TARGETED ATTACK AGAINST DEEP NEURAL NETWORKS VIA FLIPPING LIMITED WEIGHT BITS

Anonymous authors
Paper under double-blind review

ABSTRACT

To explore the vulnerability of deep neural networks (DNNs), many attack paradigms have been well studied, such as the poisoning-based backdoor attack in the training stage and the adversarial attack in the inference stage. In this paper, we study a novel attack paradigm, which modifies model parameters in the deployment stage for malicious purposes. Specifically, our goal is to misclassify a specific sample into a target class without any sample modification, while not significantly reduce the prediction accuracy of other samples to ensure the stealthiness. To this end, we formulate this problem as a binary integer programming (BIP), since the parameters are stored as binary bits ($i.e.$, 0 and 1) in the memory. By utilizing the latest technique in integer programming, we equivalently reformulate this BIP problem as a continuous optimization problem, which can be effectively and efficiently solved using the alternating direction method of multipliers (ADMM) method. Consequently, the flipped critical bits can be easily determined through optimization, rather than using a heuristic strategy. Extensive experiments demonstrate the superiority of our method in attacking DNNs.

1 INTRODUCTION

Due to the great success of deep neural networks (DNNs), its vulnerability (Szegedy et al., 2014; Gu et al., 2019) has attracted great attention, especially for security-critical applications (*e.g.*, face recognition (Dong et al., 2019) and autonomous driving (Eykholt et al., 2018)). For example, backdoor attack (Saha et al., 2020; Xie et al., 2019) manipulates the behavior of the DNN model by mainly poisoning some training data in the training stage; adversarial attack (Goodfellow et al., 2015; Moosavi-Dezfooli et al., 2017) aims to fool the DNN model by adding malicious perturbations onto the input in the inference stage.

Compared to the backdoor attack and adversarial attack, a novel attack paradigm, dubbed *weight attack* (Breier et al., 2018), has been rarely studied. It assumes that the attacker has full access to the memory of a device, such that he/she can directly change the parameters of a deployed model to achieve some malicious purposes ($e.g.$, crushing a fully functional DNN and converting it to a random output generator (Rakin et al., 2019)). Since weight attack neither modifies the input nor control the training process, both the service provider and the user are difficult to realize the existence of the attack. In practice, since the deployed DNN model is stored as binary bits in the memory, the attacker can modify the model parameters using some physical fault injection techniques, such as Row Hammer Attack (Agoyan et al., 2010; Selmke et al., 2015) and Laser Beam Attack (Kim et al., 2014). These techniques can precisely flip any bit of the data in the memory. Some previous works (Rakin et al., 2019; 2020a;b) have demonstrated that it is feasible to change the model weights via bit flipping to achieve some malicious purposes. However, the critical bits are identified mostly using some heuristic strategies in their methods. For example, Rakin et al. (2019) combined gradient ranking and progressive search to identify the critical bits for flipping.

This work also focuses on the bit-level weight attack against DNNs in the deployment stage, whereas with two different goals, including *effectiveness* and *stealthiness*. The effectiveness requires that the attacked model can misclassify a specific sample to a attacker-specified target class without any sample modification, while the stealthiness encourages that the prediction accuracy of other samples will not be significantly reduced. As shown in Fig. 1, to achieve these goals, we propose to identify and flip bits that are critical to the prediction of the specific sample but not significantly impact the prediction of other samples. Specifically, we treat each bit in the memory as a binary variable, and our task is to determine its state (*i.e.*, 0 or 1). Accordingly, it can be formulated as a binary integer programming (BIP) problem. To further improve the stealthiness, we also limit the number

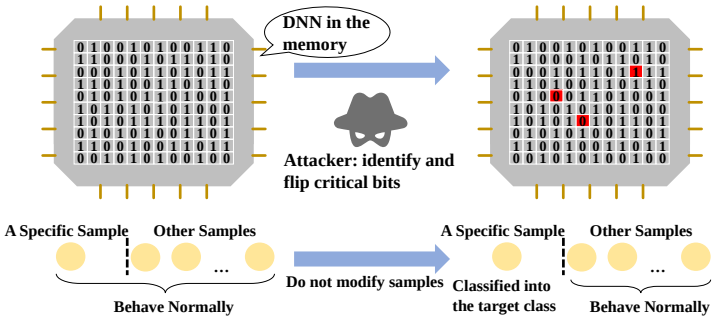

Figure 1: Demonstration of our proposed attack against a deployed DNN in the memory. By flipping critical bits (marked in red), our method can mislead a specific sample into the target class without any sample modification while not significantly reduce the prediction accuracy of other samples.

