. Therefore, we propose a mechanism called re-training interval to control the frequency of updates in terms of the number of poisoning samples added to the training set.

Parallel Processing: a parallel process is used to further speed up the poisoning sample generation. Our method optimizes a batch of poisoning samples (refer to the 1^{st} loop of Algorithm 1) in parallel. Although the precision may slightly drop due to the parallel process, the running time can decrease significantly.

3.7. Time Complexity

We assume that the training of neural network needs N_{tr} iterations, and N_p poisoning samples are generated. With the *re-training interval* mechanism, the poisoning attack only needs $O(N_p/N_r)$ times of the full training, which requires much less time than the previous one, *i.e.,* $O(N_p)$. The total time complexity of the generation algorithm is $O(N_p(N_{tr}/N_r + N_{out}N_{in}))$ according to Algorithm 1. By using the parallel processing, the time complexity is further reduced to $O((N_p/N_{bp})(N_{tr}/N_r + N_{out}N_{in}))$.

4. Experiments

Our attack model is evaluated and compared with the baseline method experimentally in this section. Section 4.1 describes the experimental settings. The results and discussions are given in Section 4.2.

4.1. Experimental Settings

All experiments are carried out ten independent runs, and the average performance is reported. We implement our method with Pytorch [34], and the experiments are run on a personal computer with the GeForce RTX 2080 Ti GPU.

Datasets: MNIST [17], CIFAR-10 [16], CIFAR-100 [16] and Mini-ImageNet [42] are used in our experiments. 1,000 samples are randomly selected for training, validation, and test sets in each experiment on MNIST. *Airplane* and *automobile* are used on CIFAR-10. Similarly, 1000 samples are randomly selected for training, validation, and test sets. For both CIFAR-100 and Mini-ImageNet, 60,000 images are randomly chosen from 100 classes (*i.e.,* each class has 600 images), including 40,000 images for training, 10,000 images for validation, and 10,000 images for testing.

Attack Setting: both the *indiscriminate attack* and the *targeted attack* are considered. In the *indiscriminate attack*, the proposed method aims to reduce the robustness of a model against all adversarial samples, while in the *targeted attack*, we consider the robustness against adversarial samples of a target class, which is randomly chosen. The attack ratios are set as 5% and 0.5% of the training set for *indiscriminate* and *targeted attacks*, respectively. We assume the attacker has full knowledge of the target model, including the model and training samples. In addition, the limited knowledge scenario is also considered in Section 4.2.3.

Regarding our method, we set $h = 10^{-6}$ for MLP and $h = 10^{-8}$ for LeNet and VGG to estimate the Hessian-vector product. The re-training interval N_r is 16 for MNIST and CIFAR-10, while $N_r = 512$ for CIFAR-100 and Mini-ImageNet. The numbers of iterations in the outer and inner loops are $N_{out} = 10$ and $N_{in} = N_{epoch}$ respectively, where N_{epoch} is the number of training epochs of the target model. The loss threshold η is $\ln 10$. We set $\beta = 1.0$ on MNIST, CIFAR-10, CIFAR-100 and $\beta = 7.0$ on Mini-ImageNet.

The random flip attack (RAN) is used as the baseline method due to its effectiveness shown in the previous studies [28,37,45]. RAN misleads the training of a model by randomly flipping the label of training samples to an incorrect one.

Target Models: two models, *i.e.,* LeNet [17] and VGG [41], are considered as the target model. Preliminary experiments aiming to minimize the loss were carried out in order to determine the training parameters. The mini-batch stochastic gradient descent (MBGD) has been used as the optimizer for training, and the batch size N_B is set as 16. The number of training epochs N_{epoch} is set as 35 and 20, and the learning rate λ is set as 0.1 and 0.01 for LeNet and VGG respectively on MNIST. N_{epoch} is set as 10 and 40, and λ as 0.01 and 0.1 for LeNet and VGG on CIFAR-10. We set $N_{epoch} = 50, \lambda = 0.001$ and $N_B = 64$ for VGG on CIFAR-100, and $N_{epoch} = 20, \lambda = 0.001$ and $N_B = 32$ for VGG on Mini-ImageNet. It should be noted that no data augmentation method or pre-trained model is used.

Evaluation: two measurements are used to evaluate the effectiveness of the proposed attack, *i.e.,* attack success rate (ASR) and classification accuracy (ACC). ACC refers to the accuracy of the contaminated model on a clean sample set, while ASR is defined as the ratio of adversarial samples that successfully mislead the target model under limited modification on test samples. Only the samples correctly predicted by the target model are manipulated in the attack. Two benchmark adversarial attacks, *i.e., FGSM* and *PGD*, with l_∞ are considered in ASR calculation. These attacks are implemented using the Pytorch

library *torchattacks* [14], and their parameters are listed in Table 2. A higher ASR value indicates less robustness to adversarial attack, while a higher ACC value means the poisoning attack is better in terms of stealthiness.

4.2. Results and Discussions

4.2.1. Robustness Measurements

The nine robustness measures, including gradient-norm-based ($S_{l_1}, S_{l_2}, S_{l_\infty}$), score-based ($S^g_{max}(\boldsymbol{x}), S^g_{diff}(\boldsymbol{x}), S^g_{avg}(\boldsymbol{x})$) and logits-based ($S^f_{max}(\boldsymbol{x}), S^f_{diff}(\boldsymbol{x}), S^f_{avg}(\boldsymbol{x})$) measurements defined in Section 3.4 are evaluated experimentally. The results of LeNet contaminated by our attack method using different robustness measurements on MNIST are shown in Table 3. The clean model in the first row serves as the baseline for comparison.

The performances of all methods are consistent in both indiscriminate and targeted attacks. The score-based measurements S^g have the best performance compared with gradient-norm-based S_l and logits-based S^f measurements generally. Although the average loss-gradient norm is regarded as a crucial component of adversarial vulnerability [40], the performance of S^f based measurements is not effective as S^g. In addition, S^g is bounded (*i.e.,* its range is from 0 to 1) but not S^f. Thus, minimizing logit differences may not necessarily lead to smaller score difference between classes. The calculation error of the gradient of the loss-gradient norm with respect to the model parameters (*i.e.,* $||\nabla_{\boldsymbol{x}}\mathcal{L}(\boldsymbol{w}', \boldsymbol{x})||_q$) may be accumulated in the back-gradient optimization. This might be the reason why the performance of S_l is worse than S^g.

