Adversarial Neuron Pruning Purifies Backdoored Deep Models

Dongxian Wu*
Dept. of Computer Science and Technology
Tsinghua University
wudx16@gmail.com

Yisen Wang†
Key Lab. of Machine Perception (MoE)
School of EECS, Peking University
yisen.wang@pku.edu.cn

Abstract

As deep neural networks (DNNs) are growing larger, their requirements for computational resources become huge, which makes outsourcing training more popular. Training in a third-party platform, however, may introduce potential risks that a malicious trainer will return backdoored DNNs, which behave normally on clean samples but output targeted misclassifications whenever a trigger appears at the test time. Without any knowledge of the trigger, it is difficult to distinguish or recover benign DNNs from backdoored ones. In this paper, we first identify an unexpected sensitivity of backdoored DNNs, that is, they are much easier to collapse and tend to predict the target label on clean samples when their neurons are adversarially perturbed. Based on these observations, we propose a novel model repairing method, termed *Adversarial Neuron Pruning* (ANP), which prunes some sensitive neurons to purify the injected backdoor. Experiments show, even with only an extremely small amount of clean data (*e.g.*, 1%), ANP effectively removes the injected backdoor without causing obvious performance degradation. Our code is available at `https://github.com/csdongxian/ANP_backdoor`.

1 Introduction

In recent years, deep neural networks (DNNs) achieve satisfactory performance in many tasks, including computer vision [15], speech recognition [45], and gaming agents [36]. However, their success heavily relies on a large amount of computation and data, forcing researchers to outsource the training in "Machine Learning as a Service" (MLaaS) platforms or download pretrained models from the Internet, which brings potential risks of training-time attacks [17, 35, 2]. Among them, backdoor attack [14, 6, 42] is remarkably dangerous because it stealthily builds a strong relationship between a trigger pattern and a target label inside DNNs by poisoning a small proportion of the training data. As a result, the returned model always behaves normally on the clean data but is controlled to make target misclassification by presenting the trigger pattern such as a specific single-pixel [41] or a black-white checkerboard [14] at the test time.

Since DNNs are deployed in many real-world and safety-critical applications, it is urgent to defend against backdoor attacks. While there are many defense methods during training [38, 41, 8, 29, 20], this work focuses on a more realistic scenario in outsourcing training that repairs models after training. In particular, the defenders try to remove the injected backdoor without access to the model training process. Without knowledge of the trigger pattern, previous methods only achieve limited robustness [7, 24, 22]. Some works try to reconstruct the trigger [43, 5, 33, 50, 44], however, the trigger pattern in brand-new attacks becomes natural [26], invisible [49], and dynamic [30], leading

*Work was done as an internship at Peking University when he was a student at Tsinghua University. Now, he is a Post-doc at the University of Tokyo.

†Corresponding author: Yisen Wang (yisen.wang@pku.edu.cn).

35th Conference on Neural Information Processing Systems (NeurIPS 2021).

to the reconstruction infeasible. Different from them, in this paper, we turn to explore *whether we can successfully remove the injected backdoor even without knowing the trigger pattern*.

Usually, the adversary perturbs inputs to cause misclassification, *e.g.*, attaching triggers for backdoored models or adding adversarial perturbations. Parallel to this, we are also able to cause misclassification by perturbing neurons of DNNs [47]. For any neuron inside DNN, we could perturb its weight and bias by multiplying a relatively small number, to change its output. Similar to adversarial perturbations, we can optimize the neuron perturbations to increase its classification loss. Surprisingly, we find that, within the same perturbation budget, backdoored DNNs are much easier to collapse and prone to output the target label than normal DNNs even without the presence of the trigger. That is, the neurons that are sensitive to adversarial neuron perturbation are strongly related to the injected backdoor. Motivated by this, we propose a novel model repairing method, named ***A**dversarial **N**euron **P**runing* (ANP), which prunes most sensitive neurons under the adversarial neuron perturbation. Since the number of neurons is much smaller than weight parameters, *e.g.*, only 4810 neurons for ResNet-18 while 11M parameters for the same model, our method can work well only based on 1% of clean data. Our main contributions are summarized as follows:

- We find that through adversarially perturbing neurons, backdoored DNNs can present backdoor behaviors even without the presence of the trigger patterns and are much easier to output misclassification than normal DNNs.
- To defend backdoor attacks, we propose a simple yet effective model repairing method, *Adversarial Neuron Pruning* (ANP), which prunes the most sensitive neurons under adversarial neuron perturbations without fine-tuning.
- Extensive experiments demonstrate that ANP consistently provides state-of-the-art defense performance against various backdoor attacks, even using an extremely small amount of clean data.

2 Related Work

2.1 Backdoor Attacks

The backdoor attack is a type of attacks occurring during DNN training. The adversary usually poisons a fraction of training data via attaching a predefined trigger and relabeling them as target labels (dirty-label setting) [14, 6]. All poisoned samples can be relabeled as a single target class (all-to-one), or poisoned samples from different source classes are relabeled as different target classes (all-to-all) [30]. After training, the model can be controlled to predict the target label in the presence of the trigger at the test time. Different from evasion attacks (*e.g.*, adversarial attacks) [3, 39, 13], backdoor attacks aim to embed a input- and model-agnostic trigger into the model, which is a severe threat to the applications of deep learning [12]. Since incorrectly-labeled samples are easy to be detected, some attacks attach the trigger to samples from the target class (clean-label setting) [35, 42, 1]. Apart from simple forms like a single-pixel [41] or a black-and-white checkerboard [14], the trigger patterns can be more complex such as a sinusoidal strip [1] and a dynamic pattern [30]. These triggers in recent attacks become more natural [26] and human-imperceptible [49, 31], making them stealthy and hard to be detected by human inspection. Besides, powerful adversaries who have access to the model can optimize the trigger pattern [25], and even co-optimize the trigger pattern and the model together to enhance the power of backdoor attacks [32].

2.2 Backdoor Defense

Defense during Training. With access to training, the defender is able to detect the poisoned data or make poisoning invalid during the training process. Regarding poisoned data as outliers, we can detect and filter them out using robust statistics in the input space [38, 18, 8, 9] or the feature space [17, 41, 28, 10]. Meanwhile, other studies focus on training strategies to make the poisoned data have little or no effect on the trained model [23, 40] through randomized smoothing [34, 46], majority vote [21], differential privacy [29], and input preprocessing [27, 4].

Defense after Training. For a downloaded model, we have lost control of its training. To repair the risk model, one direct way is to first reconstruct an approximation of the backdoor trigger via adversarial perturbation [43] or generative adversarial network (GAN) [5, 33, 50]. Once the trigger

is reconstructed, it is feasible to prune neurons that are activated in the presence of the trigger, or fine-tune the model to unlearn the trigger [43]. As the trigger patterns in recently proposed attacks become more complicated such as the dynamics trigger [30] or natural phenomenon-based trigger [26], reconstruction becomes increasingly difficult. There are other studies on trigger-agnostic repairing via model pruning [24] or fine-tuning [7, 22] on the clean data. While they may suffer from severe accuracy degradation when only small clean data are available [7]. Unlike previous defensive pruning methods which are based on the rule-of-thumb, our proposed method is data-driven and does not require additional fine-tuning.

3 The Proposed Method

3.1 Preliminary

Deep Neural Network and Its Neurons. In this paper, we take a fully-connected network as an example (other convolutional networks are also applicable). Its layers are numbered from 0 (input) to L (output) and each layer contains $n_0, \cdots, n_L$ neurons. The network has $n = n_1 + n_2 + \cdots + n_L$ parameterized neurons in total[3]. We denote the weight parameters of the k-th neuron in l-th layer as $\mathbf{w}_k^{(l)}$, the bias as $b_k^{(l)}$, and its output is

$$h_k^{(l)} = \sigma(\mathbf{w}_k^{(l)\top}\mathbf{h}^{(l-1)} + b_k^{(l)}), \tag{1}$$

where $\sigma(\cdot)$ is the nonlinear function and $\mathbf{h}^{(l-1)}$ is the outputs of all neurons in the previous layer, *i.e.*,$\mathbf{h}^{(l-1)} = [h_1^{(l-1)}, \cdots, h_{n_l}^{(l-1)}]$. For simplicity, we denote all weights of the network as $\mathbf{w} = [\mathbf{w}_1^{(1)}, \cdots, \mathbf{w}_{n_1}^{(1)}, \cdots, \mathbf{w}_1^{(L)}, \cdots, \mathbf{w}_{n_L}^{(L)}]$, and all biases as $\mathbf{b} = [b_1^{(1)}, \cdots, b_{n_l}^{(1)}, \cdots, b_1^{(L)}, \cdots, b_{n_L}^{(L)}]$. For a given input $\mathbf{x}$, the network makes the prediction $f(\mathbf{x}; \mathbf{w}, \mathbf{b})$.