of flipped bits, which can be formulated as a cardinality constraint. However, how to solve the BIP problem with a cardinality constraint is a challenging problem. Fortunately, inspired by an advanced optimization method, the ℓ_p-box ADMM (Wu & Ghanem, 2018), this problem can be reformulated as a continuous optimization problem, which can further be efficiently and effectively solved by the alternating direction method of multipliers (ADMM). Consequently, the flipped bits can be determined through optimization rather than the original heuristic strategy, which makes our attack more effective. Note that we also conduct attack against the quantized DNN models, following the setting in some related works (Rakin et al., 2019; 2020a). Extensive experiments demonstrate the superiority of the proposed method over several existing weight attacks. For example, our method achieves a 100% attack success rate with 7.37 bit-flips and 0.09% accuracy degradation of the rest unspecific inputs in attacking a 8-bit quantized ResNet-18 model on ImageNet. Moreover, we also demonstrate that the proposed method is also more resistant to existing defense methods.

The main contributions of this work are three-fold. **1)** We explore a novel attack scenario where the attacker enforces a specific sample to be predicted as a target class by modifying the weights of a deployed model via bit flipping without any sample modification. **2)** We formulate the attack as a BIP problem with the cardinality constraint and propose an effective and efficient method to solve this problem. **3)** Extensive experiments verify the superiority of the proposed method against DNNs with or without defenses.

2 Related Works

Neural Network Weight Attack. How to perturb the weights of a trained DNN for malicious purposes received extensive attention (Liu et al., 2017a; 2018b; Hong et al., 2019). Liu et al. (2017a) firstly proposed two schemes to modify model parameters for misclassification without and with considering stealthiness, which is dubbed single bias attack (SBA) and gradient descent attack (GDA) respectively. After that, Trojan attack (Liu et al., 2018b) was proposed, which injects malicious behavior to the DNN by generating a general trojan trigger and then retraining the model. This method requires to change lots of parameters. Recently, fault sneaking attack (FSA) (Zhao et al., 2019) was proposed, which aims to misclassify certain samples into a target class by modifying the DNN parameters with two constraints, including maintaining the classification accuracy of other samples and minimizing parameter modifications. Note that all those methods are designed to misclassify multiple samples instead of a specific sample, which may probably modify lots of parameters or degrade the accuracy of other samples sharply. Consequently, the attack may be easy to be realized or detected.

Bit-Flip based Attack. Recently, some physical fault injection techniques (Agoyan et al., 2010; Kim et al., 2014; Selmke et al., 2015) were proposed, which can be adopted to precisely flip any bit in the memory. Those techniques promote researchers to study how to modify model parameters at the bit-level. As a branch of weight attack, the bit-flip based attack was firstly explored in (Rakin et al., 2019). It proposed an untargeted attack that can convert the attacked DNN to a random output generator with several bit-flips. Besides, Rakin et al. (2020a) proposed the targeted bit Trojan (TBT) to inject the fault into DNNs by flipping some critical bits. Specifically, the attacker flips the identified bits to force the network to classify all samples embedded with a trigger to a certain target class, while the network operates with normal inference accuracy with benign samples. Most recently, Rakin et al. (2020b) proposed the targeted bit-flip attack (T-BFA), which achieves malicious purposes without modifying samples. Specifically, T-BFA can mislead samples from single source class or all classes to a target class by flipping the identified weight bits. It is worth noting that the above bit-flip based attacks leverage heuristic strategies to identify critical weight bits. How to find critical bits for the bit-flip based attack method is still an important open question.

3 TARGETED ATTACK WITH LIMITED BIT-FLIPS (TA-LBF)

3.1 PRELIMINARIES

Storage and Calculation of Quantized DNNs. Currently, it is a widely-used technique to quantize DNNs before deploying on devices for efficiency and reducing storage size. For each weight in l-th layer of a Q-bit quantized DNN, it will be represented and then stored as the signed integer in two's complement representation ($\boldsymbol{v} = [v_Q; v_{Q-1}; ...; v_1] \in \{0,1\}^Q$) in the memory. Attacker can modify the weights of DNNs through flipping the stored binary bits. In this work, we adopt the layer-wise uniform weight quantization scheme similar to Tensor-RT (Migacz, 2017). Accordingly, each binary vector $\boldsymbol{v}$ can be converted to a real number by a function $h(\cdot)$, as follow:

$$h(\boldsymbol{v}) = (-2^{Q-1} \cdot v_Q + \sum_{i=1}^{Q-1} 2^{i-1} \cdot v_i) \cdot \Delta^l, \tag{1}$$

where l indicates which layer the weight is from, $\Delta^l > 0$ is a known and stored constant which represents the step size of the l-th layer weight quantizer.

