Invisible Backdoor Attacks Against Deep Neural Networks

Shaofeng Li[1], **Benjamin Zi Hao Zhao**[2], **Jiahao Yu**[1], **Minhui Xue**[3], **Dali Kaafar**[4], **Haojin Zhu**[1]
[1]Shanghai Jiao Tong University, China
[2]UNSW and CSIRO-Data61, Australia
[3]The University of Adelaide, Australia
[4]Macquarie University and CSIRO-Data61, Australia

Abstract

Deep neural networks (DNNs) have been proven vulnerable to backdoor attacks, where hidden features (patterns) trained to a normal model, and only activated by some specific input (called triggers), trick the model into producing unexpected behavior. In this paper, we design an optimization framework to create covert and scattered triggers for backdoor attacks, *invisible backdoors*, where triggers can amplify the specific neuron activation, while being invisible to both backdoor detection methods and human inspection. We use the Perceptual Adversarial Similarity Score (PASS) (Rozsa, Rudd, and Boult 2016) to define invisibility for human users and apply L_2 and L_0 regularization into the optimization process to hide the trigger within the input data. We show that the proposed invisible backdoors can be fairly effective across various DNN models as well as three datasets CIFAR-10, CIFAR-100, and GTSRB, by measuring their attack success rates and invisibility scores.

1 Introduction

Deep neural networks have been proven to outperform traditional machine learning techniques and outperform humans' cognitive capacity in many domains, such as image processing (Ciregan, Meier, and Schmidhuber 2012), speech recognition (Hinton et al. 2012) and board games (Mnih et al. 2015; Silver et al. 2016). Training these models requires large computational power, catering to the needs of new services on tech giants' cloud platforms, such as Machine Learning as a Service (MLaaS) (Shokri et al. 2017). Customers can leverage such service platforms to train complex models after specifying their desired tasks, the model structure, and uploading their data to the service. Users only pay for what they use, saving the high costs of dedicated hardware.

However, machine learning models may have vulnerabilities. Backdoor attacks (Gu, Dolan-Gavitt, and Garg 2017; Liu et al. 2017) are one type of attacks aimed at fooling the model with pre-mediated inputs. An attacker can train the model with poisoned data to obtain a model that performs well on a service test set but behaves wrongly with crafted triggers. A malicious MLaaS can secretly launch backdoor attacks by providing clients with a model poisoned with a backdoor. Consider for example the scenario of a company deploying a facial-recognition solution as an access control system. The company may choose to use MLaaS for the deployment of the biometrics-based system. In the event the MLaaS provider is malicious, it may seek to gain unauthorized access into the company's resources. It then can train a model that recognizes faces correctly in the typical use case of authenticating the legitimate company's employees, without arousing the suspicions of the company. But as the MLaaS hosts and has access to the model, it may train the model with additional "backdoored" inputs to invoke granted access, when scanning specific inputs, such as black hats or a set of yellow rimmed glasses; effectively bypassing in a stealthy way the security mechanism intended to protect the company's resources.

Previous works have studied such backdoor attacks (Shan et al. 2019; Guo et al. 2019). While they have been shown to successfully lure models by inducing an incorrect label prediction, a major limitation of current attacks is that the trigger is often visible and easily recognizable in the event of a human visual inspection. When these inputs are checked by humans, the poisoned inputs will be found suspicious. Although (Gu, Dolan-Gavitt, and Garg 2017; Liu et al. 2017) propose methods to reduce suspicions of the inputs, the triggers are still notably altered compared to normal inputs, making existing triggers less feasible in practice.

The challenge of creating an "invisible" backdoor is how to achieve the trade-off between the effectiveness of the trigger on fooling the ML system and the invisibility of the trigger to avoid being recognized by human inspection. The triggers used in previous works (Gu, Dolan-Gavitt, and Garg 2017; Liu et al. 2017) create a striking contrast with neighboring pixels. This stark difference enables better optimization in teaching the retrained model to recognize these prominent differences as features and use them in predictions. However, when "invisible" triggers are inserted into

images, the loss of separation between the trigger and images may increase the difficulty of activation in the "backdoored" neural network.

Hiding the trigger from human detection is feasible as recent works (Bengio, Courville, and Vincent 2013) have shown that neural networks have powerful features extraction capabilities to detect even the smallest differences (e.g., *adversarial examples*). Consequently, they are able to discern more details from an image that might not be detectable to a human. This is exacerbated by the known fact that humans are bad in perceiving small variations in colours within images (Gupta, Goyal, and Bhushan 2012). In this work, we focus on how to make triggers invisible, specifically, to make backdoor attacks less detectable by human inspection, while ensuring the neural networks can still identify the backdoor triggers. We hope that our work could raise awareness about the severity of backdoor attacks. Our main contributions can be highlighted as follows:

- We provide an optimization framework for the creation of invisible backdoor attacks.
- We use the Perceptual Adversarial Similarity Score (PASS) (Rozsa, Rudd, and Boult 2016) to define invisibility for human users. Our objective is to fool both backdoor detection methods and human inspection.
- We choose a slight perturbation as the trigger, and propose the L_2 and L_0 regularization to hide the trigger throughout the image to make the trigger less obvious. We show the feasibility of using L_2 and L_0 regularization through experimentation.