DNN Training. We consider a c-class classification problem. The parameters of weights and biases in DNN are learned on a training dataset $\mathcal{D}_\mathcal{T} = \{(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_s, y_s)\}$ containing s inputs, where each input is $\mathbf{x}_i \in \mathbb{R}^d, i = 1, \cdots, s$, and its ground-truth label is $y_i \in \{1, \cdots, c\}$. The training procedure tries to find the optimal $(\mathbf{w}^\star, \mathbf{b}^\star)$ which minimize the training loss on the training data $\mathcal{D}_\mathcal{T}$,

$$\mathcal{L}_{\mathcal{D}_T}(\mathbf{w}, \mathbf{b}) = \mathop{\mathbb{E}}_{\mathbf{x},y\sim\mathcal{D}_T} \ell(f(\mathbf{x}; \mathbf{w}, \mathbf{b}), y), \tag{2}$$

where $\ell(\cdot, \cdot)$ is usually cross entropy loss.

Defense Setting. We adopt a typical defense setting that an untrustworthy model is downloaded from a third party (*e.g.*, outsourcing training) without knowledge of the training data $\mathcal{D}_\mathcal{T}$. For defense, we are assumed to have a small amount of clean data $\mathcal{D}_\mathcal{V}$. The goals of model repairing are to remove the backdoor behavior while keeping the accuracy on clean samples.

3.2 Adversarial Neuron Perturbations

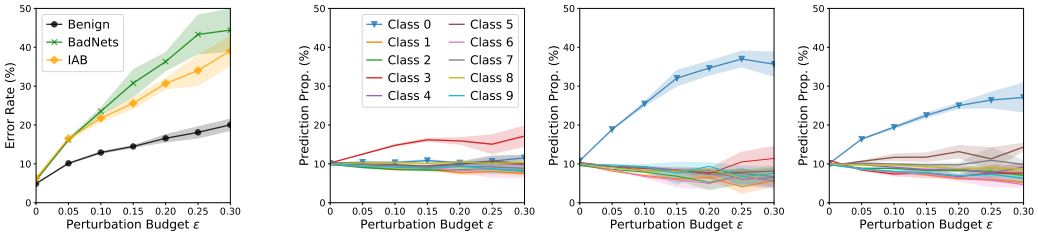
In previous studies, the adversary causes misclassification by perturbing inputs before feeding them to DNNs. For example, attaching triggers to inputs to make backdoored DNN output the target label or adversarially perturbing inputs to make normal/backdoored DNN output incorrectly.

Here, we try another direction to perturb DNN neurons to cause misclassification. Given the k-th neuron in the l-th layers, we can perturb its weight $\mathbf{w}_k^{(l)}$ and bias $b_k^{(l)}$ by multiplying a small number respectively. As a result, the perturbed weight is $(1 + \delta_k^{(l)})\mathbf{w}_k^{(l)}$, the perturbed bias is $(1 + \xi_k^{(l)})b_k^{(l)}$, and the output of the perturbed neuron becomes

$$h_k^{(l)} = \sigma\big((1 + \delta_k^{(l)})\mathbf{w}_k^{(l)\top}\mathbf{h}^{(l-1)} + (1 + \xi_k^{(l)})b_k^{(l)}\big), \tag{3}$$

where $\delta_k^{(l)}$ and $\xi_k^{(l)}$ indicate the relative sizes of the perturbations to the weight and bias respectively. Similarly, all neurons can be perturbed like Eq. (3). We denote the relative weight perturbation size of all neurons as $\boldsymbol{\delta} = [\delta_1^{(1)}, \cdots, \delta_{n_1}^{(1)}, \cdots, \delta_1^{(L)}, \cdots, \delta_{n_L}^{(L)}]$, and the relative bias perturbation

[3]The input neurons are not considered since they have no parameters.



(a) The error rate (± std) of different models under neuron perturbations.

(b) The proportion (± std over 5 random runs) of different classes in predictions by a benign model (*Left*) and two models backdoored by BadNets (*Middle*) and IAB attack (*Right*) under neuron perturbations.

Figure 1: The performance of neuron-perturbed models with different perturbation budgets.

size as $\boldsymbol{\xi} = [\xi_1^{(1)}, \cdots, \xi_{n_1}^{(1)}, \cdots, \xi_1^{(L)}, \cdots, \xi_{n_L}^{(L)}]$ for simplicity. For a clean input $\mathbf{x}$, the output of the perturbed model is

$$f(\mathbf{x}; (1+\boldsymbol{\delta}) \odot \mathbf{w}, (1+\boldsymbol{\xi}) \odot \mathbf{b}), \tag{4}$$

where neuron-wise multiplication $\odot$ multiplies the parameters by the perturbation sizes belonging to the same neuron as follows:

$$(1+\boldsymbol{\delta}) \odot \mathbf{w} = \left[(1+\delta_1^{(1)})\mathbf{w}_1^{(1)}, \cdots, (1+\delta_{n_1}^{(1)})\mathbf{w}_{n_1}^{(1)}, \cdots, (1+\delta_1^{(L)})\mathbf{w}_1^{(L)}, \cdots, (1+\delta_{n_L}^{(L)})\mathbf{w}_{n_L}^{(L)}\right], \tag{5}$$

$$(1+\boldsymbol{\xi}) \odot \mathbf{b} = \left[(1+\xi_1^{(1)})b_1^{(1)}, \cdots, (1+\xi_{n_1}^{(1)})b_{n_1}^{(1)}, \cdots, (1+\xi_1^{(L)})b_1^{(L)}, \cdots, (1+\xi_{n_L}^{(L)})b_{n_L}^{(L)}\right]. \tag{6}$$

Given a trained model, similar to adversarial perturbations [39, 13], we optimize the neuron perturbations $\boldsymbol{\delta}$ and $\boldsymbol{\xi}$ to increase the classification loss on the clean data:

$$\max_{\boldsymbol{\delta}, \boldsymbol{\xi} \in [-\epsilon, \epsilon]^n} \mathcal{L}_{\mathcal{D}_\mathcal{V}}((1+\boldsymbol{\delta}) \odot \mathbf{w}, (1+\boldsymbol{\xi}) \odot \mathbf{b}), \tag{7}$$

where ϵ is the perturbation budget, which limits the maximum perturbation size.

Specifically, we generate adversarial neuron perturbations for three ResNet-18 models [15] on CIFAR-10 [19]: a benign model, a backdoored one with a predefined trigger (Badnets [14]), and a backdoored one with a dynamic trigger (Input-aware Backdoor Attack [30], IAB). We all set the target label in backdoor as class 0. We solve the maximization in Eq. (7) using project gradient descent (PGD) with random initialization similar to generating adversarial examples. The number of iterations is 300 and the step size is 0.001. Figure 1(a) shows adversarial neuron perturbations leads to misclassification, and the larger the perturbation budget ϵ is, the larger the error rate is. In addition, backdoored models always have larger error rates with the same perturbation budget compared to the benign one. To explore the misclassification in detail, we illustrate the proportion of different classes in predictions by three models in Figure 1(b). For benign model, adversarial neuron perturbations mislead the model to output the untargeted class (*e.g.*, class 3). While for backdoored models, the majority of misclassified samples are predicted as the target label (*e.g.*, class 0) whose proportion rate is much higher than the benign model. More results to support these findings can be found in Appendix B. Therefore, even without the trigger, we can still induce the backdoor behaviours via adversarial neuron perturbations. The reasons can be explained as follows: assuming the k-th neuron in l-th layers is backdoor-related, it is dormant on clean sample [24] (*i.e.*, output is close to 0: $h_k^{(l)} \approx 0$). If $\mathbf{w}_k^{(l)\top}\mathbf{h}^{l-1} > 0$ and $b_k^{(l)} > 0$, we can increase its output by enlarging the norm of the weights (*i.e.*, $\delta_k^{(l)} > 0$), and further continue to increase it by adding a larger bias (*i.e.*, $\xi_k^{(l)} > 0$)[4]. With suitable perturbations, the backdoor-related neuron is activated even on clean samples. Thus, the neurons that are sensitive to adversarial neuron perturbation are strongly related to the injected backdoor.

3.3 The Proposed Adversarial Neuron Pruning (ANP)

As mentioned above, the sensitivity under adversarial neuron perturbations is strongly related to the injected backdoor. Inspired by this, we try to prune some sensitive neurons to defend backdoor

[4] If $\mathbf{w}_k^{(l)\top}\mathbf{h}^{l-1} < 0$ or $b_k^{(l)} < 0$, we still can increase its output by reducing the norm of the weight or bias.

attacks, named Adversarial Neuron Pruning method (ANP). We denote the pruning masks as $\mathbf{m} = [m_1^{(1)}, \cdots, m_{n_1}^{(1)}, \cdots, m_1^{(L)}, \cdots, m_{n_L}^{(L)}] \in \{0,1\}^n$. For the k-th neuron in the l-th layer, we set its weight $\mathbf{w}_k^{(l)} = \mathbf{0}$ if $m_k^{(l)} = 0$, and keep it unchanged if $m_k^{(l)} = 1$. Similar to Ghorbani and Zou [11], we always keep the bias $b_k^{(l)}$ to avoid extra harm to the accuracy on clean data[5].