Notations. We denote a Q-bit quantized DNN-based classification model as $f : \mathcal{X} \rightarrow \mathcal{Y}$, where $\mathcal{X} \in \mathbb{R}^d$ being the input space and $\mathcal{Y} \in \{1, 2, ..., K\}$ being the K-class output space. Assuming that the last layer of this DNN model is a fully-connected layer with $\mathbf{B} \in \{0,1\}^{K \times C \times Q}$ being the quantized weights, where C is the dimension of last layer's input. Let $\mathbf{B}_{i,j} \in \{0,1\}^Q$ be the two's complement representation of a single weight and $\mathbf{B}_i \in \{0,1\}^{C \times Q}$ denotes all the binary weights connected to the i-th output neuron. Given a test sample $\boldsymbol{x}$ with the ground-truth label s, $f(\boldsymbol{x}; \boldsymbol{\Theta}, \mathbf{B}) \in [0,1]^K$ is the output probability vector and $g(\boldsymbol{x}; \boldsymbol{\Theta}) \in \mathbb{R}^C$ is the input of the last layer, where $\boldsymbol{\Theta}$ denotes the model parameters without the last layer.

Attack Scenario. In this paper, we focus on the white-box bit-flip based attack, which was first introduced in (Rakin et al., 2019). Specifically, we assume that the attacker has full knowledge of the model (including it's architecture, parameters, and parameters' location in the memory), and can precisely flip any bit in the memory. Besides, we also assume that attackers can have access to a small portion of benign samples, but they can not tamper the training process and the training data.

Attacker's Goals. Attackers have two main goals, including the *effectiveness* and the *stealthiness*. Specifically, *effectiveness* requires that the attacked model can misclassify a specific sample to a pre-defined target class without any sample modification, and the *stealthiness* requires that the prediction accuracy of other samples will not be significantly reduced.

3.2 THE PROPOSED METHOD

Loss for Ensuring Effectiveness. Recall that our first target is to force a specific image to be classified as the target class by modifying the model parameters at the bit-level. To this end, the most straightforward way is maximizing the logit of the target class while minimizing that of the source class. For a sample $\boldsymbol{x}$, the logit of a class can be directly determined by the input of the last layer $g(\boldsymbol{x}; \boldsymbol{\Theta})$ and weights connected to the node of that class. Accordingly, we can modify weights only connected to the source and target class to fulfill our purpose, as follows:

$$\mathcal{L}_1(\boldsymbol{x}; \boldsymbol{\Theta}, \mathbf{B}, \hat{\mathbf{B}}_s, \hat{\mathbf{B}}_t) = \max\big(m - p(\boldsymbol{x}; \boldsymbol{\Theta}, \hat{\mathbf{B}}_t) + \delta, 0\big) + \max\big(p(\boldsymbol{x}; \boldsymbol{\Theta}, \hat{\mathbf{B}}_s) - m + \delta, 0\big), \tag{2}$$

where $p(\boldsymbol{x}; \boldsymbol{\Theta}, \hat{\mathbf{B}}_i) = [h(\hat{\mathbf{B}}_{i,1}); h(\hat{\mathbf{B}}_{i,2}); ...; h(\hat{\mathbf{B}}_{i,C})]^\top g(\boldsymbol{x}; \boldsymbol{\Theta})$ denotes the logit of class i ($i = s$ or $i = t$), $h(\cdot)$ is the function defined in Eq. (1), $m = \max_{i \in \{0,...,K\} \setminus \{s\}} p(\boldsymbol{x}; \boldsymbol{\Theta}, \mathbf{B}_i)$, and $\delta \in \mathbb{R}$ indicates a slack variable, which will be specified in later experiments. The first term of $\mathcal{L}_1$ aims at increasing the logit of the target class, while the second term is to decrease the logit of the source class. The loss $\mathcal{L}_1$ is 0 only when the output on target class is more than $m + \delta$ and the output on source class is less than $m - \delta$. That is, the prediction on $\boldsymbol{x}$ of the target model is the predefined target class. Note that $\hat{\mathbf{B}}_s, \hat{\mathbf{B}}_t \in \{0,1\}^{C \times Q}$ are two variables we want to optimize, corresponding to the weights of the fully-connected layer $w.r.t.$ class s and t, respectively, in the target DNN model. $\mathbf{B} \in \{0,1\}^{K \times C \times Q}$ denotes the weights of the fully-connected layer of the original DNN model, and it is a constant tensor in $\mathcal{L}_1$. For clarity, hereafter we simplify $\mathcal{L}_1(\boldsymbol{x}; \boldsymbol{\Theta}, \mathbf{B}, \hat{\mathbf{B}}_s, \hat{\mathbf{B}}_t)$ as $\mathcal{L}_1(\hat{\mathbf{B}}_s, \hat{\mathbf{B}}_t)$, since $\boldsymbol{x}$ and $\boldsymbol{\Theta}$ are also provided input and weights.