The robustness measurement calculating the difference between the largest values (S^g_{diff} and S^f_{diff}) is better than other types (*i.e.,* S^g_{max}, S^g_{avg} and S^f_{max}, S^f_{avg}). The reason might be that more precise information is considered in the calculation, and the difference is directly related to the adversarial cost. Since S^g_{diff} has the best performance, it serves as the robustness measurement in the following experiments.

4.2.2. Attack Ability and Stealthiness

The performance of our proposed is evaluated and compared with RAN in terms of stealthiness and attack ability. Figs. 1 and 2 show the performance of the attack methods with different attack ratios, *i.e.,* {0%, 2%, 4%, 6%, 8%, 10%} and {0%, 0.2%, 0.4%, 0.6%, 0.8%, 1.0%} for indiscriminate and targeted attacks, respectively. The x-axis denotes the attack ratio, while the y-axes denote ASR and ACC in the upper and lower rows, respectively. Our method and RAN attack are represented by red and black lines. Two adversarial attacks, FGSM and PGD, are denoted by solid and dotted lines.

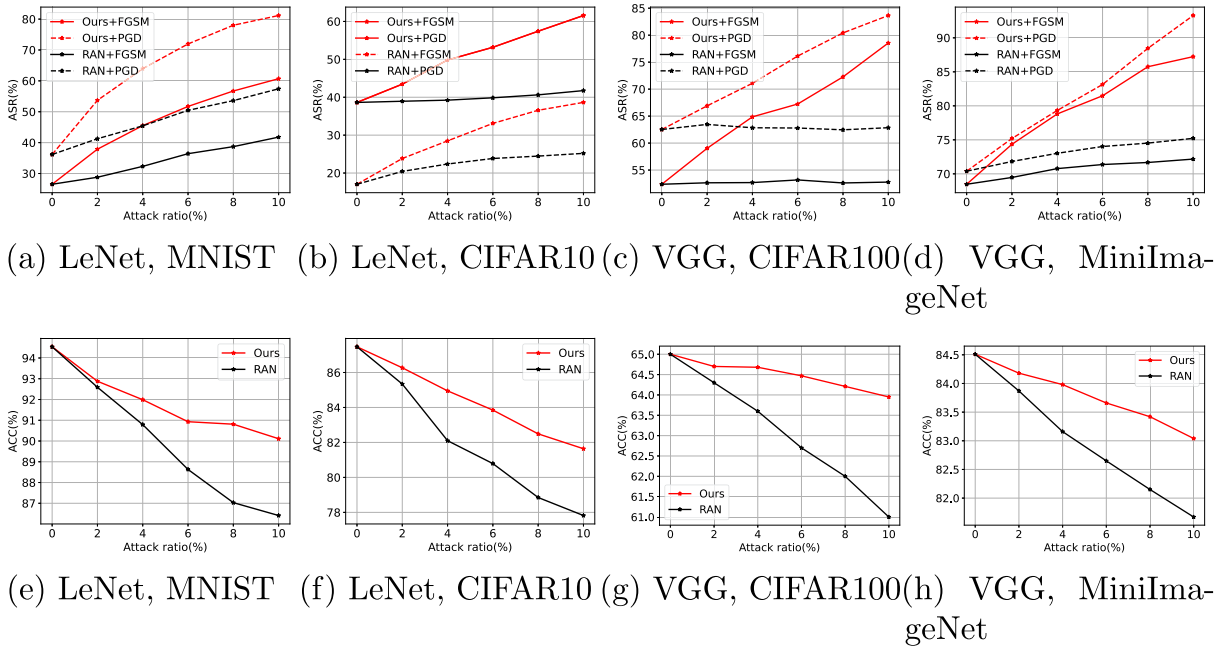
The results confirm that our attack method significantly decreases the robustness of the target model. In the indiscriminate attack, the ASR values increase around 35%, 20%, 20%, 20% when the attack ratios increase from zero (*i.e.,* a clean model)

Table 2
The parameter settings of adversarial attacks on MNIST, CIFAR-10, CIFAR-100 and Mini-ImageNet.

Datasets	FGSM	PGD
MNIST	ϵ=0.1	ϵ=0.1; α=0.015; steps=10
CIFAR-10	ϵ=0.025	ϵ=0.01; α=0.002; steps=10
CIFAR-100	ϵ=0.004	ϵ=0.004; α=0.001; steps=10
Mini-ImageNet	ϵ=0.004	ϵ=0.004; α=0.001; steps=10

Table 3
ACC and ASR of LeNet contaminated by our proposed poisoning attacks with different robustness measures on MNIST. "Clean" indicates the model is trained on a clean training set.