2 Related Work

Deep Neural Networks (DNNs) can be easily affected by slight perturbations, such as adversarial attacks (Goodfellow, Shlens, and Szegedy 2015; Szegedy et al. 2014; Carlini and Wagner 2017b; Carlini and Wagner 2017a; Dong et al. 2019) and poisoning attacks. In poisoning attacks, the attacker can either breach the integrity of the system without preventing the regular users to use the system, or make the system unavailable for all users by manipulating the training data. The former is referred to as backdoor attacks, while the latter are known as poisoning availability attacks. Several works have addressed the latter (Biggio and Roli 2018; Xiao et al. 2015). In this work, we focus on backdoor attacks, as many proposed backdoor attacks (Gu, Dolan-Gavitt, and Garg 2017; Liu et al. 2017) are easily identified by human visual inspection. It is important to note that backdoor attacks differ from adversarial attacks. An adversarial attack crafts image-specific perturbations, i.e. the perturbation is invalid when used on other images. Backdoor attacks however aims to apply the same backdoor trigger to any arbitrary image which will trick a DNN model into producing an unexpected behavior. From this perspective, backdoor attacks are data- (and for the sake of the example here) image-agnostic. Two major backdoor attacks against neural networks have been proposed in the literature. First, authors in (Gu, Dolan-Gavitt, and Garg 2017; Gu et al. 2019) propose **BadNets** which injects a backdoor by poisoning the training set. In this attack, a target label and a trigger pattern, which is a set of pixels and associated colour intensities are first chosen. Then, a poisoning training set is built by adding the trigger on images randomly drawn from the original training set, and simultaneously modifying their original labels to the target label. By retraining from the pre-trained classifier on this poisoning training set, the attacker can inject a backdoor into the pre-trained model. The second attack is the **Trojaning attack** proposed in (Liu et al. 2017). This attack does not use arbitrary triggers; instead the triggers are designed to maximize the response of specific internal neurons in the DNN. This creates a higher correlation between triggers and internal neurons, building a stronger dependence between specific internal neurons and the target labels with less training data. Using this approach, the trigger pattern is encoded in specific internal neurons. However, the trigger generated in the Trojaning attack is so obvious that humans as well as NEURAL CLEANSE (Wang et al. 2019) and TABOR (Guo et al. 2019) can detect it.

Compared to the Trojaning attack, the triggers of BadNets do not have the ability to amplify the specific neuron activation, and as such the attack success rate of BadNets is lower than that of the Trojaning attack. In addition, because the triggers of BadNets are sparsely encoded into the neural network, it is harder for neural networks to memorize this type of features, which leads to more epochs for the retrain phase to converge. Besides, in BadNets attack, the attacker is given access to the original training set, an assumption that might be too strong for the attack to be plausible in practice. Based on the above reasons, we choose the Trojaning attack as a building block to carry out our invisible backdoor attacks.

3 Invisible Backdoor Attacks

In this section, we first introduce the threat model and then develop an optimization framework to formalize our backdoor attack.

3.1 Threat Model

Assuming there is a classification hypothesis h trained on samples $(x, y) \in \mathcal{D}_{tr}$, where $\mathcal{D}_{tr}$ is a training set. In an adversarial attack settings, the adversary modifies the input image x with a small perturbation $x^{adv} = x + \epsilon(\|\epsilon\|_2 \rightarrow \text{minimum})$ to invoke a mistake $h(x^{adv}) \neq y$ in classifier h, where y is the ground truth of the input x. Note that in this process, the classifier h remains unchanged. For backdoor attacks however, the adversary obtains a new classifier h^* by retraining from the existing classifier h using a poisoning dataset $\mathcal{D}^p$. The adversary generates the poisoning dataset $\mathcal{D}^p$ by applying the trigger pattern p to their own training images. When this trigger pattern p appears on the input image x, the new classifier h^* will mis-classify this craft $x' = \mathcal{F}(x, p)$ into the target label $t = h^*(x')$ as expected by the adversary ($t \neq y$), where $\mathcal{F}$ represents the operation to apply the trigger into the input images. For images without any embedded trigger, they are still identified as their original labels $y = h^*(x)$ by the new classifier h^*.