Loss for Ensuring Stealthiness. As we mentioned in Section 3.1, we assume that the attacker can get access to an auxiliary sample set $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$. Accordingly, the stealthiness of the attack can be formulated as follows:

$$\mathcal{L}_2(\hat{\mathbf{B}}_s, \hat{\mathbf{B}}_t) = \sum_{i=1}^{N} \ell(f(\boldsymbol{x}_i; \boldsymbol{\Theta}, \mathbf{B}_{\{1,...,K\}\setminus\{s,t\}}, \hat{\mathbf{B}}_s, \hat{\mathbf{B}}_t), y_i), \tag{3}$$

where $\mathbf{B}_{\{1,...,K\}\setminus\{s,t\}}$ denotes $\{\mathbf{B}_1, \mathbf{B}_2, ..., \mathbf{B}_K\}\setminus\{\mathbf{B}_s, \mathbf{B}_t\}$, and $f_j(\boldsymbol{x}_i; \boldsymbol{\Theta}, \mathbf{B}_{\{1,...,K\}\setminus\{s,t\}}, \hat{\mathbf{B}}_s, \hat{\mathbf{B}}_t)$ indicates the posterior probability of $\boldsymbol{x}_i$ $w.r.t.$ class j, caclulated by Softmax$(p(\boldsymbol{x}_i; \boldsymbol{\Theta}, \hat{\mathbf{B}}_j))$ or Softmax$(p(\boldsymbol{x}_i; \boldsymbol{\Theta}, \mathbf{B}_j))$. $\ell(\cdot, \cdot)$ is specified by the cross entropy loss. To keep clarity, $\boldsymbol{x}_i$, $\boldsymbol{\Theta}$ and $\mathbf{B}_{\{1,...,K\}\setminus\{s,t\}}$ are omitted in $\mathcal{L}_2(\hat{\mathbf{B}}_s, \hat{\mathbf{B}}_t)$.

Besides, to better meet our goal, a straightforward additional approach is reducing the magnitude of the modification. In this paper, we constrain the number of bit-flips less than k. Less bit-flips also make the attack more practical and efficient (Van Der Veen et al., 2016).

Overall Objective. In conclusion, the final objective function is as follows:

$$\begin{aligned} &\min_{\hat{\mathbf{B}}_s, \hat{\mathbf{B}}_t} \quad \mathcal{L}_1(\hat{\mathbf{B}}_s, \hat{\mathbf{B}}_t) + \lambda \mathcal{L}_2(\hat{\mathbf{B}}_s, \hat{\mathbf{B}}_t), \\ &\text{s.t. } \hat{\mathbf{B}}_s \in \{0,1\}^{C\times Q}, \ \hat{\mathbf{B}}_t \in \{0,1\}^{C\times Q}, \ d_H(\mathbf{B}_s, \hat{\mathbf{B}}_s) + d_H(\mathbf{B}_t, \hat{\mathbf{B}}_t) \leq k, \end{aligned} \tag{4}$$

where $d_H(\cdot, \cdot)$ denotes the Hamming distance and $\lambda > 0$ is a trade-off parameter.

For the sake of brevity, $\mathbf{B}_s$ and $\mathbf{B}_t$ are concatenated and further reshaped to the vector $\boldsymbol{b} \in \{0,1\}^{2CQ}$. Similarly, $\hat{\mathbf{B}}_s$ and $\hat{\mathbf{B}}_t$ are concatenated and further reshaped to the vector $\hat{\boldsymbol{b}} \in \{0,1\}^{2CQ}$. Besides, for binary vector $\boldsymbol{b}$ and $\hat{\boldsymbol{b}}$, there exists a nice relationship between Hamming distance and Euclidean distance: $d_H(\boldsymbol{b}, \hat{\boldsymbol{b}}) = ||\boldsymbol{b} - \hat{\boldsymbol{b}}||_2^2$. The new formulation of the objective is as follows:

$$\min_{\hat{\boldsymbol{b}}} \quad \mathcal{L}_1(\hat{\boldsymbol{b}}) + \lambda \mathcal{L}_2(\hat{\boldsymbol{b}}), \quad \text{s.t. } \hat{\boldsymbol{b}} \in \{0,1\}^{2CQ}, \ ||\boldsymbol{b} - \hat{\boldsymbol{b}}||_2^2 - k \leq 0. \tag{5}$$

Problem (5) is denoted as TA-LBF (targeted attack with limited bit-flips). Note that TA-LBF is an binary integer programming (BIP) problem, whose optimization is challenging. We will introduce an effective and efficient method to solve it in the following section.