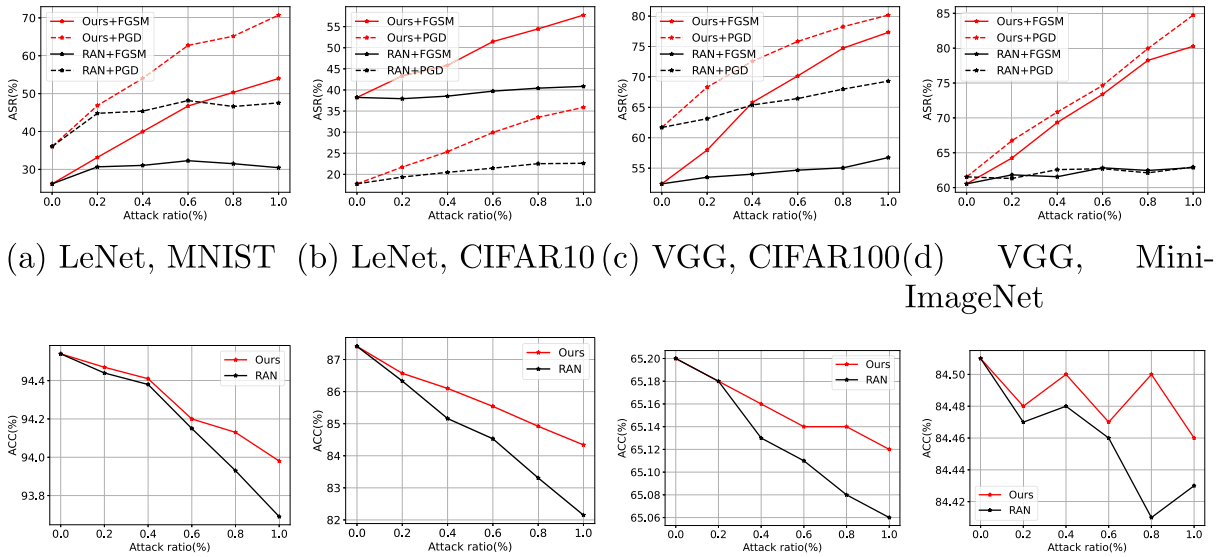
	Indiscriminate						Targeted					
	ASR(%)		ACC(%)	ASR(%)		ACC(%)	ASR(%)		ACC(%)	ASR(%)		ACC(%)
	FGSM-Inf	PGD-Inf		FGSM-L2	PGD-L2		FGSM-Inf	PGD-Inf		FGSM-L2	PGD-L2	
Clean	26.52	36.16	94.54	33.77	24.97	94.54	26.18	36.05	94.54	32.78	24.52	94.54
S_{l_1}	49.37	67.84	91.47	60.06	54.28	91.47	42.12	56.12	94.22	57.23	43.40	94.22
S_{l_2}	49.00	67.94	**91.56**	60.40	53.83	**91.56**	43.08	56.22	94.38	57.05	42.46	94.38
S_{l_∞}	48.05	67.25	91.16	58.54	52.65	91.16	43.38	57.24	94.25	57.32	43.06	94.25
S^f_{max}	46.61	65.78	91.26	55.56	51.39	91.26	41.25	55.55	94.21	54.57	42.25	94.21
S^f_{diff}	46.54	67.74	91.02	56.58	54.29	91.02	42.19	56.24	94.44	55.14	42.22	94.44
S^f_{avg}	46.17	65.51	91.45	54.72	51.36	91.45	40.76	55.58	94.17	54.69	41.43	94.17
S^g_{max}	49.09	67.73	91.44	58.78	52.62	91.44	43.64	57.99	94.26	58.80	44.32	94.26
S^g_{diff}	**49.97**	**68.61**	91.33	**60.09**	**54.35**	91.33	**45.10**	**58.24**	94.27	**59.59**	**45.64**	94.27
S^g_{avg}	49.42	68.51	91.52	60.06	54.02	91.52	40.40	57.09	**94.45**	56.27	43.78	**94.45**



(a) LeNet, MNIST (b) LeNet, CIFAR10 (c) VGG, CIFAR100 (d) VGG, MiniImageNet

(e) LeNet, MNIST (f) LeNet, CIFAR10 (g) VGG, CIFAR100 (h) VGG, MiniImageNet

Fig. 1. Comparison of the proposed method and RAN in the indiscriminate attack.



(a) LeNet, MNIST (b) LeNet, CIFAR10 (c) VGG, CIFAR100 (d) VGG, MiniImageNet

(e) LeNet, MNIST (f) LeNet, CIFAR10 (g) VGG, CIFAR100 (h) VGG, MiniImageNet

Fig. 2. Comparison of the proposed method and RAN in targeted attack.

to 10% in LeNet-MNIST, LeNet-CIFAR10, VGG-CIFAR100 and VGG-MiniImageNet. On the other hand, the ACC values of the models contaminated by our method only drop around 4%, 6%, 1%, and 1% in those settings. Similar observations, *i.e.,* the significant increase of ASR and a slight drop in ACC, have been made in the targeted attack, These results confirm that our method guarantees not only the attack ability but also stealthiness in both attack settings.

RAN attack also increases the vulnerability of the contaminated models in all situations, except VGG-CIFAR100 with the indiscriminate attack. Its influence is much less than our method since the attack samples of RAN are generated randomly

and independently to the target model. Moreover, the ACC values of the contaminated model by RAN attack drop more than our method obviously. This demonstrates the effectiveness of our attack model.

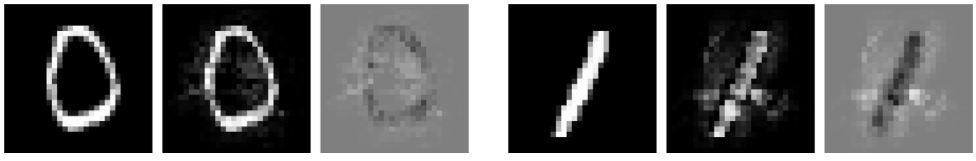
The average generation times of a poisoning sample in our method are 7.06s, 27.3s, 8.3s and 29.4s in LeNet-MNIST, LeNet-CIFAR10, VGG-CIFAR100, and VGG-MiniImageNet. It should be noted that the parallel process is only applied to VGG-CIFAR100 and VGG-MiniImageNet. Although the running time of our method is much longer than RAN, poisoning samples can be generated offline.

Two poisoning samples generated by our attack are randomly selected as examples shown in Fig. 3. For each sample, the left, middle, and right images denote the original image, the poisoning image, and their difference. Different from a backdoor attack, no trigger is used in our method. As a result, no similar simple pattern can be observed from the contaminated images. The adversarial manipulation is less significant when the dataset is more complicated, *e.g.,* the change on MNIST is larger compared to the one on Mini-ImageNet.

The vulnerability of a model contaminated by our attack method is demonstrated in another way. The maximum perturbations of the adversarial attacks (*i.e.,* FGSM and PGD) are set from 0.2 to 2.0 with an interval of 0.2. The results of the indiscriminate attacks with different attack ratios on MNIST are shown in Fig. 4. When the perturbation limitation is small, the ASR increases slowly along with the poisoning ratio. The attack performance is not obvious when the manipulation ability is weak. For example, when the perturbation limitations are 0.2 and 0.4, the ASR values remain stable in all cases. ASR rapidly increases with the increase of the poisoning ratio when the perturbation limitation is larger. However, the larger perturbation limitation does not necessarily mean larger increase of ASR. For example, the increase rates of ASR of limitation 1.0 are larger than the corresponding ones using 2.0 in LeNet-PGD, VGG-FGSM, and VGG-PGD. The reason might be that most samples can be easily attacked when a large perturbation limitation is used. The proposed poisoning attack can significantly increase the ASR values when the maximum perturbation of the adversarial attack is limited.