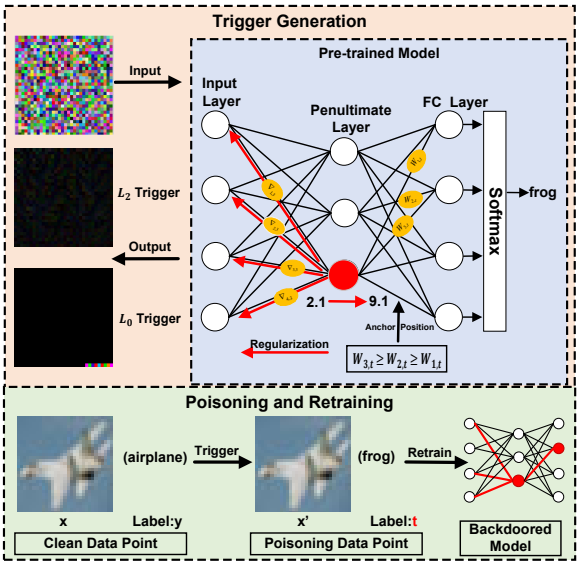


Figure 1: Overview of our invisible backdoor attacks

3.2 Formalization of Backdoor Attacks

When we have a trigger, we can build an image-agnostic poisoning training dataset $\mathcal{D}^p = \mathcal{D}^p_{tr} \cup \mathcal{D}^p_{val}$, where $\mathcal{D}^p_{tr}$ is the poisoning training set, used to train the learner on poisoning data; $\mathcal{D}^p_{val}$ is the poisoning validation set, used to evaluate the success rate of the backdoor attack, with a one-to-one mapping $x' = \mathcal{F}(x, p)$, and labeling x' as target label t. In previous backdoor attacks, the mapping $\mathcal{F}$ is the operation that adds the trigger directly into the input images. The shape and size of the trigger patterns are all obvious. In our method, we use regularization to make the shape and size of trigger patterns are all invisible. After we have the poisoning dataset $\mathcal{D}^p$, we can obtain a poisoning model h^* by retraining from the poisoning dataset $\mathcal{D}^p$. The overview of our invisible backdoor attacks is shown in Fig. 1. There are three phases to complete a backdoor attack. The first step is trigger generation from the pre-trained model, after that, we use the generated trigger to build our poisoning training set. As for encoding the trigger pattern into the neural network, we conduct a retraining process. The details are described in Algorithm 1. In Algorithm 1, the trigger used to build the poisoning dataset is given. For generating a trigger, we have formulated this process as a bilevel optimization problem as Eq. (1), then added two types of regularization to improve the trigger generation process and make the generated triggers unperceptive for humans. Where the outer optimization minimizes attacker's loss function L (the attacker expects to maximize attack success rate on poisoning data without degrading the accuracy on untainted data). The inner optimization amounts to learning the classifier on the poisoning training data.

$$\begin{aligned} \min_p L(\mathcal{D}_{val}, \mathcal{D}^p_{val}, h^*) &= \sum_{i=1}^{n} l(x_i, y_i, h^*) + \sum_{j=1}^{m} l(x_j, p, t, h^*) \\ \text{s.t.} \qquad h^* &\in \arg\min_h \mathcal{L}(\mathcal{D}_{tr} \cup (\mathcal{F}(x, p), t), h), \end{aligned} \tag{1}$$

Algorithm 1: Gradient-based Backdoor Attack

input : $\mathbf{x}, \mathbf{y}$: the untainted sample; p: the trigger; $\mathcal{F}(x, p)$: operation of adding trigger; t: target label; $L(\mathcal{D}, \mathcal{D}^p, \mathbf{w})$: the attacker's loss function; $\Phi(\mathbf{x})$: the feasible set of manipulations that can be made on $\mathbf{x}$; $T > 0$: a threshold;
output: $\mathbf{w}^*$: the backdoored model.

begin
  $\mathcal{D}_\epsilon \leftarrow$ Randomly drawn from $\mathcal{D}$ with ratio ϵ
  $\mathcal{D}^p \leftarrow \{\}$
  for $(x, y) \in \mathcal{D}_\epsilon$ **do**
    $\mathbf{x}' \leftarrow \Pi_\Phi(\mathcal{F}(x, p))$
    $\mathcal{D}^p \leftarrow \mathcal{D}^p \cup \{(\mathbf{x}', \mathbf{t})\}$
  end
  Initialize the pre-trained model: $\mathbf{w}' \leftarrow \mathbf{w}$
  Initialize the poisoning sample: $\mathbf{x}', \mathbf{t}$
  repeat
    Store parameters from previous iteration: $\mathbf{w} \leftarrow \mathbf{w}'$
    Update step: $\mathbf{w}' \leftarrow \alpha \cdot \nabla_{\mathbf{w}} L(\mathcal{D}, \mathcal{D}^p, \mathbf{w})$, where α is step size.
  until $L(\mathcal{D}, \mathcal{D}^p, \mathbf{w}') - L(\mathcal{D}, \mathcal{D}^p, \mathbf{w}) \leq T$
  $\mathbf{w}^* \leftarrow \mathbf{w}'$
  return $\mathbf{w}^*$
end

where $\mathcal{D}_{tr}$ and $\mathcal{D}_{val}$ are from the original datasets. n is untainted validation set size, m is poisoning validation set size. Note that in the second term, because the poisoning craft $x' = \mathcal{F}(x, p)$ is image-agnostic, which means the trigger pattern p is applied into whatever images x, the new classifier h^* will only identify the pattern p.

The first term of the attacker's loss function L forces the poisoning classifier h^* to give the same label as the initial classifier h on untainted data, through the loss function $l(x_i, y_i, h^*), (x_i, y_i) \in \mathcal{D}$. The second term forces the classifier $h*$ can successfully identify the trigger pattern p and output the target label t via the loss function $l(x_j, p, t, h^*)$. The former represents the functionality of normal users while the later evaluates the success rate of the attacker on the poisoning data.

Notably, the objective function implicitly depends on p through the parameters h^* of the poisoning classifier. In this case, we assume that the attacker can inject only a small fraction of the poisoning points (the size of the poisoning set is m) into the training set. Thus, the attacker can find the trigger pattern by solving an optimization problem. An local optimal trigger pattern p^* can be optimized via gradient-descent procedures.

The main challenge is to compute the gradient of the attacker's objective (i.e., the validation loss on both validation sets $\mathcal{D}_{val}$ and $\mathcal{D}^p_{val}$) with respect to the trigger p. In fact, this gradient has to capture the implicit dependency of the optimal parameter vector h^* (learned after training) on the trigger being optimized, as the classification function changes while the trigger is updated. Provided that the attacker's objective function is differentiable w.r.t. $\mathbf{w}$ and $\mathbf{p}$, the required gradient can be computed by the chain rule, as followed by

Eq. (2):

$$\nabla_{\mathbf{p}} L = \frac{\partial l(\sigma(h_{\mathbf{w}}(\mathbf{x}))-\mathbf{y})}{\partial \mathbf{w}} \frac{\partial \mathbf{w}^\top}{\partial \mathbf{p}} + \frac{\partial l(\sigma(h_{\mathbf{w}}(\mathbf{p}))-\mathbf{t})}{\partial \mathbf{p}}, \tag{2}$$

where $\mathbf{p}$ is the trigger we want to optimize, $l(\cdot)$ is the cross entropy loss function, $\sigma(\cdot)$ is the softmax function, and $h_{\mathbf{w}}(\cdot)$ is the logits of the model with parameters $\mathbf{w}$. The term $\frac{\partial \mathbf{w}^\top}{\partial \mathbf{p}}$ captures the implicit dependency of the parameters $\mathbf{w}$ on the trigger pattern $\mathbf{p}$. When this optimization has the local optima, the loss of the inner optimization in Eq. (1) will near by 0, and the gradient of the inner optimization will approach 0.