Continuous Relaxation and Optimization. Due to the binary masks, pruning is a discrete optimization problem that is difficult to solve within feasible time. To address this, we apply continuous relaxation to let $\mathbf{m} \in [0,1]^n$ and optimizes them using projected gradient descent (PGD). To restore the continuous masks to discreteness after optimization, we set all mask smaller than a threshold as 0 (prune neurons with small masks), and others as 1 (keep neurons with large masks). We expect the pruned model to not only behaves well on clean data but also keep stable under adversarial neuron perturbations. Therefore, we solve the following minimization problem,

$$\min_{\mathbf{m}\in[0,1]^n} \left[\alpha \mathcal{L}_{\mathcal{D}_\mathcal{V}}(\mathbf{m} \odot \mathbf{w}, \mathbf{b}) + (1-\alpha) \max_{\boldsymbol{\delta},\boldsymbol{\xi}\in[-\epsilon,\epsilon]^n} \mathcal{L}_{\mathcal{D}_\mathcal{V}}((\mathbf{m}+\boldsymbol{\delta}) \odot \mathbf{w}, (1+\boldsymbol{\xi}) \odot \mathbf{b}) \right], \quad (8)$$

where $\alpha \in [0,1]$ is a trade-off coefficient. If α is close to 1, the minimization object focuses more on accuracy of the pruned model on clean data, while if α is close to 0, it focuses more on robustness against backdoor attacks.

Implementation of ANP. As shown in Algorithm 1, we start from an unpruned network, that is, initializing all masks as 1 (Line 2). We randomly initialize perturbations (Line 5), and update them using one step (Line 6-7), then project them onto the feasible regime (Line 8-9). Following that, we update mask values based on the adversarial neuron perturbation (Line 10) and project mask value onto $[0,1]$ (Line 11). As convergence after iterations, we pruned neurons with small mask value and keep others unchanged (Line 13). Note that the proposed ANP is data-driven, different from previous pruning-based defenses which are based on the thumb-of-rule [24, 43]. Besides, ANP can work well on an extremely small amount of clean data due to the small number of masks.

Algorithm 1 Adversarial Neuron Pruning (ANP)

1: **Input:** Network $f(\cdot; \mathbf{w}, \mathbf{b})$, hyper-parameter α, learning rate η, batch size b, maximum perturbation size ϵ
2: Initialize all elements in $\mathbf{m}$ as 1
3: **repeat**
4: Read mini-batch $\mathcal{B} = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_b, y_b)\}$ from training set
5: $\boldsymbol{\delta}_0, \boldsymbol{\xi}_0 \sim U(-\epsilon, \epsilon)$, where $U(-\epsilon, \epsilon)$ is the uniform distribution
6: $\boldsymbol{\delta} \leftarrow \boldsymbol{\delta}_0 + \epsilon \text{sign}(\nabla_{\boldsymbol{\delta}} \mathcal{L}((\mathbf{m}+\boldsymbol{\delta}_0) \odot \mathbf{w}, (\mathbf{1}+\boldsymbol{\xi}_0) \odot \mathbf{b}))$
7: $\boldsymbol{\xi} \leftarrow \boldsymbol{\xi}_0 + \epsilon \text{sign}(\nabla_{\boldsymbol{\xi}} \mathcal{L}((\mathbf{m}+\boldsymbol{\delta}_0) \odot \mathbf{w}, (\mathbf{1}+\boldsymbol{\xi}_0) \odot \mathbf{b}))$
8: $\boldsymbol{\delta} \leftarrow \max(-\epsilon, \min(\boldsymbol{\delta}, \epsilon))$
9: $\boldsymbol{\xi} \leftarrow \max(-\epsilon, \min(\boldsymbol{\xi}, \epsilon))$
10: $\mathbf{m} \leftarrow \mathbf{m} - \eta \nabla_{\mathbf{m}} [\alpha \mathcal{L}(\mathbf{m} \odot \mathbf{w}, \mathbf{b}) + (1-\alpha) \mathcal{L}((\mathbf{m}+\boldsymbol{\delta}) \odot \mathbf{w}, (\mathbf{1}+\boldsymbol{\xi}) \odot \mathbf{b})]$
11: $\mathbf{m} \leftarrow \max(0, \min(\mathbf{m}, 1))$
12: **until** training converged
13: $m_k^{(l)} = \mathbb{I}(m_k^{(l)} > threshold)$, for all k, l
14: **Output:** A robust network $f(\cdot; \mathbf{m} \odot \mathbf{w}, \mathbf{b})$ against backdoor attacks

Adaptation to Batch Normalization. Batch Normalization (BatchNorm) [16] always normalizes its input and controls the mean and variance of the output by a pair of trainable parameters (scale γ and shift β). If BatchNorm is inserted between matrix multiplication and the nonlinear activation, the perturbations to weight and bias may cancel each other out. For example, if we perturb the weight and bias of a neuron to the maximum by multiplying $(1+\epsilon)$, the normalization offsets them and nothing changes after BatchNorm. To address this problem, we perturb the scale and shift parameters instead. We also make similar changes to the pruning masks.

[5] We can still compress the model since these biases can be absorbed by their following layers.

Table 1: Performance (average over 5 random runs) of 4 defense methods against 6 backdoor attacks on 1% (500 images) of clean data on CIFAR-10 training set using ResNet-18. The *AvgDrop* indicates the average changes in ACC or ASR over 6 backdoor attacks compared to no defense results (*Before*).

Metric	Defense	Badnets	Blend	IAB-one	IAB-all	CLB	SIG	AvgDrop
ACC	Before	93.73	94.82	93.89	94.10	93.78	93.64	–
	FT($lr = 0.01$)	90.48	92.12	88.68	89.06	91.26	91.19	$\downarrow$ 3.53
	FT($lr = 0.02$)	87.23	88.98	84.85	83.77	88.25	88.63	$\downarrow$ 7.04
	FP	**92.18**	92.40	91.57	92.28	91.91	91.64	$\downarrow$ 2.00
	MCR($t = 0.3$)	85.95	88.26	86.30	84.53	86.87	85.88	$\downarrow$ 7.70
	ANP	90.20	**93.44**	**92.62**	**92.79**	**92.67**	**93.40**	$\downarrow$ **1.47**
ASR	Before	99.97	100.0	98.49	92.88	99.94	94.26	–
	FT($lr = 0.01$)	11.70	47.17	0.99	1.36	12.51	0.40	$\downarrow$ 85.24
	FT($lr = 0.02$)	2.95	10.20	1.70	1.83	**1.17**	0.39	$\downarrow$ 94.55
	FP	5.34	65.39	20.73	32.36	3.40	0.32	$\downarrow$ 76.33
	MCR($t = 0.3$)	5.70	13.57	30.23	35.17	12.77	0.52	$\downarrow$ 81.26
	ANP	**0.45**	**0.46**	**0.88**	**0.86**	3.98	**0.28**	$\downarrow$ **96.44**

4 Experiments

In this section, we conduct comprehensive experiments to evaluate the effectiveness of ANP, including its benchmarking robustness, ablation studies and performance under limited computation resources.

4.1 Experimental Settings

Backdoor Attacks and Settings. We consider 5 state-of-the-art backdoor attacks: 1) BadNets [14], 2) Blend backdoor attack (Blend) [6], 3) Input-aware backdoor attack with all-to-one target label (IAB-one) or all-to-all target label (IAB-all) [30], 4) Clean-label backdoor (CLB) [42], and 5) Sinusoidal signal backdoor attack (SIG) [1]. For fair comparisons, we follow the default configuration in their original papers such as the trigger patterns, the trigger sizes and the target labels. We evaluate the performance of all attacks and defenses on CIFAR-10 [19] using ResNet-18 [15] as the base model. We use 90% training data to train the backdoored DNNs and use the all or part of the remaining 10% training data for defense. More details about implementation can be found in Appendix A.

Backdoor Defenses and Settings. We compare our proposed ANP with 3 existing model repairing methods: 1) standard fine-tuning (FT), 2) fine-pruning (FP) [24], and 3) mode connectivity repair (MCR) [48]. All defense methods are assumed to have access to the same 1% of clean training data (500 images). The results based on 10% (5000 images) and 0.1% (50 images) of clean training data can be found in Appendix. For ANP, we optimize all masks using Stochastic Gradient Descent (SGD) with the perturbation budget $\epsilon = 0.4$ and the trade-off coefficient $\alpha = 0.2$. We set the batch size 128, the constant learning rate 0.2, and the momentum 0.9 for 2000 iterations in total. Typical data augmentation like random crop and horizontal flipping are applied. After optimization, neurons with mask value smaller than 0.2 are pruned.