3.3 An Effective Optimization Method for TA-LBF

To solve the challenging BIP problem (5), we adopt the generic solver for integer programming, dubbed ℓ_p-Box ADMM (Wu & Ghanem, 2018). It proposed to replace the binary constraint equivalently by the intersection of two continuous constraints, as follows

$$\hat{\boldsymbol{b}} \in \{0,1\}^{2CQ} \Leftrightarrow \hat{\boldsymbol{b}} \in \mathcal{S}_b \cap \mathcal{S}_p, \tag{6}$$

where $\mathcal{S}_b = [0,1]^{2CQ}$ indicates the box constraint, and $\mathcal{S}_p = \{\hat{\boldsymbol{b}} : ||\hat{\boldsymbol{b}} - \frac{\mathbf{1}}{2}||_2^2 = \frac{2CQ}{4}\}$ denotes the ℓ_2-sphere constraint. Utilizing (6), Problem (5) is equivalently reformulated as

$$\min_{\hat{\boldsymbol{b}}, \boldsymbol{u}_1 \in \mathcal{S}_b, \boldsymbol{u}_2 \in \mathcal{S}_p, u_3 \in \mathbb{R}^+} \quad \mathcal{L}_1(\hat{\boldsymbol{b}}) + \lambda \mathcal{L}_2(\hat{\boldsymbol{b}}), \quad \text{s.t. } \hat{\boldsymbol{b}} = \boldsymbol{u}_1, \hat{\boldsymbol{b}} = \boldsymbol{u}_2, ||\boldsymbol{b} - \hat{\boldsymbol{b}}||_2^2 - k + u_3 = 0, \tag{7}$$

where two extra variables $\boldsymbol{u}_1$ and $\boldsymbol{u}_2$ are introduced to split the constraints $w.r.t.$ $\hat{\boldsymbol{b}}$. Besides, the non-negative slack variable $u_3 \in \mathbb{R}^+$ is used to transform $||\boldsymbol{b} - \hat{\boldsymbol{b}}||_2^2 - k \leq 0$ in (5) into $||\boldsymbol{b} - \hat{\boldsymbol{b}}||_2^2 - k + u_3 = 0$. The above constrained optimization problem can be efficiently solved by the alternating direction method of multipliers (ADMM) (Boyd et al., 2011).

Following the standard procedure of ADMM, we firstly present the augmented Lagrangian function of the above problem, as follows:

$$\begin{aligned} L(\hat{\boldsymbol{b}}, \boldsymbol{u}_1, \boldsymbol{u}_2, u_3, \boldsymbol{z}_1, \boldsymbol{z}_2, z_3) = &\mathcal{L}_1(\hat{\boldsymbol{b}}) + \lambda \mathcal{L}_2(\hat{\boldsymbol{b}}) + \boldsymbol{z}_1^\top(\hat{\boldsymbol{b}} - \boldsymbol{u}_1) + \boldsymbol{z}_2^\top(\hat{\boldsymbol{b}} - \boldsymbol{u}_2) + z_3(||\boldsymbol{b} - \hat{\boldsymbol{b}}||_2^2 - k + u_3) \\ &+ c_1(\boldsymbol{u}_1) + c_2(\boldsymbol{u}_2) + c_3(u_3) + \frac{\rho_1}{2}||\hat{\boldsymbol{b}} - \boldsymbol{u}_1||_2^2 + \frac{\rho_2}{2}||\hat{\boldsymbol{b}} - \boldsymbol{u}_2||_2^2 + \frac{\rho_3}{2}(||\boldsymbol{b} - \hat{\boldsymbol{b}}||_2^2 - k + u_3)^2, \end{aligned} \tag{8}$$

where $\boldsymbol{z}_1, \boldsymbol{z}_2 \in \mathbb{R}^{2CQ}$ and $z_3 \in \mathbb{R}$ are dual variables, and ρ_1, ρ_2, $\rho_3 > 0$ are penalty factors, which will be specified later. $c_1(\boldsymbol{u}_1) = \mathbb{I}_{\{\boldsymbol{u}_1 \in \mathcal{S}_b\}}$, $c_2(\boldsymbol{u}_2) = \mathbb{I}_{\{\boldsymbol{u}_2 \in \mathcal{S}_p\}}$, and $c_3(u_3) = \mathbb{I}_{\{u_3 \in \mathbb{R}^+\}}$ capture

the constraints $\mathcal{S}_b, \mathcal{S}_p$ and $\mathbb{R}^+$, respectively. The indicator function $\mathbb{I}_{\{a\}} = 0$ if a is true; otherwise, $\mathbb{I}_{\{a\}} = +\infty$. Based on the augmented Lagrangian function, the primary and dual variables are updated iteratively, with r indicating the iteration index.