4.2.3. Attacks with Different Knowledge

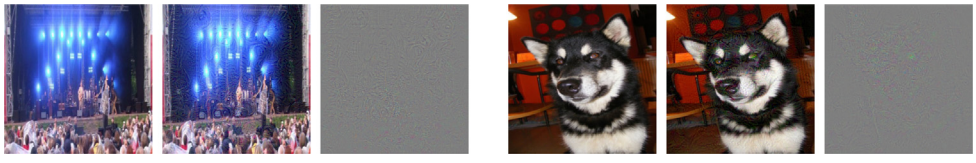
The performance of our model is evaluated experimentally in the scenarios with different levels of attack knowledge, and the results on MNIST are shown in Table 4. The limited knowledge scenarios with both the model and dataset (LK-MD), with only the model (LK-M), and with only the dataset (LK-D). Different from full knowledge (FK), the training parameters of the



(a) LeNet-MNIST

(b) LeNet-CIFAR-10

(c) VGG-CIFAR-100



(d) VGG-Mini-ImageNet

Fig. 3. Poisoning samples generated by our attack model in different settings. The original and poisoning images are shown at left and middle, while their difference is shown at right.

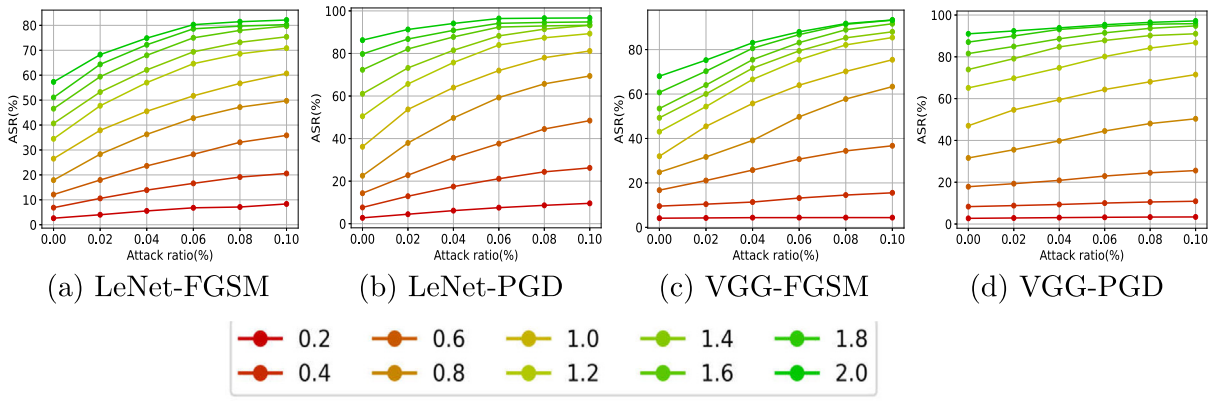


Fig. 4. ASR of the models (LeNet and VGG) under the indiscriminate adversarial attacks (FGSM and PGD) with different perturbation limitations represented by different colors on MNIST.

Table 4
The ASR of adversarial attacks (FGSM and PGD) on the models (LeNet and VGG) contaminated by our indiscriminate attacks with different levels of knowledge on MNIST.

Model	LeNet		VGG	
	FGSM	PGD	FGSM	PGD
LK-MD	49.97	68.61	56.16	60.74
LK-M	48.14	69.42	57.32	60.63
LK-D	49.90	67.36	52.47	56.18
NK	50.18	67.74	52.61	56.97

model are unknown to an adversary in LK-MD. A surrogate model and surrogate training data are used in LK-D and LK-M, respectively. An adversary basically has no information in the No Knowledge (NK) scenario. Only the input and output domains are provided. A surrogate model and surrogate training data are used. In our experiments, when VGG is served as the target model, LeNet is used as its surrogate model, and vice versa. The same amount of training samples are randomly selected as surrogate training data. 5% samples are contaminated.

The average ASR values are 58.87%, 58.87%, 56.48%, and 56.88% in LK-DM, LK-M, LK-D, and LK scenarios, respectively, which indicates that the influence of knowledge on data is insignificant to the result. Although the adversary does not have the training set used by the target model, a surrogate training set provides enough information to learn a similar decision plane to the target model. On the other hand, missing the model's information slightly downgrades the performance. By comparing LK-DM and LK-D, and LK-M and NK, the ASR differences are around 2%. It is because the behaviors of different classifiers may vary. These results indicate that the transferability of our method is good. Our attack performance drops by less than 4% due to lack of knowledge of the target model.

4.2.4. Ablation Studies

The optimization problem of our attack model is solved by the back-gradient optimization (BGO) with the proposed *re-training interval* and *gradient truncation* mechanisms. We consider our original method as well as its three ablated methods in ablation studies, including BGO+T+I, BGO+T, BGO+I, and BGO, representing our method with all functions, our method with the *gradient truncation* only, with the *re-training interval* only, and with no proposed mechanism in the back-gradient optimization respectively. The ASR and running time (in seconds) of generating a poisoning sample are used in evaluation.

Table 5 shows the results of the indiscriminate attack on MNIST. Due to the instability of the *reverse learning procedure*, no result of LeNet and VGG without the *gradient truncation* can be obtained. A simple example is used to visualize the gradient explosion problem. Fig. 5 shows the change of the loss in training. The blue lines represent the tendency of training loss along with the standard training procedure, while the green lines show the ones of the *reverse training procedure*. The result shows that the *reverse training loss* increases dramatically at a certain step, *i.e.,* around 15 to 16, in our experiments. The explosion problem rarely happens in MLP. It may be because the error caused by the randomness is accumulated more easily in deep models. As a result, we avoid the gradient explosion problem by proposing the *gradient truncation* mechanism. The blue dashed line in the figure represents the loss threshold of the *truncate gradient* mechanism. The training is terminated when the loss value is larger than the threshold. As a result, the *gradient truncation* mechanism enables the attack on DNNs, *i.e.,* LeNet and VGG. Moreover, by comparing BGO and BGO+T, and BGO+I and BGO+T+I, ASR increases when using the *gradient truncation* mechanism since the improper gradients accumulated in the last stage of the *reverse learning procedure* are dis-

Table 5
Comparison of the back-gradient optimization algorithms (BGO) with our proposed *gradient truncation* (T) and *re-training interval* (I) in terms of ASR (%) and the average generation time (in second) of each poisoning sample.