Evaluation Metrics. We evaluate the performance of different defenses using two metrics: 1) the accuracy on clean data (ACC), and 2) and the attack success rate (ASR) that is the ratio of triggered samples that are misclassified as the target label. For better comparison between different strategies for the target label (*e.g.*, all-to-one and all-to-all), we remove the samples whose ground-truth labels already belong to the target class in the all-to-one setting before calculating ASR. Therefore, an ideal defense always has close-to-zero ASR and high ACC.

4.2 Benchmarking the State-of-the-art Robustness

To verify the effectiveness of the proposed ANP, we compare its performance with other 3 existing model repairing methods using ACC and ASR in Table 1. All experiments are repeated over 5 runs with different random seeds, and we only report the average performance without the standard deviation due to the space constraint. More detailed results including the standard deviation can be found in Appendix C.

Table 2: The average training time of defense method against 6 backdoor attacks on 1% (500 images) of clean data on CIFAR-10 training set using ResNet-18.

Defense	FT	FP	MCR	ANP
Time (s)	93.80	1427.1	286.1	241.5

The experimental results show that the proposed ANP remarkably provides almost the highest robustness against several state-of-the-art backdoor attacks. In particular, in 5 of total 6 attacks, our proposed ANP successfully reduces the ASR to lower than 1% while only has a slight drop ($\sim 1.47\%$ on average) in ACC. Note that we only use 1% of clean data from CIFAR-10 training set, which is extremely small. We find the standard fine-tuning (FT) with a larger learning rate ($lr = 0.02$) also provides strong robustness similar to ANP. However, it has an obvious drop in ACC since a large learning rate may destroy the features learned before and the new ones learned on limited data are poor in generalization. The poor accuracy hinders its usage in practical scenarios. In contrast, ANP has similar robustness against backdoor attacks while maintains ACC at a relatively high level at the same time. Similar to our method, Fine-pruning (FP) [24] also prunes some neurons and fine-tunes the pruned model on clean data. Specifically, It only prunes neurons in the last convolutional layer using a rule-of-thumb[6], and provides limited robustness against backdoor attacks. Differently, the proposed ANP is data-driven (*i.e.*, optimizing pruning masks based on data) and prunes neurons globally (*i.e.*, pruning neurons in all layers), leading to higher ACC and lower ASR than FP. MCR suggests mitigating backdoor by using a curve model lying on a path connection between two backdoored models in the loss landscapes. However, with limited data, it fails in providing high robustness against complex triggers (*e.g.* a random trigger sampled from Gaussian in blend backdoor attack or the dynamic trigger in input-aware attack). By contrast, ANP always reduces ASR significantly no matter what the trigger looks like. Note that ANP only prunes neurons and never changes any parameters of backdoored DNNs, while other methods all fine-tune parameters. Under this situation, ANP still provides the almost best robustness against these state-of-the-art attacks, which indicates that pure pruning (without fine-tuning) is still a promising defense against backdoor attacks.

We also compare the training time of ANP and other baselines in Table 2, which indicates the efficiency of the proposed ANP. In particular, we apply 2000 iterations for FT, FP, and ANP, and 200 epochs (*i.e.*, 800 iterations) for MCR following the open-sourced code[7]. Note that ANP is not as slow as the vanilla adversarial training (AT) since ANP only requires one extra backpropagation while PGD inside AT usually needs 10 extra backpropagations (PGD-10) in each update of model parameters. Besides, although ANP takes $2.5\times$ time compared to FT, it improves accuracy (*e.g.*, $88.98\% \rightarrow 93.44\%$ for Blend) and reduce vulnerability (*e.g.*,$10.20\% \rightarrow 0.46\%$ for Blend) significantly, making the overhead caused by ANP acceptable.

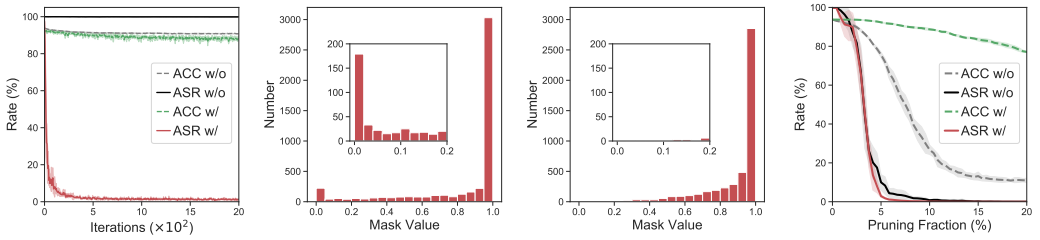
4.3 Ablation Studies

Effect of Adversarial Neuron Perturbation. Recalling Section 3.2, we apply adversarial neuron perturbations to induce the injected backdoor and then defend against backdoor attacks. Here, we evaluate the effect of the adversarial neuron perturbation in our defense strategy. In the following experiments, we defend against Badnets attack by default, unless otherwise specified. First, we optimize masks with perturbations ($\alpha = 0.2$ and $\epsilon = 0.4$, *i.e.*, ANP) and without perturbations ($\alpha = 1.0$, *i.e.*, the vanilla pruning method). Figure 2(a) shows the ACC and ASR on the test set during optimization. We find the optimization under adversarial neuron perturbations helps the backdoored model suppress ASR significantly while the vanilla one always has high ASR throughout the training process. Next, we illustrate the mask distribution after optimization. Figure 2(b) illustrates a certain fraction of masks becomes zero, which means that adversarial neuron perturbations indeed find some sensitive neurons and helps the model to remove them. In contrast, there is no mask close to 0 for the vanilla pruning method in Figure 2(c). Finally, we prune different fractions of neurons according to their mask in Figure 2(d). For both methods, ASR becomes low after pruning more than 5% of neurons[8]. However, ACC also starts to degrade significantly for the vanilla pruning method with more

[6]Fine-pruning prunes neurons that are less active on clean samples until there is an obvious drop in accuracy on a clean validation set.

[7]`https://github.com/IBM/model-sanitization`

[8]This is conceivable. As long as the backdoor-related neurons are assigned slightly smaller masks than other neurons, we still can prune them first and remove the backdoor.



(a) Optimization (b) With perturbations (c) Without perturbations (d) Pruning

Figure 2: Comparison between optimization with and without adversarial neuron perturbations.

Table 3: Results with small budget $\epsilon = 0.1$ against Blend attack on 500 clean images. "Neurons↓" indicates the number of neurons pruned by ANP.

Steps	1	2	5	10
Time (s)	239.9	359.2	551.8	943.9
Neurons↓	159	188	235	259
ASR (%)	65.19	13.40	1.06	0.90
ACC (%)	93.62	93.07	92.95	92.72

Table 4: Results with large budget $\epsilon = 0.4$ against Blend attack on 500 clean images. "Neurons↓" indicates the number of neurons pruned by ANP.

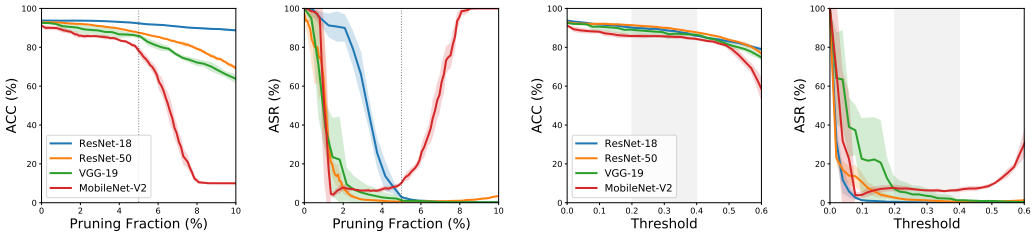
Steps	1	2	5	10
Time (s)	241.5	357.2	557.1	950.1
Neurons↓	233	239	281	296
ASR (%)	0.46	1.30	5.30	31.34
ACC (%)	93.44	94.07	93.57	94.28

than 5% of neurons pruned. Since backdoored-related neurons are still mixed with other neurons (*e.g.*, discriminant neurons), the vanilla pruning method prunes some discriminant neurons incorrectly. Meanwhile, ANP always keeps ACC at a relatively high level. In conclusion, the adversarial neuron perturbation is the key to distinguish the backdoor-related neurons from the other neurons, which thereby successfully obtains high ACC as well as low ASR.

In addition, we explore the effects of number of iterations in crafting adversarial neuron perturbations. We conduct experiments with varying numbers of steps (1/2/5/10) in ANP with a small perturbation budget ($\epsilon = 0.1$) and a large perturbation budget ($\epsilon = 0.4$) respectively. The other settings are the same as Section 4.1. The experimental results are shown in Tables 3-4. Under a small perturbation budget, with more steps for ANP, the ASR decreases with a slight drop in ACC. This is because ANP with a single step and small size is too weak to distinguish benign neurons and backdoor-related neurons, while more steps can help ANP find more backdoor-related neurons. However, under a large perturbation budget, ANP with more steps has worse robustness. This is because, with a large perturbation budget, more neurons become sensitive. As a result, ANP with more steps finds too many "suspicious" neurons, and it is unable to identify backdoor-related neurons from them. In conclusion, we can strengthen the power of ANP to find backdoor-related neurons using a larger perturbation budget or more steps. Among them, single-step ANP with a slightly larger budget is more practical due to its time efficiency, and we adopt this by default.