Given $(\hat{\boldsymbol{b}}^r, \boldsymbol{z}_1^r, \boldsymbol{z}_2^r, z_3^r)$, update $(\boldsymbol{u}_1^{r+1}, \boldsymbol{u}_2^{r+1}, u_3^{r+1})$. Given $(\hat{\boldsymbol{b}}^r, \boldsymbol{z}_1^r, \boldsymbol{z}_2^r, z_3^r)$, $(\boldsymbol{u}_1, \boldsymbol{u}_2, u_3)$ are independent, and they can be optimized in parallel, as follows

$$\begin{cases} \boldsymbol{u}_1^{r+1} = \underset{\boldsymbol{u}_1 \in \mathcal{S}_b}{\arg\min} \ (\boldsymbol{z}_1^r)^\top (\hat{\boldsymbol{b}}^r - \boldsymbol{u}_1) + \frac{\rho_1}{2} ||\hat{\boldsymbol{b}}^r - \boldsymbol{u}_1||_2^2 = \mathcal{P}_{\mathcal{S}_b}(\hat{\boldsymbol{b}}^r + \frac{\boldsymbol{z}_1^r}{\rho_1}), \\ \boldsymbol{u}_2^{r+1} = \underset{\boldsymbol{u}_2 \in \mathcal{S}_p}{\arg\min} \ (\boldsymbol{z}_2^r)^\top (\hat{\boldsymbol{b}}^r - \boldsymbol{u}_2) + \frac{\rho_2}{2} ||\hat{\boldsymbol{b}}^r - \boldsymbol{u}_2||_2^2 = \mathcal{P}_{\mathcal{S}_p}(\hat{\boldsymbol{b}}^r + \frac{\boldsymbol{z}_2^r}{\rho_2}), \\ u_3^{r+1} = \underset{u_3 \in \mathbb{R}^+}{\arg\min} \ z_3^r(||\boldsymbol{b} - \hat{\boldsymbol{b}}^r||_2^2 - k + u_3) + \frac{\rho_3}{2}(||\boldsymbol{b} - \hat{\boldsymbol{b}}^r||_2^2 - k + u_3)^2 \\ \qquad = \mathcal{P}_{\mathbb{R}^+}(-||\boldsymbol{b} - \hat{\boldsymbol{b}}^r||_2^2 + k - \frac{z_3^r}{\rho_3}), \end{cases} \tag{9}$$

where $\mathcal{P}_{\mathcal{S}_b}(\boldsymbol{a}) = \min((\mathbf{1}, \max(\mathbf{0}, \boldsymbol{a}))$ with $\boldsymbol{a} \in \mathbb{R}^n$ is the projection onto the box constraint $\mathcal{S}_b$; $\mathcal{P}_{\mathcal{S}_p}(\boldsymbol{a}) = \frac{\sqrt{n}}{2} \frac{\bar{\boldsymbol{a}}}{||\boldsymbol{a}||} + \frac{\mathbf{1}}{2}$ with $\bar{\boldsymbol{a}} = \boldsymbol{a} - \frac{\mathbf{1}}{2}$ indicates the projection onto the ℓ_2-sphere constraint $\mathcal{S}_p$ (Wu & Ghanem, 2018); $\mathcal{P}_{\mathbb{R}^+}(a) = \max(0, a)$ with $a \in \mathbb{R}$ indicates the projection onto $\mathbb{R}^+$.