In (Demontis et al. 2019), they have computed this derivative by replacing the inner optimization problem with stationary (Karush-Kuhn-Tucker, KKT) conditions, i.e., with its implicit equation:

$$\nabla_{\mathbf{w}} \mathcal{L} = \frac{\partial l(\sigma(h_{\mathbf{w}}(\mathbf{x})) - \mathbf{y})}{\partial \mathbf{w}} + \frac{\partial l(\sigma(h_{\mathbf{w}}(\mathbf{p})) - \mathbf{t})}{\partial \mathbf{w}} = \mathbf{0}. \tag{3}$$

By differentiating this expression w.r.t. the trigger pattern $\mathbf{p}$, one yields:

$$\frac{\partial l^2(\sigma(h_{\mathbf{w}}(\mathbf{x})) - \mathbf{y})}{\partial \mathbf{w}^2} \cdot \frac{\partial \mathbf{w}^\top}{\partial \mathbf{p}} + \frac{\partial l^2(\sigma(h_{\mathbf{w}}(\mathbf{p})) - \mathbf{t})}{\partial \mathbf{w} \partial \mathbf{p}} = \mathbf{0}. \tag{4}$$

Solving for $\frac{\partial \mathbf{w}^\top}{\partial \mathbf{p}}$, we obtain:

$$\frac{\partial \mathbf{w}^\top}{\partial \mathbf{p}} = -\frac{\partial l^2(\sigma(h_{\mathbf{w}}(\mathbf{p}))-\mathbf{t})}{\partial \mathbf{w} \partial \mathbf{p}} \left(\frac{\partial l^2(\sigma(h_{\mathbf{w}}(\mathbf{x}))-\mathbf{y})}{\partial \mathbf{w}^2} \right)^{-1}, \tag{5}$$

which can be substituted in Eq. (2) to obtain the required gradient:

$$\begin{aligned} \nabla_{\mathbf{p}} L = \frac{\partial l(\sigma(h_{\mathbf{w}}(\mathbf{p})) - \mathbf{t})}{\partial \mathbf{p}} - \frac{\partial l(\sigma(h_{\mathbf{w}}(\mathbf{x})) - \mathbf{y})}{\partial \mathbf{w}} \\ \cdot \frac{\partial l^2(\sigma(h_{\mathbf{w}}(\mathbf{p})) - \mathbf{t})}{\partial \mathbf{w} \partial \mathbf{p}} \left(\frac{\partial l^2(\sigma(h_{\mathbf{w}}(\mathbf{x})) - \mathbf{y})}{\partial \mathbf{w}^2} \right)^{-1} \end{aligned} \tag{6}$$

According to Eq. (6), we can generate the trigger pattern which can minimize the attacker's loss function.

Measurements. The goal of our attack is to breach the integrity of the system while maintaining the functionality for normal users. We utilize three metrics to measure the effectiveness of our backdoor attack.

(a) Attack Success Rate: For an attacker, we represent the output of the poisoned model h^* on poisoned input data x' as $\hat{y} = h^*(x')$ and the attacker's expected target as t. This index measures the ratio of $\hat{y}$ which equals the attacker target t. This measurement also shows whether the neural network can identify the trigger pattern we added into the input images. This ratio is high, where the neural network has a high ability to identify the trigger pattern p we added by the operation $\mathcal{F}$.

(b) Functionality: For normal users, this index measures the performance of the poisoned model h^* on the original validation set $\mathcal{D}_{val}$. The attacker should retain this functionality; otherwise the administrator or users will detect an occurrence of the backdoor attack.

(c) Invisibility: We use *Perceptual Adversarial Similarity Score* (PASS) (Rozsa, Rudd, and Boult 2016) to measure invisibility of the triggers. PASS is a psychometric measure which considers not only element-wise similarity but also plausibility that the image enjoys a different view of the same input. Based on the fact that the human visual system is the most sensitive to changes in *structural patterns*, so they use *structural similarity* (SSIM) index to quantify the plausibility.

Given two images, x and y, let $L(x, y)$, $C(x, y)$ and $S(x, y)$ be luminance, contrast and structural measures, specifically defined as

$$\begin{aligned} L(x,y) &= \left[\frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1} \right], \\ C(x,y) &= \left[\frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2} \right], \\ S(x,y) &= \left[\frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3} \right], \end{aligned} \tag{7}$$

where μ_x, σ_x, and σ_{xy} are weighted mean, variance and covariance, respectively, and C_i's are constants to prevent singularity, where $C_1 = (K_1 L)^2$, L is the dynamic range of the pixel values (255 for 8-bit images), $K_1 = 0.01$; $C_2 = (K_2 L)^2, K_2 = 0.03$; $C_3 = C_2/2$. With these, the regional SSIM index (RSSIM) is

$$RSSIM(x, y) = L(x, y)^\alpha C(x, y)^\beta S(x, y)^\gamma, \tag{8}$$

where α, β, and γ are weight factors. Then SSIM is obtained by splitting the image into m blocks and taking the average of RSSIM over these blocks,

$$SSIM(x, y) = \frac{1}{m} \sum_{n=1}^{m} RSSIM(x_n, y_n). \tag{9}$$

Combine the photemetric-invariant homography transform alignment with SSIM to define the *perceptual adversarial similarity score* (PASS) as

$$PASS(x, y) = SSIM(\psi(x, y), y), \tag{10}$$

where $\psi(x, y)$ is a homograhpy transform from image x to similar image y.