Results with Varying Hyperparameters. As mentioned in Section 3.3, the hyper-parameter α in ANP controls the trade-off between the accuracy on clean data and the robustness against backdoor attacks. To test the performance with different α, we optimize masks for a Badnets ResNet-18 based on 1% of clean data using different $\alpha \in [0, 1]$ with a fixed budget $\epsilon = 0.4$. In pruning, we always prune neurons by the threshold 0.2. As shown in the left plot of Figure 4, ANP is able to achieve a high robustness (ASR $< 4\%$) when $\alpha \leq 0.6$. Meanwhile, ANP maintains a high natural accuracy (ACC $\geq 90\%$) as long as $\alpha \geq 0.1$. As a result, ANP behaves well with high ACC and low ASR in a range of $\alpha \in [0.1, 0.6]$.

Similarly, we also test the performance with different perturbation budgets ϵ. The experiment settings are almost the same except a fixed trade-off coefficient $\alpha = 0.2$ and varying budgets $\epsilon \in [0, 1]$. For example, we can provide obvious robustness (ASR becomes 7.45%) with a small perturbation budget ($\epsilon = 0.2$) as shown in the right plot of Figure 4. However, we find the accuracy on clean data degrades with a large perturbation budget. This is because too large perturbation budget brings much difficulty to converge well and ANP only finds a poor solution, which fails in identifying these discriminant and robust neurons and prunes part of them. As a result, ACC decreases significantly. In conclusion, the



(a) ACC by fraction (b) ASR by fraction (c) ACC by threshold (d) ASR by threshold

Figure 3: Performance ($\pm$std over 5 random runs) of pruned models using different architectures by different pruning fractions or different thresholds.

proposed ANP is stable in a large range of the trade-off coefficient $\alpha \in [0.1, 0.6]$ and the perturbation budget $\epsilon \in [0.2, 0.7]$, demonstrating that ANP is not sensitive to hyper-parameters.

From the discussion above, we find that the hyperparameters are insensitive as shown in Figure 4 and ANP performs well across a wide range of hyperparameters, *e.g.*, trade-off coefficient $\alpha \in [0.1, 0.6]$ against Badnets attack. In addition, with varying hyperparameters, the performance trends are very consistent across different backdoor attacks as shown in Figure X in Appendix C.1. As a result, even though the attack (*e.g.*, Blend) is unknown, the defender could tune α against a known attack (*e.g.*, Badnets) and find the 0.2 is a good choice. ANP with $\alpha = 0.2$ also achieves satisfactory performance against Blend attack. In conclusion, the selection of hyperparameters in ANP is not difficult.

Pruning by Fraction or by Threshold. Different from previous pruning methods that use a fixed pruning fraction, the proposed ANP prunes neurons by a threshold when generalized across different model architectures. To verify this, we train different Badnets models using ResNet-18 [15], ResNet-50 [15], and VGG-19 [37] with the same settings as Section 4.1. We also train a model with Badnets MobileNet-V2 with 300 epochs . We optimize the pruning masks for these models with $\epsilon = 0.4, \alpha = 0.2$ for ResNet-18, ResNet-50, and VGG-19. For MobileNet-V2, we found the perturbation budget $\epsilon = 0.4$ is too large, leading to a failure in convergence. So we apply a smaller budget $\epsilon = 0.2$ and the same trade-off coefficient $\alpha = 0.2$. We show the ACC and ASR after pruning by varying pruning fractions in Figure 3(a) and 3(b), which indicates that the suitable fraction varies across model architectures. For example, ResNet-18 only has low ASR after pruning 5% of neurons, while MobileNet-V2 has a large accuracy drop on clean samples with more than 5% neurons pruned. Figure 3(c) and Figure 3(d) show ACC and ASR after pruning by different thresholds respectively. When pruning by the threshold, we find there is a large overlap in $[0.2, 0.5]$ (the gray zone) in which all models have high ACC and low ASR simultaneously. That is why we adopt the strategy of pruning by the threshold in ANP. Note that, even for low-capacity models as MobileNet-V2, it is still possible to remove the injected backdoor by neuron pruning (without the fine-tuning).

Pruning on Different Components. We also compare the performance when applying ANP to different components inside DNNs. We prune a backdoored 4-layer CNN (2 conv layers + 1 fully-connected layer + output layer) on MNIST, and find it more efficient to prune neurons in some layers. In particular, we achieve 0.43% of ASR (99.04% of ACC) when two neurons in the 2nd layer are pruned, while we only achieve 4.42% of ASR (92.48% of ACC) even after pruning 150 neurons in the fully-connected layer. The experimental settings and more results can be found in Appendix C.2. This indicates that the structure of components matter in pruning-based defense. We leave this in our future work.

4.4 Performance under Limited Computation Resource

In practical scenarios, the defender usually has limited clean data and computation resources, bringing more difficulty to repair backdoored models. While Section 4.2 has discussed the effectiveness of the proposed ANP on the extremely small amount of data, this part focuses on the performance with limited computation resources in defense.

In particular, we optimize all masks for a Badnets ResNet-18 with varying number of iterations (20, 100, 400, and 2000) on 1% of clean data (500 images) and prune the model from neurons with the small mask to neurons with the large mask. Figure 5 shows the ACC and ASR with varying pruning

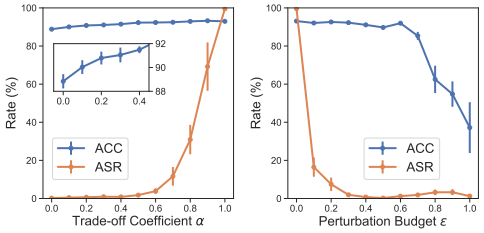


Figure 4: Performance ($\pm$ std over 5 random runs) of the pruned model by a threshold 0.2 with different hyper-parameters (*Left*: trade-off coefficient α; *Right*: perturbation budget ϵ).

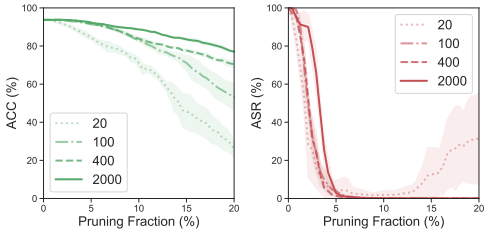


Figure 5: Performance ($\pm$std over 5 random runs) of the proposed ANP with varying pruning fractions based on different numbers of iterations.

fractions based on different numbers of iterations. We find that some backdoor-related neurons have already been distinguished just after 20 iterations. For example, after removing 4% of neurons with the smallest masks, the pruned model has 6.28% of ASR and 86.65% of ACC. As the pruning fraction increases, ASR falls significantly first and raises again after 10% of neurons are pruned. This is because ANP has not yet separated them completely due to extremely limited computation (only 20 iterations). With more than 100 iterations, this phenomenon disappears. And ANP based on 100 iterations already has comparable performance to ANP on 2000 iterations, especially when we have not pruned too many neurons ($< 8\%$).

We also find that ANP with 2000 iterations should prune more ($+1\%$) neurons to degrade ASR than ones with fewer iterations. We conjecture this is because there also exist some sensitive neurons that are unrelated to the injected backdoor[9], which are assigned with small masks at the late optimization stage. Besides, for ACC, we can see that more iterations maintain a higher ACC than fewer iterations, especially when more neurons are pruned, as the left neurons are almost the discriminant and robust ones. In conclusion, even with extremely small number of iterations (*e.g.*, 20 iterations), ANP is able to distinguish the backdoor-related neurons from other neurons and obtains satisfactory robustness against backdoor attacks.

5 Conclusion

In this paper, we identified a sensitivity of backdoored DNNs to adversarial neuron perturbations, which induces them to present backdoor behaviours even without the presence of the trigger patterns. Based on these findings, we proposed *Adversarial Neuron Pruning* (ANP) to prune the sensitive neurons under adversarial neuron perturbations. Comprehensive experiments show that ANP consistently removes the injected backdoor and provides the highest robustness against several state-of-the-art backdoor attacks even with a limited amount of clean data and computation resources. Finally, our work also reminds researchers that pruning (without fine-tuning) is still a promising defense against backdoor attacks.

Broader Impact

The backdoor attack has become a threat to outsourcing training and open-sourcing models. We propose ANP to improve the robustness against these backdoor attacks, which may help to build a more secure model even trained in an untrustworthy third-party platform. Further, we do not want this paper to bring overoptimism about AI safety to the society. Since the backdoor attack is only a part of potential risks (*e.g.*, adversarial attacks, privacy leakage, and model extraction), there is still a long way towards secure AI and trustworthy AI.