Given $(\boldsymbol{u}_1^{r+1}, \boldsymbol{u}_2^{r+1}, u_3^{r+1}, \boldsymbol{z}_1^r, \boldsymbol{z}_2^r, z_3^r)$, update $\hat{\boldsymbol{b}}^{r+1}$. Although there is no closed-form solution to $\hat{\boldsymbol{b}}^{r+1}$, it can be easily updated by the gradient descent method, as both $\mathcal{L}_1(\hat{\boldsymbol{b}})$ and $\mathcal{L}_2(\hat{\boldsymbol{b}})$ are differentiable *w.r.t.* $\hat{\boldsymbol{b}}$, as follows

$$\hat{\boldsymbol{b}}^{r+1} \leftarrow \hat{\boldsymbol{b}}^r - \eta \cdot \frac{\partial L(\hat{\boldsymbol{b}}, \boldsymbol{u}_1^{r+1}, \boldsymbol{u}_2^{r+1}, u_3^{r+1}, \boldsymbol{z}_1^r, \boldsymbol{z}_2^r, z_3^r)}{\partial \hat{\boldsymbol{b}}} \Big|_{\hat{\boldsymbol{b}} = \hat{\boldsymbol{b}}^r}, \tag{10}$$

where $\eta > 0$ denotes the step size. Note that we can run multiple steps of gradient descent in the above update. Both the number of steps and η will be specified in later experiments. Besides, due to the space limit, the detailed derivation of $\partial L / \partial \hat{\boldsymbol{b}}$ will be presented in **Appendix A**.

Given $(\hat{\boldsymbol{b}}^{r+1}, \boldsymbol{u}_1^{r+1}, \boldsymbol{u}_2^{r+1}, u_3^{r+1})$, update $(\boldsymbol{z}_1^{r+1}, \boldsymbol{z}_2^{r+1}, z_3^{r+1})$. The dual variables are updated by the gradient ascent method, as follows

$$\begin{cases} \boldsymbol{z}_1^{r+1} = \boldsymbol{z}_1^r + \rho_1(\hat{\boldsymbol{b}}^{r+1} - \boldsymbol{u}_1^{r+1}), \\ \boldsymbol{z}_2^{r+1} = \boldsymbol{z}_2^r + \rho_2(\hat{\boldsymbol{b}}^{r+1} - \boldsymbol{u}_2^{r+1}), \\ z_3^{r+1} = z_3^r + \rho_3(||\boldsymbol{b} - \hat{\boldsymbol{b}}^{r+1}||_2^2 - k + u_3^{r+1}). \end{cases} \tag{11}$$

Remarks. 1) Note that since $(\boldsymbol{u}_1^{r+1}, \boldsymbol{u}_2^{r+1}, u_3^{r+1})$ are updated in parallel, their updates belong to the same block. Thus, the above algorithm is a two-block ADMM algorithm. We provide the algorithm outline in **Appendix B**. **2)** Except for the update of $\hat{\boldsymbol{b}}^{r+1}$, all other updates are very simple and efficient. The computational cost of the whole algorithm will be analyzed in **Appendix C**. **3)** Due to the inexact solution to $\hat{\boldsymbol{b}}^{r+1}$ using gradient descent, the theoretical convergence of the whole ADMM algorithm cannot be guaranteed. However, as demonstrated in many previous works (Gol'shtein & Tret'yakov, 1979; Eckstein & Bertsekas, 1992; Boyd et al., 2011), the inexact two-block ADMM often shows good practical convergence, which is also the case in our later experiments. Besides, the numerical convergence analysis is presented in **Appendix D**. **4)** The proper adjustment of (ρ_1, ρ_2, ρ_3) could accelerate the practical convergence, which will be specified later .

4 Experiments

4.1 Evaluation Setup

Settings. We compare our method (TA-LBF) with GDA (Liu et al., 2017a), FSA (Zhao et al., 2019), T-BFA (Rakin et al., 2020b), and TBT (Rakin et al., 2020a). All those methods can be adopted to misclassify a specific image into a target class. We also take the fine-tuning (FT) of the last fully-connected layer as a baseline method. We conduct experiments on CIFAR-10 (Krizhevsky et al., 2009) and ImageNet (Russakovsky et al., 2015). We randomly select 1,000 images from each dataset as the evaluation set for all methods. Specifically, for each of the 10 classes in CIFAR-10, we perform attacks on the 100 randomly selected validation images from the other 9 classes. For ImageNet, we randomly choose 50 target classes. For each target class, we perform attacks on 20 images randomly selected from the rest classes in the validation set. Besides, for all methods except GDA which does not employ auxiliary samples, we provide 128 and 512 auxiliary samples

Table 1: Results of all attack methods across different bit-widths and architectures on CIFAR-10 and ImageNet (bold: the best; underline: the second best). The mean and standard deviation of PA-ACC and $\mathrm{N_{flip}}$ are calculated by attacking the 1,000 images. Our method is denoted as **TA-LBF**.