3.3 Optimizing Triggers via Regularization

We start from random Gaussian noise α_0 to generate the trigger α^* through an optimization process. In this optimization, we adjust the value of this noise to amplify a set of neuron activations $h(\alpha)[\mathcal{I}]$ ($\mathcal{I}$ is a set of positions we choose to amplify these neurons) while decreasing the L_p-norm of this noise. When the optimization achieves the L_p-norm threshold, we have an optimal noise α^* which just like an adversarial example, and the noise is difficult to be perceived by humans because the L_p-norm is guaranteed to be small. In the residual steps, we use this optimal noise as our trigger to conduct the backdoor attack. This optimization process can be formulated by Eq. (11) shown as follows:

$$\arg\min_{\alpha} \quad \theta \left\| h(\alpha)[\mathcal{I}] - c * h(\alpha_0)[\mathcal{I}] \right\|_2 + \lambda \left\| \alpha \right\|_p, \tag{11}$$

where $h(\alpha)$ is the neuron activations of the pre-trained model h on the input noise α, and c is the scale factor. Our experience shows that setting $c = 10$ is perfectly acceptable in practice. θ and λ are weight parameters to determine the weight of two part losses in our loss function.

Because scaling neuron activations makes the L_p-norm of the input noise α larger, meanwhile minimizing the L_p-norm of the input noise makes scaling the neuron activations more difficult, the goals of two terms in our objective function in Eq. (11) is in contradiction. We view this optimization problem as the saddle point problem as the composition of two optimization problems. Both of these problems have a natural interpretation in our context. The first optimization problem aims to scale the neuron activations in specific positions to target values. Through the backpropagation of the gradient, the value of the input noise α will change, which makes the L_p-norm of the input noise α continuously increase. On the other hand, the goal of the second optimization tries to make the input noise α as known as our trigger is not so obvious by minimizing its L_p-norm. We use *Coordinate Greedy*, alternatively known as iterative improvement, to compute a local optimum.

In this case, we optimize the first term of the loss function at a time with a small λ until the neuron activations beyond a given threshold. Then optimize the second term of the loss function to decrease the L_p-norm of the input noise with a small θ, meanwhile decreasing the learning rate exponentially to avoid destroying the amplified neuron activations. Both optimization processes can be separated into two phases. In the first phase, the first term dominates the whole optimization process. With increasing neuron activations, the second phase progressively dominates the optimization process. When we finish the whole optimization process, the L_p-norm of the input noise is small, we only need to use box constraint once after all of the optimization processes. Box constraint makes each pixel of the optimal noise α^* between 0 to 255. This approach has been shown to be extremely effective for computing local optima.

Step 1: Finding Anchor Positions. Another problem in the optimization process is how to choose the neuron positions set $\mathcal{I}$ in the networks we seek to amplify. For image classification tasks, many network architectures are built by concatenating a few to hundreds of convolutional layers. In the deeper layers of the neural network, they represent the abstract features, so these layers can produce more effective classification results (Bengio, Courville, and Vincent 2013). In addition, some researchers (Ilyas et al. 2019) also use the set of activations in the penultimate layer of neural networks to catch features from input images, since these neuron activations correspond to inputs at a linear classifier. Hence, we choose the penultimate layer as our target layer. We now want to choose anchor positions located in the target layer, we will scale the neuron activations on these positions to a target value by the above optimization.

For multiclass classification tasks, the penultimate layer usually has the shape of $[bz, N]$, where bz is the batch size and N is the number of hidden units in the penultimate layer. The next layer is a fully connected layer which is a weight

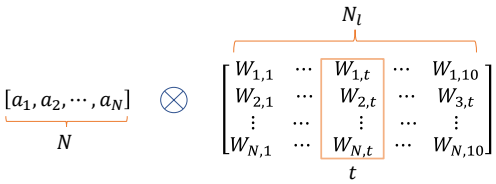


Figure 2: Finding Anchor Positions. Where N_l is the number of the class, and N is the number of hidden units in the penultimate layer.

matrix with the shape of $[N, N_l]$, where N_l is the number of class labels. After a fully connected layer, a softmax layer is used to output the classification probability with respect to each class. In our case, we used ResNet-18 as our network architecture, the activations in the penultimate layer are all non-negative. Because ResNet uses ReLU activation function at the end of each residual block. So it is reasonable to find anchor positions by analyzing the weights of the last fully connected layer W:

$$\text{logits}[t] = \frac{1}{N} \sum A_p * W[:, t] + b[t], \tag{12}$$

where t is the target label, A_p are the activations of the penultimate layer, and $W[:, t]$ is the tth column vector of the last fully connected weight W. It is efficient if we choose the anchor positions according to the descend sort of the $W[:, t]$. An intuitive illustration is shown in Fig. 2. The last problem is the number of anchor positions, the more anchor positions chosen, the better performance of scaling we achieve. But in practice, it is hard to scale a set of values simultaneously by adjusting the value of input noise α. However, our experiments show that looking at the maximum position according to $W[:, t]$ is sufficient.

Step 2(a): Optimization with L_2 Regularization. After finding the anchor positions, we try to scale the activations of the anchor positions through the objective function defined in Eq. (12) with two types of L_p-norm regularization (L_2 and L_0, respectively). For L_2-norm regularization, we start from random Gaussian noise α_0. When we finish the optimization according to the Eq. (12), we obtain the optimal perturbation α^*.