[9]Recalling Figure 1(b), we can see that the benign model also suffers from the adversarial neuron perturbations, although its sensitivity is much weaker than that of backdoored models.

Acknowledgments

Yisen Wang is partially supported by the National Natural Science Foundation of China under Grant 62006153, and Project 2020BD006 supported by PKU-Baidu Fund.

References

[1] Mauro Barni, Kassem Kallas, and Benedetta Tondi. A new backdoor attack in cnns by training set corruption without label poisoning. In *ICIP*, 2019.

[2] Battista Biggio, Blaine Nelson, and Pavel Laskov. Support vector machines under adversarial label noise. In *ACML*, 2011.

[3] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. In *ECML/PKDD*, 2013.

[4] Eitan Borgnia, Valeriia Cherepanova, Liam Fowl, Amin Ghiasi, Jonas Geiping, Micah Goldblum, Tom Goldstein, and Arjun Gupta. Strong data augmentation sanitizes poisoning and backdoor attacks without an accuracy tradeoff. In *ICASSP*, 2021.

[5] Huili Chen, Cheng Fu, Jishen Zhao, and Farinaz Koushanfar. Deepinspect: A black-box trojan detection and mitigation framework for deep neural networks. In *IJCAI*, 2019.

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

[7] Xinyun Chen, Wenxiao Wang, Chris Bender, Yiming Ding, Ruoxi Jia, Bo Li, and Dawn Song. Refit: a unified watermark removal framework for deep learning systems with limited data. *arXiv preprint arXiv:1911.07205*, 2019.

[8] Ilias Diakonikolas, Gautam Kamath, Daniel Kane, Jerry Li, Jacob Steinhardt, and Alistair Stewart. Sever: A robust meta-algorithm for stochastic optimization. In *ICML*, 2019.

[9] Chao Gao, Yuan Yao, and Weizhi Zhu. Generative adversarial nets for robust scatter estimation: A proper scoring rule perspective. *Journal of Machine Learning Research*, 21(160):1–48, 2020.

[10] Yansong Gao, Change Xu, Derui Wang, Shiping Chen, Damith C Ranasinghe, and Surya Nepal. Strip: A defence against trojan attacks on deep neural networks. In *ACSAC*, 2019.

[11] Amirata Ghorbani and James Zou. Neuron shapley: Discovering the responsible neurons. In *NeurIPS*, 2021.

[12] Micah Goldblum, Dimitris Tsipras, Chulin Xie, Xinyun Chen, Avi Schwarzschild, Dawn Song, Aleksander Madry, Bo Li, and Tom Goldstein. Data security for machine learning: Data poisoning, backdoor attacks, and defenses. *arXiv preprint arXiv:2012.10544*, 2020.

[13] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *ICLR*, 2015.

[14] Tianyu Gu, Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Evaluating backdooring attacks on deep neural networks. *IEEE Access*, 7:47230–47244, 2019.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[16] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

[17] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *ICML*, 2017.

[18] Pang Wei Koh, Jacob Steinhardt, and Percy Liang. Stronger data poisoning attacks break data sanitization defenses. *arXiv preprint arXiv:1811.00741*, 2018.

[19] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. *Technical Report, University of Toronto*, 2009.

[20] Alexander Levine and Soheil Feizi. Deep partition aggregation: Provable defense against general poisoning attacks. *arXiv preprint arXiv:2006.14768*, 2020.

[21] Alexander Levine and Soheil Feizi. Deep partition aggregation: Provable defense against general poisoning attacks. In *ICLR*, 2021.

[22] Yige Li, Nodens Koren, Lingjuan Lyu, Xixiang Lyu, Bo Li, and Xingjun Ma. Neural attention distillation: Erasing backdoor triggers from deep neural networks. In *ICLR*, 2021.

[23] Yige Li, Xixiang Lyu, Nodens Koren, Lingjuan Lyu, Bo Li, and Xingjun Ma. Anti-backdoor learning: Training clean models on poisoned data. In *NeurIPS*, 2021.

[24] Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Fine-pruning: Defending against backdooring attacks on deep neural networks. In *RAID*, 2018.

[25] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. Trojaning attack on neural networks. In *NDSS*, 2018.

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

[27] Yuntao Liu, Yang Xie, and Ankur Srivastava. Neural trojans. In *ICCD*, 2017.

[28] Shiqing Ma and Yingqi Liu. Nic: Detecting adversarial samples with neural network invariant checking. In *NDSS*, 2019.

[29] Yuzhe Ma, Xiaojin Zhu Zhu, and Justin Hsu. Data poisoning against differentially-private learners: Attacks and defenses. In *IJCAI*, 2019.

[30] Anh Nguyen and Anh Tran. Input-aware dynamic backdoor attack. In *NeurIPS*, 2020.

[31] Anh Nguyen and Anh Tran. Wanet–imperceptible warping-based backdoor attack. In *ICLR*, 2021.

[32] Ren Pang, Hua Shen, Xinyang Zhang, Shouling Ji, Yevgeniy Vorobeychik, Xiapu Luo, Alex Liu, and Ting Wang. A tale of evil twins: Adversarial inputs versus poisoned models. In *CCS*, 2020.

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

[34] Elan Rosenfeld, Ezra Winston, Pradeep Ravikumar, and Zico Kolter. Certified robustness to label-flipping attacks via randomized smoothing. In *ICML*, 2020.

[35] Ali Shafahi, W Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. Poison frogs! targeted clean-label poisoning attacks on neural networks. In *NeurIPS*, 2018.

[36] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[37] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[38] Jacob Steinhardt, Pang Wei Koh, and Percy Liang. Certified defenses for data poisoning attacks. In *NeurIPS*, 2017.

[39] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *ICLR*, 2014.

[40] Lue Tao, Lei Feng, Jinfeng Yi, Sheng-Jun Huang, and Songcan Chen. Better safe than sorry: Preventing delusive adversaries with adversarial training. In *NeurIPS*, 2021.

[41] Brandon Tran, Jerry Li, and Aleksander Madry. Spectral signatures in backdoor attacks. In *NeurIPS*, 2018.

[42] Alexander Turner, Dimitris Tsipras, and Aleksander Madry. Label-consistent backdoor attacks. *arXiv preprint arXiv:1912.02771*, 2019.

[43] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y Zhao. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *S&P*, 2019.

[44] Ren Wang, Gaoyuan Zhang, Sijia Liu, Pin-Yu Chen, Jinjun Xiong, and Meng Wang. Practical detection of trojan neural networks: Data-limited and data-free cases. In *ECCV*, 2020.

[45] Yisen Wang, Xuejiao Deng, Songbai Pu, and Zhiheng Huang. Residual convolutional ctc networks for automatic speech recognition. *arXiv preprint arXiv:1702.07793*, 2017.

[46] Maurice Weber, Xiaojun Xu, Bojan Karlas, Ce Zhang, and Bo Li. Rab: Provable robustness against backdoor attacks. *arXiv preprint arXiv:2003.08904*, 2020.

[47] Dongxian Wu, Shu-Tao Xia, and Yisen Wang. Adversarial weight perturbation helps robust generalization. In *NeurIPS*, 2020.

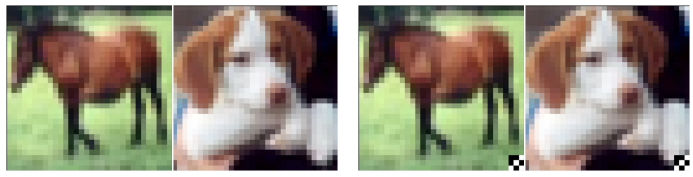
[48] Pu Zhao, Pin-Yu Chen, Payel Das, Karthikeyan Natesan Ramamurthy, and Xue Lin. Bridging mode connectivity in loss landscapes and adversarial robustness. In *ICLR*, 2019.

[49] Haoti Zhong, Cong Liao, Anna Cinzia Squicciarini, Sencun Zhu, and David Miller. Backdoor embedding in convolutional neural network models via invisible perturbation. In *CODASPY*, 2020.

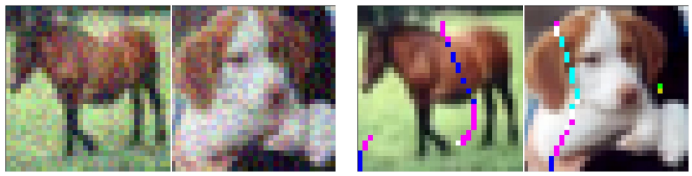
[50] Liuwan Zhu, Rui Ning, Cong Wang, Chunsheng Xin, and Hongyi Wu. Gangsweep: Sweep out neural backdoors by gan. In *MM*, 2020.

A More Implementation Details on Backdoor Attacks

In this part, we provide more implementation details on several state-of-the-art backdoor attacks. The backdoor triggers applied in our experiments are shown in Figure 6 and Figure 7.