Model	MLP			LeNet			VGG		
	ASR(%)		Time	ASR(%)		Time	ASR(%)		Time
	FGSM	PGD		FGSM	PGD		FGSM	PGD	
BGO	81.93	95.51	29.10	/	/	/	/	/	/
BGO+T	86.36	96.93	28.24	59.69	80.42	50.94	72.37	68.55	353.61
BGO+I	88.65	96.72	4.33	/	/	/	/	/	/
BGO+T+I (Ours)	90.30	97.13	4.08	61.49	82.92	7.06	75.43	71.50	47.76

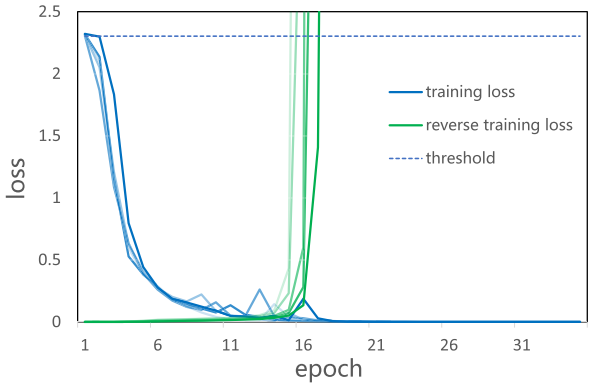


Fig. 5. Losses of different standard training procedures (blue lines) and *reverse learning procedures* (green lines) of LeNet on MNIST.

carded. The *re-training interval* mechanism largely reduces the average generation time of each poisoning sample (around 24s) by comparing BGO and BGO+I, and BGO+T and BGO+T+I. In addition, ASR is also improved by using the *re-training interval* mechanism. It may be because frequent update yields instability.

4.2.5. Robustness Evaluation Strategies

The robustness of our model is calculated by using correctly classified samples. This section evaluates the effectiveness of using only correctly classified samples rather than all samples in the robustness calculation. Fig. 6 shows the results of LeNet attacked by our method using only correctly classified samples and all samples on MNIST. 10% and 1% attack ratios are chosen for indiscriminate and targeted attacks.

The results confirm that our model with the robustness term measured by using correctly classified validation samples performs better then the one using all samples. The methods using only the correctly classified samples achieve 3-4% and 2-

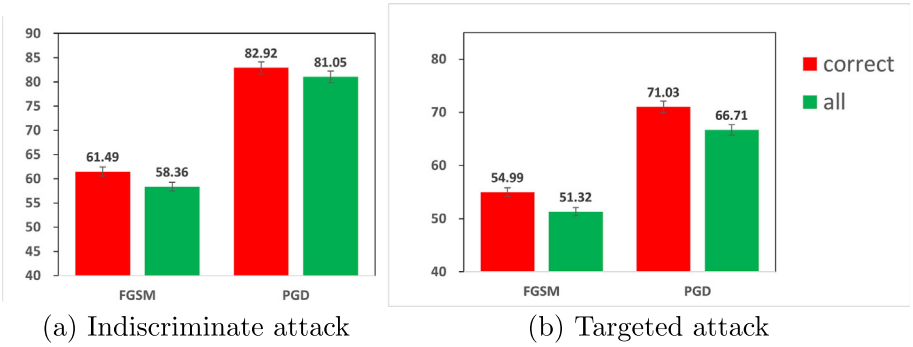


Fig. 6. The ASR of adversarial attacks (FGSM and PGD) on LeNet contaminated by our indiscriminate attacks with the robustness estimated by using only correctly classified samples and all samples on MNIST.

3% higher in ASR than the ones using all samples in the targeted and the indiscriminate attack, respectively. The incorrectly classified samples may mislead the quantification of robustness.

5. Conclusion

This paper proposes a poisoning attack that not only reduces the robustness of a target model against adversarial samples, but also explicitly improves its concealment. The problem is formulated as a bi-level optimization problem and is solved by the reverse learning procedure. Re-training interval and parallel process mechanisms are proposed to simplify and speed up the generation process of poisoning samples. Gradient truncation mechanism is devised to deal with the gradient explosion problem of the reverse learning procedure in deep neural networks. The experimental results confirm the attack ability and stealthiness of the proposed method. The transferability of our method is also demonstrated in scenarios with different levels of knowledge on the target models. This study provides a new perspective on a poisoning attack and advances the understanding of DNNs' vulnerability.

Reducing the running time of the poisoning sample generation can be one of the future works. Generating a group of attack samples in each step may speed up the progress, and an approximation on the training of the target model can also be considered. Another possible work is to investigate whether and how the poisoning samples generated by our method can be detected. The data distribution of the contaminated training sets can be analyzed and compared with the untainted ones. The classes may be closer to each other in the contaminated sets.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work is supported by Key Platform, Research Project of Education Department of Guangdong Province (No.2020KTSCX132), Guangdong Basic and Applied Basic Research Foundation (Nos.2021A1515012138, 2019A1515011166) and the National Natural Science Foundation of China (Nos.61802061, 61972091).

Appendix A. Details of the Back-Gradient Algorithm

This section provides a detailed explanation of the back-gradient algorithm [11] used to estimate the hyper-gradient of the model's robustness S to a training sample $\boldsymbol{x_p}$ (*i.e.*, $\nabla_{\boldsymbol{x}_p} S$). The detailed process is shown in Algorithm 2. If a neural network is trained by the standard gradient descent algorithm, the model parameters $\boldsymbol{w}$ are updated by

Algorithm 2: Back-Gradient Algorithm

Input: $\mathcal{D}_{tr}$: training set; $\mathcal{D}_{val}$: validation set; $\hat{\boldsymbol{w}}$: the parameters of a trained model; $\mathcal{L}$: learning objective (*i.e.*, training loss); S: robustness measurement; N: number of iterations; λ: learning rate.