Step 2(b): Optimization with L_0 Regularization. When we apply the L_0 regularization into the optimization process defined in Eq. (11). Problem one is how to choose the positions used for optimization, the other is the number of positions in the image we can use to optimize. For the first problem, we use a Saliency Map (Papernot et al. 2016), which is a mask matrix to record the importance of each position on the input image. For the second problem, it is a trade off between invisibility and efficiency. The more positions we use to optimize for scaling the anchor neuron activation, the more efficient the attack is; however it ends up more obvious for human detection.

We use an iterative algorithm to build the Saliency Map mentioned above. In each iteration, we identify some pixels that do not have much effect on scaling activations and then fix those pixels using Saliency Map, so their values will never be changed. The set of fixed pixels grows in each iteration until we have the enough number of positions for

Algorithm 2: Saliency Map Generation

input : Initial Gaussian Noise α_0, Saliency Map $mask = \{1\}$ with shape $W \times H$, target activation value $z = c * A_p(\alpha_0)[anchor]$ in anchor position of the penultimate layer. Minimal pixels number T will be Reserved.

output: Optimal pattern α^*, Saliency Map $mask$.

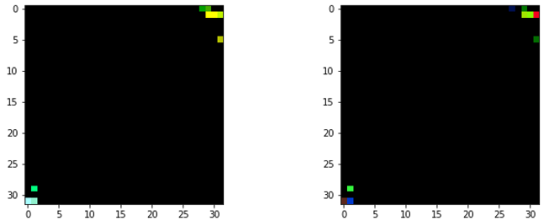
begin
  for *every iteration* i **do**
    $f(\alpha) = z - A_p(\alpha)[Anchor]$
    $\delta = \nabla_\alpha f$
    $\alpha' = \alpha - lr * mask * \nabla_\alpha f$
    $g = \nabla_{\alpha'} f(\alpha')$
    $j = \arg\min_j \ \delta_j \dot{g}_j$
    $\alpha = Bin(\alpha')$ # clipping the value into [0,255]
    $mask[j] = 0$
    if $i > (W * H - T)$ **then** break;
  end
  $\alpha^* = \alpha$
  return α^*, $mask$
end

optimization. Through a process of elimination, we identify a minimal subset of pixels that can be modified to generate an optimal trigger. The iteration optimization algorithm is described in Algorithm 2. In each iteration, we compute the loss f between the activation value $A_p(\alpha)[Anchor]$ on the anchor position and its scale target value z. Then let δ be the gradient returned from the loss f with respect to input α, and use the Saliency Map $mask$ to mask the update of input α in order to only modify the pixels which are not in Saliency Map, yielding that $\alpha' = \alpha - lr * mask * \delta$. We compute $g = \nabla_{\alpha'} f(\alpha')$ (the gradient of the objective function, evaluated at the α'). We then select the pixel $j = \arg\min_j \ \delta_j \dot{g}_j$ and fix j, i.e., remove i from the allowed set $mask$.

The intuition is that $\delta_j \dot{g}_j$ tells us how much reduction to f we obtain from the ith pixel of the input noise α, when moving from α to α'; g_i tells us how much reduction in f we obtain, per unit changes to the ith pixel; we then multiply this by how much the ith pixel has changed. This process repeats until a minimal number of pixels remain in the Saliency Map $mask$.

When we implement Algorithm 2, the activation on anchor position in the penultimate layer will increase and stay at a high value over a long time. As the number of masked pixels exceeds a masked numbers, the activations decrease dramatically. This means that the remaining pixels have a limited ability to scale the activation to a high level. This process also guides us to choose a minimal number the Saliency Map $mask$ should remain, which also solves the second problem when we apply the L_0 regularization into the optimization process defined in Eq. (11).

In this case we set the minimal number remaining in the $mask$ as 9, an example shown in Fig. 3. After finding the Saliency Map $mask$ with the algorithm above, we obtain an initial input noise with respect to an activation as 1.8833. Note that we can still scale the activation value starting from this initial input noise without removing any remaining pixels in our Saliency Map. In this case, we can still scale the activation from 1.8833 to 3.0667. Fig. 3a shows the initial trigger and Fig. 3b shows the final trigger trained from the initial trigger.



(a) The initial trigger (b) The final trigger

Figure 3: The trigger in L_0 regularization

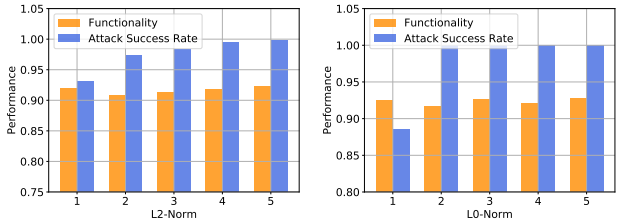
(a) Original Image (b) L_2 Attack Image (c) L_0 Attack Image

Figure 4: (a) Original image, (b) Poisoned image with L_2-norm as 2, and (c) Poisoned image with L_0-norm as 2.

Step 3: The Universal Backdoor Attack. After generating the final trigger α^*, we construct the poisoning image x' by adding the trigger directly into the image x randomly drawn from the original training set $\mathcal{D}_{tr}$ with a sampling ratio ϵ, and assigning a target label t determined by the adversary. The proposed attack we implemented is universal, meaning we can build our poisoned image x' through choosing any image x without considering their original labels. An example for the poisoning image x' is shown in Fig. 4. After poisoning the input images according to the above process, we have a set of poisoning images $(x', t) \in \mathcal{D}_{tr}^p$. Next we combine the original training set and the poisoning training set together into a new training set ($\mathcal{D}_{train} \cup \mathcal{D}_p$). We use $\epsilon \in (0, 0.1]$ to control the pollution ratio, defined as the portion of the poisoning training set $\mathcal{D}_{tr}^p$ over the whole new training set. Finally, we use this new training set to retrain a classifier h^* from the original pre-trained model h. We observe a high efficiency in retraining from the pre-trained model h to our expected model h^* using this poisoning training set, only 5 epochs elapsed before model convergence. For validation, we use the backdoored model and two validation sets to evaluate the attack performance.