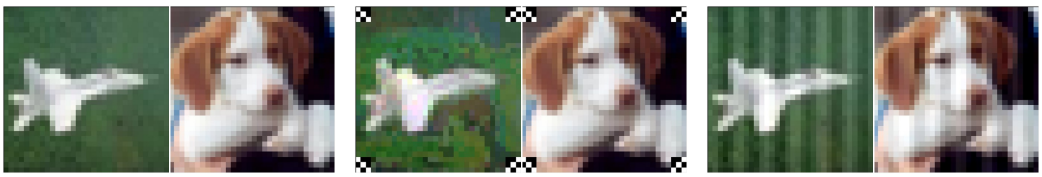


(a) Clean Data (b) Poisoned Data in Badnets



(c) Poisoned Data in Blend (d) Poisoned Data in IAB

Figure 6: Examples of poisoned CIFAR-10 images at the training time (*Left*) and the test time (*Right*). Given the target label (class 0, "plane"), Badnets, Blend, and IAB always attach a predefined trigger to some samples from other classes and relabel them as the target label at the training time.



(a) Clean Data (b) Poisoned Data in CLB (c) Poisoned Data in SIG

Figure 7: Examples of poisoned CIFAR-10 images at the training time (*Left*) and the test time (*Right*). Given the target label (class 0, "plane"), CLB and SIG instead attach a predefined trigger to some training samples belonging to target label to avoid incorrect labels at the training time.

We always set the target label as class 0 ("plane"). The detailed implementation on several state-of-the-art backdoor attacks are as follows,

- **Badnets** [14]: The trigger is a 3×3 checkerboard at the bottom right corner of images as shown in Figure 6(b). Given the target label, we attach the trigger to 5% of training samples from other classes and relabel them as the target label. After training for 200 epochs, we achieve the attack success rate (ASR) of 99.97% and the natural accuracy on clean data (ACC) of 93.73%.
- **Blend attack** [6]: We first generate a trigger pattern where each pixel value is sampled from a uniform distribution in $[0, 255]$ as shown in Figure 6(c). Given the target class, we randomly select 5% of training samples from other classes for poisoning. We attach the trigger $\mathbf{t}$ to the sample $\mathbf{x}$ using a blended injection strategy, *i.e.*, $\alpha\mathbf{t} + (1-\alpha)\mathbf{x}$. Here, we set the blend ratio $\alpha = 0.2$. Next, we relabel them as the target label. After training for 200 epochs, we achieve ASR of 100.00% and ACC of 94.82%.
- **Input-aware Attack (IAB)** [30]: The dynamic trigger varies across samples as shown in Figure 6(d). Based on the open-source code[10], we train the classifier and the trigger

[10] https://github.com/VinAIResearch/input-aware-backdoor-attack-release

generator concurrently. We apply two types of target label selection. For all-to-one, we always set the class 0 as the target label, attach the dynamic trigger to the samples from other classes and relabel them as the target label. For all-to-all, after attaching the trigger, we relabel samples from class i as class $(i+1)$[11]. Finally, we achieve ASR of $98.49\%/92.88\%$ and ACC of $93.89\%/94.10\%$ for IAB with the all-to-one target label or the all-to-all target label respectively.

- **Sinusoidal signal attack (SIG)** [1]: We superimpose a sinusoidal signal over the inputs as the trigger following Barni et al. [1]. Given the target label, we attach the trigger to 80% of samples from the target class. After training for 200 epochs, we achieve ASR of 94.26% and ACC of 93.64%.
- **Clean-label Attack (CLB)** [42]: The trigger is a 3×3 checkerboard at the four corners of images as shown in Figure 7(b). Given the target label, we attach the trigger to 80% of samples from the target class. To make the backdoored DNN rely more on the trigger pattern rather than the salient features from the class, we apply adversarial perturbations to render these poisoned samples harder to classify during training following Turner et al. [42]. Specifically, we use Projected Gradient Descent (PGD) to generate adversarial perturbations with the ℓ_∞-norm maximum perturbation size $\epsilon = 16$ based on the open-source code[12]. After training for 200 epochs, we achieve ASR of 99.94% and ACC of 93.78%.

B More Results on Adversarial Neuron Perturbations

In Section 3.2, we have demonstrated that the backdoored models are much easier to collapse and tend to predict the target label on clean samples when their neurons are adversarially perturbed. Here, we further explain that such sensitivity is from the backdoor itself rather than the unbalanced dataset caused by some attacks. For example, on CIFAR-10 training set (5000 samples for every class), Badnets relabel 5% of samples from other classes. As a result, the target class has 7250 samples, while each of the other classes only has 4750 samples.

First, we create an unbalanced dataset by replacing the incorrectly-labeled samples in Badnets training set with extra samples from target class from open-source labeled data[13]. We train another benign model on the unbalanced dataset using the same training settings as the backdoored model by Badnets. Besides, we also illustrate the results of a backdoored model by SIG, which is also based on a balanced dataset since SIG only attaches the trigger to the samples from the target class.

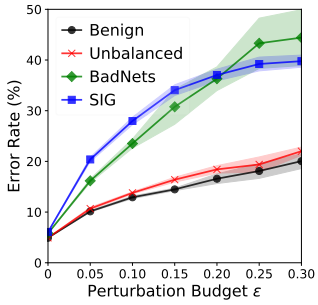


Figure 8: The error rate ($\pm$ std over 5 random runs) of different models under neuron perturbations.

Following Section 3.2, we generate adversarial neuron perturbations for these models. Figure 8 shows the benign model based on the unbalanced dataset almost has a similar error rate compared to the original benign one, and the backdoored models by Badnets and SIG always have larger error rates with the same perturbation budget compared to the benign models. We also illustrate the proportion of different classes in predictions in Figure 9. For the backdoored model by SIG, the majority of misclassified samples have already been predicted as the target label. For the backdoored model by Badnets, much more samples are classified as the target class compared to the benign one on an unbalanced dataset. In conclusion, the injected backdoor itself results in sensitivity under adversarial neuron perturbation rather than an unbalanced dataset.

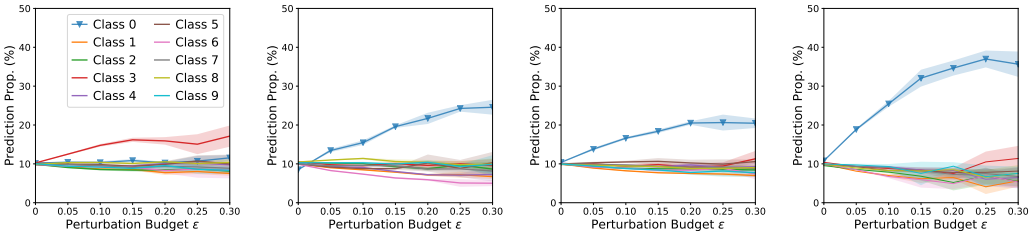
C More Results on Adversarial Neuron Pruning (ANP)

C.1 Performance Trends across Different Backdoor Attacks

[11] For class 9, the target label is class 0.

[12] https://github.com/MadryLab/label-consistent-backdoor-code

[13] https://github.com/yaircarmon/semisup-adv



(a) Benign (b) SIG (c) Benign (Unbalanced) (d) Badnets

Figure 9: The proportion ($\pm$ std over 5 random runs) of different classes in predictions by a benign model, another benign one trained on an unbalanced training set, and two backdoored models. The first benign model and the SIG model are based on a balanced training set, while the second benign model and the Badnets model are based on an unbalanced training set.

In practical scenarios, the adversary may exploit any possible backdoor attacks. While the selection of hyperparameters seriously affects the effectiveness of defense, it is important to seek a selection strategy of hyperparameters for the defender. Here, we conduct experiments to illustrate the performance trends across different backdoor attacks, including Badnets, Blend, and CLB. The experimental settings are the same as Section 4.1. We evaluate the performance of ANP against three backdoor attacks with varying trade-off coefficient α and the results are shown in Figure 10. As α increases, we find that ASR remains low until a common threshold ($\alpha = 0.5$) is exceeded. Besides, ACC always keeps high with different α. Thus, with varying hyperparameters, the performance trends are very consistent across different backdoor attacks. Besides, we also find the hyperparameters are insensitive and ANP performs well across a wide range of hyperparameters in Figure 4 of Section 4.3.

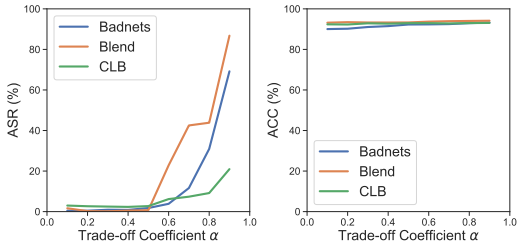


Figure 10: Performance of the pruned model by a threshold 0.2 with varying trade-off coefficient α.

Therefore, for simplicity, we always set the same hyperparameters (*e.g.*, $\alpha = 0.2$ as shown in Section 4.1) in this paper against various backdoor attacks unless otherwise specified. In practical scenarios, the defender could tune the hyperparameters in a straightforward way: assuming the unknown attack is Blend attack, the defender could tune against a known attack (*e.g.*, Badnets) and find the 0.2 is a good choice. ANP with also achieves satisfactory performance against Blend attack.