Output: Hyper-gradient $\nabla_{\boldsymbol{x}_p} S$

$d\boldsymbol{x}_p = \mathbf{0}$, $d\boldsymbol{w} = \nabla_{\boldsymbol{w}} S(\mathcal{D}_{val}, \boldsymbol{w}_N)$, $\boldsymbol{w}_N = \hat{\boldsymbol{w}}$;
for $j = N$ *to* 1 **do**
  $\boldsymbol{w}_{j-1} = \boldsymbol{w}_j + \lambda \nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w_j}, \mathcal{D}_{tr})$;
  $d\boldsymbol{x}_p = d\boldsymbol{x}_p - \lambda (\nabla_{\boldsymbol{x}_p} \nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w_{j-1}}, \mathcal{D}_{tr}))^T d\boldsymbol{w}$;
  $d\boldsymbol{w} = d\boldsymbol{w} - \lambda (\nabla_{\boldsymbol{w}} \nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w_{j-1}}, \mathcal{D}_{tr}))^T d\boldsymbol{w}$;
end
$\nabla_{\boldsymbol{x}_p} S = d\boldsymbol{x}_p$;
return $\nabla_{\boldsymbol{x}_p} S$;

$$\boldsymbol{w}_i = \boldsymbol{w}_{i-1} - \lambda \frac{\partial \mathcal{L}(\boldsymbol{w}_{i-1}, \mathcal{D}_{tr})}{\partial \boldsymbol{w}_{i-1}}, i = 1, 2, \cdots, N, \tag{A.1}$$

where λ is the learning rate and $\mathcal{D}_{tr}$ is the training set. We define a measurement $S(\mathcal{D}_{val}, \boldsymbol{w}_N)$ to evaluate the model's robustness on the validation set $\mathcal{D}_{val}$, where $\boldsymbol{w}_N$ denote the model's parameters after training with N iterations. It should be noted that S can be other functions, *e.g.,* classification error. If the validation set $\mathcal{D}_{val}$ and the initial model parameters $\boldsymbol{w}_0$ are fixed, the model's robustness S is determined by the training data $\mathcal{D}_{tr}$. According to the chain rule, the gradient of S to a training sample $\boldsymbol{x}_p$ can be calculated by

$$\frac{\partial S}{\partial \boldsymbol{x}_p} = \sum_{i=1}^{N} \left(\frac{\partial \boldsymbol{w}_i}{\partial \boldsymbol{x}_p^T} \right)^T \frac{\partial S}{\partial \boldsymbol{w}_i} = \sum_{i=1}^{N} d\boldsymbol{x}_i, \tag{A.2}$$

where $d\boldsymbol{x}_i$ is the derivative accumulated in the i-th step. In Eq. (A.2), we expand the hyper-gradient $\frac{\partial S}{\partial \boldsymbol{x}_p}$ as the summation form. Since each term $d\boldsymbol{x}_i$ is related to the model parameters in the i-th step ($\boldsymbol{w}_i$), we can obtain the model parameters in all training steps and then calculate $d\boldsymbol{x}_i$.

Because the model parameters of the trained model $\boldsymbol{w}_N$ are known, the last term in Eq. (A.2) ($d\boldsymbol{x}_N$) can be obtained easily. To calculate other cumulative terms, we only need to obtain the relationship between $d\boldsymbol{x}_i$ and $d\boldsymbol{x}_{i-1}$. Unfortunately, there is no explicit relationship between them. Therefore, we further expand the multiplication terms of $d\boldsymbol{x}_i$. Utilizing Eq. (A.1), the derivatives of $\boldsymbol{w}_i$ with respect to $\boldsymbol{w}_{i-1}$ and $\boldsymbol{x}_p$ can be calculated by

$$\begin{aligned} \frac{\partial \boldsymbol{w}_i}{\partial \boldsymbol{w}_{i-1}^T} &= \frac{\partial \boldsymbol{w}_{i-1}}{\partial \boldsymbol{w}_{i-1}^T} - \lambda \frac{\partial^2 \mathcal{L}(\boldsymbol{w}_{i-1}, \boldsymbol{x}_p)}{\partial \boldsymbol{w}_{i-1} \partial \boldsymbol{w}_{i-1}^T} \\ &= \boldsymbol{I} - \lambda \frac{\partial^2 \mathcal{L}(\boldsymbol{w}_{i-1}, \boldsymbol{x}_p)}{\partial \boldsymbol{w}_{i-1} \partial \boldsymbol{w}_{i-1}^T}, \end{aligned} \tag{A.3}$$

$$\begin{aligned} \frac{\partial \boldsymbol{w}_i}{\partial \boldsymbol{x}_p^T} &= \frac{\partial \boldsymbol{w}_{i-1}}{\partial \boldsymbol{x}_p^T} - \lambda \frac{\partial^2 \mathcal{L}(\boldsymbol{w}_{i-1}, \boldsymbol{x}_p)}{\partial \boldsymbol{w}_{i-1} \partial \boldsymbol{x}_p^T} \\ &= -\lambda \frac{\partial^2 \mathcal{L}(\boldsymbol{w}_{i-1}, \boldsymbol{x}_p)}{\partial \boldsymbol{w}_{i-1} \partial \boldsymbol{x}_p^T}, \end{aligned} \tag{A.4}$$

where $\boldsymbol{I}$ is the identity matrix.

By applying Eq. (A.4), $d\boldsymbol{x}_N$ can be further expanded as

$$\begin{aligned} d\boldsymbol{x}_N &= \left(\frac{\partial \boldsymbol{w}_N}{\partial \boldsymbol{x}_p^T} \right)^T \frac{\partial S}{\partial \boldsymbol{w}_N} \\ &= -\lambda \left(\frac{\partial^2 \mathcal{L}(\boldsymbol{w}_{N-1}, \boldsymbol{x}_p)}{\partial \boldsymbol{w}_{N-1} \partial \boldsymbol{x}_p^T} \right)^T \frac{\partial S}{\partial \boldsymbol{w}_N} \\ &= -\lambda \left(\frac{\partial^2 \mathcal{L}(\boldsymbol{w}_{N-1}, \boldsymbol{x}_p)}{\partial \boldsymbol{w}_{N-1} \partial \boldsymbol{x}_p^T} \right)^T d\boldsymbol{w}_N. \end{aligned} \tag{A.5}$$

We denote $\frac{\partial S}{\partial \boldsymbol{w}_i}$ as $d\boldsymbol{w}_i$ for notation simplicity.