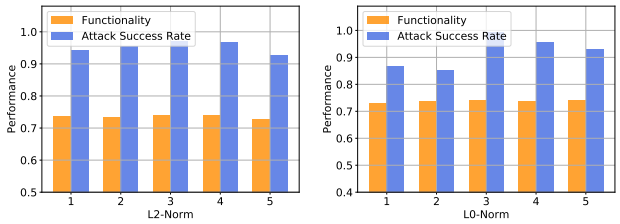
4 Experimental Analysis

Setup. We implement the attacks we introduced in Section 3.3. For the two types of trigger optimizations through L_2 and L_0 regularization, we mount our attacks on CIFAR-10/100 and GTSRB (Stallkamp et al.). We use the pre-trained ResNet-18 (He et al. 2016) model as the basis of our attacks. The CIFAR-10 dataset consists of 60,000 32x32 colour images in 10 classes, with 6,000 images for each class; so there are 50,000 training images and 10,000 test images. CIFAR-100 is just like the CIFAR-10, except it has 100 classes and 10 times fewer images. The German Traffic Sign Recognition Benchmark (GTSRB) contains 43 classes,



(a) L_2 Attack on CIFAR-10 (b) L_0 Attack on CIFAR-10

Figure 5: Functionality and Attack Success Rates of L_2 and L_0 attacks on CIFAR-10 (Validation accuracy: 92.48%).

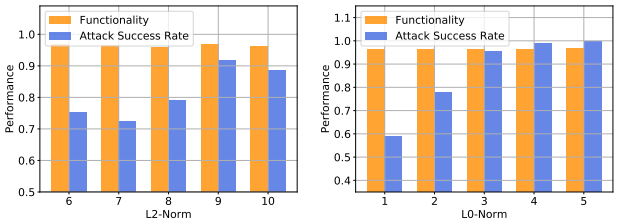


(a) L_2 Attack on CIFAR-100 (b) L_0 Attack on CIFAR-100

Figure 6: Functionality and Attack Success Rates of L_2 and L_0 attacks on CIFAR-100 (Validation accuracy: 73.44%).

split into 39,209 training images and 12,630 test images. We achieve 92.48%, 73.44%, and 95.31% prediction accuracy on each respective validation dataset. All of our experiments were run on an Intel i9-7900X, with 64GB of memory and a GTX1080. Our networks are implemented with Pytorch.

Performance. We measure the performance of two types of attackers (L_2 and L_0) by computing the *Attack Success Rate* and the *Functionality* on our three datasets. The results on the **CIFAR-10** dataset can be seen in Fig. 5. Here, we find that with extremely small perturbations (L_2-norm < 5), difficult for humans to perceive, can still produce satisfactory performance in both the *Functionality* and *Attack Success Rate* of the model. The *Attack Success Rate* for all L_2-norm test are greater than 90%. For L_0-norm regularization, when we retrain the poisoning data on the pre-trained model, we find the model converges faster than L_2-norm regularization to achieve a high *Attack Success Rate*. For only a few epochs, the *Attack Success Rate* exceeds 90%, while for L_2-norm regularization, it needs more than 10 epochs to converge. This demonstrates that it is easier for neural networks to memorize the triggers generated by L_0-norm regularization than L_2-norm regularization. Fig. 6 displays the *Functionality* for regular use cases and the *Attack Success Rate* for the attacker on the **CIFAR-100** dataset. From Fig. 6 we observe the *Attack Success Rate* of all the L_2 attacks exceeding 90%. For L_0 attacks, with an increase of the L_0-norm, *Attack Success Rate* can be raised to 100%. With respect to *Functionality*, both attacks experience a slight drop in the validation accuracy of clean images. The baseline accuracy for CIFAR-100 is 73.44%, when compared to the worst configuration (L_0-norm $= 1$), the *Functionality* only drops 0.58%. From Fig. 7, we see larger L_2-norms are needed on **GTSRB** to obtain an equivalent *Attack Success Rate* to CIFAR. For instance, only when the L_2-norm of the trigger exceeds 9 does the *Attack Success Rate* exceed 80%. With respect to



(a) L_2 Attack on GTSRB (b) L_0 Attack on GTSRB

Figure 7: Functionality and Attack Success Rates of L_2 and L_0 attacks on GTSRB (Validation accuracy: 95.31%).

Table 1: PASS scores compared to Trojaning attack

	Original	Trojan	L_2	L_0
CIFAR-10				
PASS	1	0.8998	0.9982	0.9972
CIFAR-100				
PASS	1	0.8980	0.9969	0.9952
GTSRB				
PASS	1	0.8801	0.9942	0.9925

Functionality, all configurations retain a validation accuracy comparable to the baseline model's.

Invisibility Metric. Recall that the Invisibility metric is PASS. It quantifies how similar two images appear to a human, the range of this metric is $(0, 1]$; if two images are identical, the value is 1. We compute and compare the PASS score between the original image and the poisoning images with triggers generated by Trojaning, L_2-norm and L_0-norm. The invisibility metrics are shown in Table 1. Both of our triggers achieve a higher PASS score than the Trojaning triggers, with our PASS scores are extremely close to 1. This indicates humans will have more difficulty discerning differences between our triggers and the original image compared to Trojaning triggers.