C.2 Pruning on Different Components

In this part, we explore the performance when applying ANP to different components inside DNNs. We conduct a toy experiment based on a 5-layer fully-connected (FC) neural network to show the effectiveness of the proposed ANP. We train the Badnets model on a poisoned MNIST training set whose backdoor trigger is a square at the bottom right and the injection rate is 10%. After training, the ACC is 98.08%, and the ASR is 100.00%. We apply ANP ($\alpha = 0.5, \epsilon = 0.3$) to the whole model, we obtain a repaired model with 93.86% of ACC and 1.93% of ASR after pruning 560 neurons (2000 in total).

Table 5: Results on MNIST based on a 5-layer FC neural network. "Neurons↓" indicates the number of neurons pruned by ANP, and "Total" indicates the number of all neurons.

	Before	ANP
Neurons↓	0	560
Total	2000	2000
ASR (%)	100.00	1.93
ACC (%)	98.08	93.86

Table 6: Results on MNIST of different components based on a 4-layer CNN. "Neurons↓" indicates the number of neurons pruned by ANP, and "Total" indicates the number of all neurons.

	Before	ANP on all	ANP on Conv	ANP on FC
Neurons↓	0	8	2	150
Total	560	560	48	512
ASR (%)	99.99	0.32	0.43	4.42
Acc (%)	99.34	99.22	99.04	92.48

Next, we conduct a similar experiment on the same poisoned MNIST training set based on a 4-layer CNN (2 conv layers + 1 FC layer + output layer) to compare the performance between the conv layers and the FC layer. After training, the backdoored model has 99.34% of ACC and 99.99% of ASR. We apply ANP on conv layers and the FC layer respectively. In particular, If we only apply ANP on conv layers, we achieve 99.04% of ACC and 0.43% of ASR just after pruning 2 neurons in the second conv layer. If only applying ANP on the FC layer, we achieve 92.48% of ACC and 4.42% of ASR even after pruning 150 neurons (512 in total). We conjecture this is because, for a simple trigger pattern, the neurons most related to the backdoor tend to locate in the shallow layers (*i.e.*, conv layers), which makes pruning on the conv layers has better performance. Therefore, the performance of a component is dependent on its structure.

C.3 Benchmarking Results Based on Varying Fraction of Clean Data

In this part, we provide more experimental results based on varying fraction of CIFAR-10 clean data, including 10% (5000 images), 1% (500 images), and 0.1% (50 images), in Tables 7-9.

We find all baselines suffer from difficulty in defense with a limited amount of data, *i.e.*, ASR significantly increases when the number of clean samples decreases. However, the proposed ANP always degrades ASR lower than 5% even with 0.1% of clean data (50 images), which indicates that ANP provides satisfactory robustness against backdoor attacks. Besides, when we have enough amount of data (*e.g.*, 10% of clean data), the ACC drop in ANP is negligible ($< 1\%$).

Note that Fine-tuning (FT) and Fine-pruning (FP) have better natural accuracy (ACC) based on 1% of clean data than 10% of clean data. We conjecture this is because it is much easy to overfit to a small number of samples (*e.g.*, 500 images) and keep the original natural ACC better. However, we always achieve higher ASR based on less amount of data.

Table 7: Performance ($\pm$ std over 5 random runs) of 4 defense methods against 6 backdoor attacks on 10% (5000 images) of clean data on CIFAR-10 training set using ResNet-18.

Metric	Defense	Badnets	Blend	IAB-one	IAB-all	CLB	SIG
ACC	Before	93.73	94.82	93.89	94.10	93.78	93.64
	FT (0.01)	87.74±1.14	88.39±0.96	85.59±1.11	86.54±0.57	87.12±1.53	86.95±1.12
	FT (0.02)	85.56±0.89	86.25±0.94	83.75±1.15	85.62±1.59	84.70±0.88	85.80±1.36
	FP	86.95±1.03	88.26±0.57	85.64±0.97	85.26±2.53	86.89±1.21	87.47±1.41
	MCR(0.3)	89.17±0.33	89.45±0.53	87.01±0.10	88.53±1.12	89.78±0.21	85.88±0.56
	ANP	**93.01±0.22**	**93.86±2.13**	**93.40±0.11**	**93.56±0.09**	**93.37±0.20**	**93.65±0.17**
ASR	Before	99.97	100.0	98.49	92.88	99.94	94.26
	FT (0.01)	2.60±0.98	0.86±1.29	4.74±1.74	1.91±0.26	1.56±0.19	0.86±1.29
	FT (0.02)	2.13±1.23	**0.27±0.22**	2.67±0.54	2.15±0.21	2.27±0.75	0.92±0.29
	FP	**1.83±0.42**	0.38±0.43	1.97±0.42	6.60±1.99	**1.97±0.42**	0.62±0.23
	MCR(0.3)	5.70±1.21	10.57±2.21	15.23±1.43	17.17±2.71	3.77±1.12	0.52±0.21
	ANP	3.34±1.43	2.13±1.47	**1.32±0.29**	**0.79±0.04**	3.65±1.48	**0.28±0.14**

Table 8: Performance ($\pm$ std over 5 random runs) of 4 defense methods against 6 backdoor attacks on 1% (500 images) of clean data on CIFAR-10 training set using ResNet-18.

Metric	Defense	Badnets	Blend	IAB-one	IAB-all	CLB	SIG
ACC	Before	93.73	94.82	93.89	94.10	93.78	93.64
	FT (0.01)	90.48±0.36	92.12±0.33	88.68±0.26	89.06±0.22	91.26±0.24	91.19±0.12
	FT (0.02)	87.23±0.51	88.98±0.32	84.85±0.57	83.77±0.32	88.25±0.20	88.63±0.33
	FP	**92.18±0.09**	92.40±0.04	91.57±0.12	92.28±0.05	91.91±0.12	91.64±0.12
	MCR(0.3)	85.95±0.21	88.26±0.35	86.30±0.10	84.53±1.12	86.87±0.33	85.88±0.56
	ANP	90.20±0.41	**93.44±0.43**	**92.62±0.32**	**92.79±0.16**	**92.67±0.16**	**93.40±0.12**
ASR	Before	99.97	100.0	98.49	92.88	99.94	94.26
	FT (0.01)	11.70±4.10	47.17±19.51	0.99±0.55	1.36±0.31	12.51±8.16	0.40±0.23
	FT (0.02)	2.95±0.72	10.20±20.64	1.70±0.19	1.83±0.38	**1.17±0.10**	0.39±0.04
	FP	5.34±1.94	65.39±14.55	20.73±10.35	32.36±1.92	3.40±0.69	0.32±0.21
	MCR(0.3)	5.70±2.31	13.57±1.29	30.23±3.47	35.17±3.76	12.77±2.12	0.52±0.10
	ANP	**0.45±0.17**	**0.46±0.23**	**0.88±0.14**	**0.86±0.05**	3.98±2.83	**0.28±0.25**

Table 9: Performance ($\pm$ std over 5 random runs) of 4 defense methods against 6 backdoor attacks on 0.1% (50 images) of clean data on CIFAR-10 training set using ResNet-18.

Metric	Defense	Badnets	Blend	IAB-one	IAB-all	CLB	SIG
ACC	Before	93.73	94.82	93.89	94.10	93.78	93.64
	FT (0.01)	88.90±1.90	91.21±0.74	88.17±2.15	84.21±2.89	**91.10±0.40**	**90.00±0.79**
	FT (0.02)	82.06±2.84	86.79±6.81	75.28±4.92	60.45±9.36	88.11±1.76	84.37±2.94
	FP	**90.69±0.23**	**91.29±0.17**	**90.74±0.23**	**90.26±0.24**	89.27±0.66	89.96±0.90
	MCR(0.3)	80.21±5.19	81.35±3.19	79.10±3.15	82.53±1.12	77.33±4.33	85.88±0.56
	ANP	86.62±0.64	88.12±0.38	88.54±1.25	85.66±1.27	86.73±0.99	89.54±0.28
ASR	Before	99.97	100.0	98.49	92.88	99.94	94.26
	FT (0.01)	82.58±24.42	100.00±0.00	1.24±0.53	14.69±22.87	50.25±40.73	12.59±11.57
	FT (0.02)	16.27±12.55	99.99±0.03	2.29±0.85	3.78±1.01	22.71±43.19	0.07±0.07
	FP	30.71±16.72	99.95±0.02	30.82±51.57	42.38±2.25	1.35±1.00	**0.13±0.03**
	MCR(0.3)	33.70±3.39	43.57±1.29	52.23±3.67	53.17±5.21	12.77±3.77	1.10±0.01
	ANP	**0.45±0.17**	**5.70±8.87**	**1.84±0.76**	**1.43±0.28**	**0.23±0.10**	2.85±1.19