By reversing the update rule in Eq. (A.1), we have

$$\boldsymbol{w}_{i-1} = \boldsymbol{w}_i + \lambda \frac{\partial \mathcal{L}(\boldsymbol{w}_i, \boldsymbol{x})}{\partial \boldsymbol{w}_i}, i = 1, 2, \cdots, N. \tag{A.6}$$

Then we can calculate the model parameters in all training steps on the fly instead of storing them in the memory. Similarly to $d\boldsymbol{x}_N$, we can expand $d\boldsymbol{x}_{N-1}$ as

$$\begin{aligned} d\boldsymbol{x}_{N-1} &= \left(\frac{\partial \boldsymbol{w}_{N-1}}{\partial \boldsymbol{x}_p^T} \right)^T d\boldsymbol{w}_{N-1} \\ &= -\lambda \left(\frac{\partial^2 \mathcal{L}(\boldsymbol{w}_{N-2}, \boldsymbol{x}_p)}{\partial \boldsymbol{w}_{N-2} \partial \boldsymbol{x}_p^T} \right)^T d\boldsymbol{w}_{N-1} \\ &= -\lambda \left(\frac{\partial^2 \mathcal{L}(\boldsymbol{w}_{N-2}, \boldsymbol{x}_p)}{\partial \boldsymbol{w}_{N-2} \partial \boldsymbol{x}_p^T} \right)^T \left(\frac{\partial \boldsymbol{w}_N}{\partial \boldsymbol{w}_{N-1}} \right)^T d\boldsymbol{w}_N \\ &= -\lambda \left(\frac{\partial^2 \mathcal{L}(\boldsymbol{w}_{N-2}, \boldsymbol{x}_p)}{\partial \boldsymbol{w}_{N-2} \partial \boldsymbol{x}_p^T} \right)^T \left(\boldsymbol{I} - \lambda \frac{\partial^2 \mathcal{L}(\boldsymbol{w}_{N-1}, \boldsymbol{x}_p)}{\partial \boldsymbol{w}_{N-1} \partial \boldsymbol{w}_{N-1}^T} \right)^T d\boldsymbol{w}_N \\ &= -\lambda \left(\frac{\partial^2 \mathcal{L}(\boldsymbol{w}_{N-2}, \boldsymbol{x}_p)}{\partial \boldsymbol{w}_{N-2} \partial \boldsymbol{x}_p^T} \right)^T \left(d\boldsymbol{w}_N - \lambda \left(\frac{\partial^2 \mathcal{L}(\boldsymbol{w}_{N-1}, \boldsymbol{x}_p)}{\partial \boldsymbol{w}_{N-1} \partial \boldsymbol{w}_{N-1}^T} \right)^T d\boldsymbol{w}_N \right). \end{aligned} \tag{A.7}$$

By comparing Eq. (A.5) and (A.7), we can further derive the relationship between $d\boldsymbol{w}_N$ and $d\boldsymbol{w}_{N-1}$

$$d\boldsymbol{w}_{N-1} = d\boldsymbol{w}_N - \lambda \left(\frac{\partial^2 \mathcal{L}(\boldsymbol{w}_{N-1}, \boldsymbol{x}_p)}{\partial \boldsymbol{w}_{N-1} \partial \boldsymbol{w}_{N-1}^T} \right)^T d\boldsymbol{w}_N. \tag{A.8}$$

Then, we can also derive the relationship between $d\boldsymbol{w}_{N-1}$ and $d\boldsymbol{x}_{N-1}$ by Eq. (A.7) and (A.8)

$$d\boldsymbol{x}_{N-1} = -\lambda \left(\frac{\partial^2 \mathcal{L}(\boldsymbol{w}_{N-2}, \boldsymbol{x}_p)}{\partial \boldsymbol{w}_{N-2} \partial \boldsymbol{x}_p^T} \right)^T d\boldsymbol{w}_{N-1}. \tag{A.9}$$

Combining Eq. (A.8) and (A.9), we obtain the update rule between $d\boldsymbol{x}_N$ and $d\boldsymbol{x}_{N-1}$, and then all cumulative terms $d\boldsymbol{x}$ can be calculated. We calculate the hyper-parameter $\nabla_{\boldsymbol{x}_p} S$ by adding up all cumulative terms $d\boldsymbol{x}$.