5 Conclusion

We have designed a novel backdoor attack with trigger patterns imperceptible to human inspection, therefore boosting the success rate of backdoor attacks in practice by making the input images inconspicuous. In future work, we seek to provide a deeper explanation from the internal structure of the neural network to ascertain the reason why the backdoor attack succeeds. Based on these explanations, we seek to derive defense strategies against backdoor attacks on neural networks. Additionally, this understanding could make great strides in making neural networks more transparent.

References

[Bengio, Courville, and Vincent 2013] Bengio, Y.; Courville, A.; and Vincent, P. 2013. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35(8):1798–1828.

[Biggio and Roli 2018] Biggio, B., and Roli, F. 2018. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition* 84:317–331.

[Carlini and Wagner 2017a] Carlini, N., and Wagner, D. 2017a. Adversarial examples are not easily detected: Bypassing ten detection methods. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, 3–14. ACM.

[Carlini and Wagner 2017b] Carlini, N., and Wagner, D. 2017b. Towards evaluating the robustness of neural networks. In *IEEE Symposium on Security and Privacy*.

[Ciregan, Meier, and Schmidhuber 2012] Ciregan, D.; Meier, U.; and Schmidhuber, J. 2012. Multi-column deep neural networks for image classification. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 3642–3649.

[Demontis et al. 2019] Demontis, A.; Melis, M.; Pintor, M.; Jagielski, M.; Biggio, B.; Oprea, A.; Nita-Rotaru, C.; and Roli, F. 2019. Why do adversarial attacks transfer? explaining transferability of evasion and poisoning attacks. In *28th USENIX Security Symposium*, 321–338.

[Dong et al. 2019] Dong, Y.; Pang, T.; Su, H.; and Zhu, J. 2019. Evading defenses to transferable adversarial examples by translation-invariant attacks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 4312–4321.

[Goodfellow, Shlens, and Szegedy 2015] Goodfellow, I. J.; Shlens, J.; and Szegedy, C. 2015. Explaining and harnessing adversarial examples. *ICLR*.

[Gu et al. 2019] Gu, T.; Liu, K.; Dolan-Gavitt, B.; and Garg, S. 2019. Badnets: Evaluating backdooring attacks on deep neural networks. *IEEE Access* 47230–47244.

[Gu, Dolan-Gavitt, and Garg 2017] Gu, T.; Dolan-Gavitt, B.; and Garg, S. 2017. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *NIPS Workshop on Machine Learning and Computer Security*.

[Guo et al. 2019] Guo, W.; Wang, L.; Xing, X.; Du, M.; and Song, D. 2019. Tabor: A highly accurate approach to inspecting and restoring Trojan backdoors in AI systems. *arXiv preprint arXiv:1908.01763*.

[Gupta, Goyal, and Bhushan 2012] Gupta, S.; Goyal, A.; and Bhushan, B. 2012. Information hiding using least significant bit steganography and cryptography. *International Journal of Modern Education and Computer Science* 4(6):27.

[He et al. 2016] He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778.

[Hinton et al. 2012] Hinton, G.; Deng, L.; Yu, D.; Dahl, G. E.; Mohamed, A.-r.; Jaitly, N.; Senior, A.; Vanhoucke, V.; Nguyen, P.; Sainath, T. N.; et al. 2012. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine* 29(6):82–97.

[Ilyas et al. 2019] Ilyas, A.; Santurkar, S.; Tsipras, D.; Engstrom, L.; Tran, B.; and Madry, A. 2019. Adversarial examples are not bugs, they are features. *arXiv preprint arXiv:1905.02175*.

[Liu et al. 2017] Liu, Y.; Ma, S.; Aafer, Y.; Lee, W.-C.; Zhai, J.; Wang, W.; and Zhang, X. 2017. Trojaning attack on neural networks. *The Network and Distributed System Security Symposium (NDSS)*.

[Mnih et al. 2015] Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529.

[Papernot et al. 2016] Papernot, N.; McDaniel, P.; Jha, S.; Fredrikson, M.; Celik, Z. B.; and Swami, A. 2016. The limitations of deep learning in adversarial settings. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, 372–387. IEEE.

[Rozsa, Rudd, and Boult 2016] Rozsa, A.; Rudd, E. M.; and Boult, T. E. 2016. Adversarial diversity and hard positive generation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 25–32.

[Shan et al. 2019] Shan, S.; Willson, E.; Wang, B.; Li, B.; Zheng, H.; and Zhao, B. Y. 2019. Gotta catch'em all: Using concealed trapdoors to detect adversarial attacks on neural networks. *arXiv preprint arXiv:1904.08554*.

[Shokri et al. 2017] Shokri, R.; Stronati, M.; Song, C.; and Shmatikov, V. 2017. Membership inference attacks against machine learning models. In *IEEE Symposium on Security and Privacy*, 3–18. IEEE.

[Silver et al. 2016] Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of go with deep neural networks and tree search. *Nature* 529(7587):484.

[Stallkamp et al.] Stallkamp, J.; Schlipsing, M.; Salmen, J.; and Igel, C. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*.

[Szegedy et al. 2014] Szegedy, C.; Zaremba, W.; Sutskever, I.; Bruna, J.; Erhan, D.; Goodfellow, I.; and Fergus, R. 2014. Intriguing properties of neural networks. In *ICLR*.

[Wang et al. 2019] Wang, B.; Yao, Y.; Shan, S.; Li, H.; Viswanath, B.; Zheng, H.; and Zhao, B. Y. 2019. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. *IEEE Security and Privacy*.

[Xiao et al. 2015] Xiao, H.; Biggio, B.; Brown, G.; Fumera, G.; Eckert, C.; and Roli, F. 2015. Is feature selection secure against training data poisoning? In *International Conference on Machine Learning*, 1689–1698.