References

[1] P. Bartlett, D. Foster, M. Telgarsky, Spectrally-normalized margin bounds for neural networks, Advances in Neural Information Processing Systems 30 (2017) 6241–6250.
[2] Y. Bengio, Gradient-based optimization of hyperparameters, Neural Computation 12 (8) (2000) 1889–1900.
[3] B. Biggio, B. Nelson, P. Laskov, Poisoning attacks against support vector machines, in: International Conference on Machine Learning, 2012.
[4] E. Brewer, J. Lin, D. Runfola, Susceptibility & defense of satellite image-trained convolutional networks to backdoor attacks, Information Sciences 603 (2022) 244–261.
[5] N. Carlini, D.A. Wagner, Towards evaluating the robustness of neural networks, in: IEEE Symposium on Security and Privacy, 2017, pp. 39–57.
[6] P.P. Chan, Z.-M. He, H. Li, C.-C. Hsu, Data sanitization against adversarial label contamination based on data complexity, International Journal of Machine Learning and Cybernetics 9 (6) (2018) 1039–1052.
[7] P.P.K. Chan, F. Luo, Z. Chen, Y. Shu, D.S. Yeung, Transfer learning based countermeasure against label flipping poisoning attack, Information Sciences 548 (2021) 450–460.
[8] R. Chen, J. Chen, H. Zheng, Q. Xuan, Z. Ming, W. Jiang, C. Cui, Salient feature extractor for adversarial defense on deep neural networks, Information Sciences 600 (2022) 118–143.
[9] X. Chen, C. Liu, B. Li, K. Lu, and D. Song. Targeted backdoor attacks on deep learning systems using data poisoning. arXiv:1712.05526, 2017.
[10] X. Chen, C. Liu, Y. Zhao, Z. Jia, G. Jin, Improving adversarial robustness of bayesian neural networks via multi-task adversarial training, Information Sciences 592 (2022) 156–173.
[11] J. Domke, Generic methods for optimization-based modeling, International Conference on Artificial Intelligence and Statistics 22 (2012) 318–326.
[12] I.J. Goodfellow, J. Shlens, C. Szegedy, Explaining and harnessing adversarial examples, in: International Conference on Learning Representations, 2015.
[13] N. Kees, Y. Wang, Y. Jiang, F. Lue, P.P. Chan, Segmentation based backdoor attack detection, in: International Conference on Machine Learning and Cybernetics, 2020, pp. 298–302.
[14] H. Kim. Torchattacks: A pytorch repository for adversarial attacks. arXiv:2010.01950, 2020.
[15] P.W. Koh, P. Liang, Understanding black-box predictions via influence functions, International Conference on Machine Learning 70 (2017) 1885–1894.
[16] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. Master's thesis, University of Tront, 2009.
[17] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE 86 (11) (1998) 2278–2324.
[18] Y. Li, Y. Li, B. Wu, L. Li, R. He, S. Lyu, Invisible backdoor attack with sample-specific triggers, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2021, pp. 16463–16472.
[19] Y. Li, T. Zhai, B. Wu, Y. Jiang, Z. Li, and S. Xia. Rethinking the trigger of backdoor attack. arXiv:2004.04692, 2020.
[20] G. Liu, I. Khalil, A. Khreishah, N. Phan, A synergetic attack against neural network classifiers combining backdoor and adversarial examples, in: IEEE International Conference on Big Data, 2021, pp. 834–846.
[21] H. Liu, G. Ditzler, Data poisoning against information-theoretic feature selection, Information Sciences 573 (2021) 396–411.
[22] K. Liu, B. Dolan-Gavitt, S. Garg, Fine-pruning: Defending against backdooring attacks on deep neural networks, International Symposium on Research in Attacks, Intrusions, and Defenses 11050 (2018) 273–294.
[23] D. Maclaurin, D. Duvenaud, R.P. Adams, Gradient-based hyperparameter optimization through reversible learning, International Conference on Machine Learning 37 (2015) 2113–2122.
[24] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards deep learning models resistant to adversarial attacks. In International Conference on Learning Representations, 2018.
[25] S. Mahloujifar, D.I. Diochnos, M. Mahmoody, The curse of concentration in robust learning: Evasion and poisoning attacks from concentration of measure, Proceedings of the AAAI Conference on Artificial Intelligence 33 (2019) 4536–4543.
[26] S. Mei, X. Zhu, Using machine teaching to identify optimal training-set attacks on machine learners, in: Proceedings of AAAI Conference on Artificial Intelligence, 2015, pp. 2871–2877.
[27] S. Moosavi-Dezfooli, A. Fawzi, P. Frossard, Deepfool: A simple and accurate method to fool deep neural networks, in: IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 2574–2582.
[28] K.R. Mopuri, V. Shaj, R.V. Babu, Adversarial fooling beyond flipping the label, in: IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 3374–3382.
[29] L. Muñoz-González, B. Biggio, A. Demontis, A. Paudice, V. Wongrassamee, E.C. Lupu, F. Roli, Towards poisoning of deep learning algorithms with back-gradient optimization, in: Proceedings of ACM Workshop on Artificial Intelligence and Security, 2017, pp. 27–38.
[30] T.A. Nguyen, A. Tran, Input-aware dynamic backdoor attack, Advances in Neural Information Processing Systems 33 (2020) 3454–3464.
[31] T.A. Nguyen, A.T. Tran, Wanet - imperceptible warping-based backdoor attack, in: International Conference on Learning Representations, 2021.
[32] R. Pang, H. Shen, X. Zhang, S. Ji, Y. Vorobeychik, X. Luo, A. Liu, and T. Wang. A tale of evil twins: Adversarial inputs versus poisoned models. In Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, pages 85–99, 2020a.
[33] R. Pang, Z. Zhang, X. Gao, Z. Xi, S. Ji, P. Cheng, and T. Wang. Trojanzoo: Everything you ever wanted to know about neural backdoors (but were afraid to ask). arXiv:2012.09302, 2020b.
[34] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, Pytorch: An imperative style, high-performance deep learning library, in: Advances in Neural Information Processing Systems, 2019, pp. 8024–8035.
[35] B.A. Pearlmutter, Fast exact multiplication by the hessian, Neural Computation 6 (1) (1994) 147–160.
[36] F. Pedregosa, Hyperparameter optimization with approximate gradient, International Conference on Machine Learning 48 (2016) 737–746.

[37] E. Rosenfeld, E. Winston, P. Ravikumar, J.Z. Kolter, Certified robustness to label-flipping attacks via randomized smoothing, International Conference on Machine Learning 119 (2020) 8230–8241.
[38] H. Salman, J. Li, I.P. Razenshteyn, P. Zhang, H. Zhang, S. Bubeck, and G. Yang. Provably robust deep learning via adversarially trained smoothed classifiers. In Advances in Neural Information Processing Systems, pages 11289–11300, 2019.
[39] A. Shafahi, W.R. Huang, M. Najibi, O. Suciu, C. Studer, T. Dumitras, and T. Goldstein. Poison frogs! targeted clean-label poisoning attacks on neural networks. In Advances in Neural Information Processing Systems, pages 6106–6116, 2018.
[40] C.-J. Simon-Gabriel, Y. Ollivier, L. Bottou, B. Schölkopf, D. Lopez-Paz, First-order adversarial vulnerability of neural networks and input dimension, in: International Conference on Machine Learning, 2019, pp. 5809–5817.
[41] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, in: International Conference on Learning Representations, 2015.
[42] O. Vinyals, C. Blundell, T. Lillicrap, D. Wierstra, et al. Matching networks for one shot learning. Advances in Neural Information Processing Systems, 29, 2016.
[43] C.-H. Weng, Y.-T. Lee, S.-H.B. Wu, On the trade-off between adversarial and backdoor robustness, Advances in Neural Information Processing Systems 33 (2020) 11973–11983.
[44] H. Xiao, B. Biggio, G. Brown, G. Fumera, C. Eckert, F. Roli, Is feature selection secure against training data poisoning?, International Conference on Machine Learning 37 (2015) 1689–1698
[45] H. Xiao, H. Xiao, C. Eckert, Adversarial label flips attack on support vector machines, European Conference on Artificial Intelligence 242 (2012) 870–875.
[46] C. Yang, Q. Wu, H. Li, and Y. Chen. Generative poisoning attack method against neural networks. arXiv:1703.01340, 2017.
[47] Y. Yang, R. Khanna, Y. Yu, A. Gholami, K. Keutzer, J.E. Gonzalez, K. Ramchandran, M.W. Mahoney, Boundary thickness and robustness in learning models, Advances in Neural Information Processing Systems 33 (2020) 6223–6234.
[48] F. Zhang, P.P. Chan, B. Biggio, D.S. Yeung, F. Roli, Adversarial feature selection against evasion attacks, IEEE Transactions on Cybernetics 46 (3) (2015) 766–777.
[49] P. Zhao, P.-Y. Chen, P. Das, K.N. Ramamurthy, and X. Lin. Bridging mode connectivity in loss landscapes and adversarial robustness. In International Conference on Learning Representations, 2020.
[50] L. Zhu, R. Ning, C. Xin, C. Wang, H.Wu. Clear, Clean-up sample-targeted backdoor in neural networks, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2021, pp. 16453–16462.