Dataset	Method	Target Model	PA-ACC (%)	ASR (%)	$\mathrm{N_{flip}}$	Target Model	PA-ACC (%)	ASR (%)	$\mathrm{N_{flip}}$
CIFAR-10	FT	ResNet 8-bit	85.01±2.90	**100.0**	1507.51±86.54	VGG 8-bit	84.31±3.10	98.7	11298.74±830.36
	TBT		88.07±0.84	97.3	246.70±8.19		77.79±23.35	51.6	599.40±19.53
	T-BFA		87.56±2.22	98.7	9.91±2.33		**89.83±3.92**	96.7	14.53±3.74
	FSA	ACC: 92.16%	**88.38±2.28**	98.9	185.51±54.93	ACC: 93.20%	88.80±2.86	96.8	253.92±122.06
	GDA		86.73±3.50	99.8	26.83±12.50		85.51±2.88	**100.0**	21.54±6.79
	TA-LBF		88.20±2.64	**100.0**	**5.57±1.58**		86.06±3.17	**100.0**	**7.40±2.72**
	FT	ResNet 4-bit	84.37±2.94	**100.0**	392.48±47.26	VGG 4-bit	83.31±3.76	94.5	2270.52±324.69
	TBT		87.79±1.86	96.0	118.20±15.03		83.90±2.63	62.4	266.40±18.70
	T-BFA		86.46±2.80	97.9	8.80±2.01		**88.74±4.52**	96.2	11.23±2.36
	FSA	ACC: 91.90%	87.73±2.36	98.4	76.83±25.27	ACC: 92.61%	87.58±3.06	97.5	75.03±29.75
	GDA		86.25±3.59	99.8	14.08±7.94		85.08±2.82	**100.0**	10.31±3.77
	TA-LBF		**87.82±2.60**	**100.0**	**5.25±1.09**		85.91±3.29	**100.0**	**6.26±2.37**
ImageNet	FT	ResNet 8-bit	59.33±0.93	**100.0**	277424.29±12136.34	VGG 8-bit	62.08±2.33	**100.0**	1729685.22±137539.54
	TBT		69.18±0.03	99.9	577.40±19.42		72.99±0.02	99.2	4115.26±191.25
	T-BFA		68.71±0.36	79.3	24.57±20.03		73.09±0.12	84.5	363.78±153.28
	FSA	ACC: 69.50%	69.27±0.15	99.7	441.21±119.45	ACC: 73.31%	73.28±0.03	**100.0**	1030.03±260.30
	GDA		69.26±0.22	**100.0**	18.54±6.14		**73.29±0.02**	**100.0**	197.05±49.85
	TA-LBF		**69.41±0.08**	**100.0**	**7.37±2.18**		73.28±0.03	**100.0**	**69.89±18.42**
	FT	ResNet 4-bit	15.65±4.52	**100.0**	135854.50±21399.94	VGG 4-bit	17.76±1.71	**100.0**	1900751.70±37329.44
	TBT		66.36±0.07	99.8	271.24±15.98		71.18±0.03	**100.0**	3231.00±345.68
	T-BFA		65.86±0.42	80.4	24.79±19.02		71.49±0.15	84.3	350.33±158.57
	FSA	ACC: 66.77%	66.44±0.21	99.9	157.53±33.66	ACC: 71.76%	71.69±0.09	**100.0**	441.32±111.26
	GDA		66.54±0.22	**100.0**	11.45±3.82		**71.73±0.03**	**100.0**	107.18±28.70
	TA-LBF		**66.69±0.07**	**100.0**	**7.96±2.50**		**71.73±0.03**	**100.0**	**69.72±18.84**

on CIFAR-10 and ImageNet, respectively. Following the setting in (Rakin et al., 2020a;b), we adopt the quantized ResNet (He et al., 2016) and VGG (Simonyan & Zisserman, 2015) as the target models. For our TA-LBF, the trade-off parameter λ and the constraint parameter k affect the attack stealthiness and the attack success rate. We adopt a strategy for jointly searching λ and k, which is specified in **Appendix E.3**. More descriptions of our settings are provided in **Appendix E**.

Evaluation Metrics. We adopt three metrics to evaluate the attack performance, *i.e.,* the post attack accuracy (PA-ACC), the attack success rate (ASR), and the number of bit-flips ($\mathrm{N_{flip}}$). PA-ACC denotes the post attack accuracy on the validation set except for the specific attacked sample and the auxiliary samples. ASR is defined as the ratio of attacked samples that are successfully attacked into the target class among all 1,000 attacked samples. $\mathrm{N_{flip}}$ is the number of bit-flips required for an attack. A better attack performance corresponds to a higher PA-ACC and ASR, while a lower $\mathrm{N_{flip}}$. Besides, we also show the accuracy of the original model, denoted as ACC.

4.2 Main Results

Results on CIFAR-10. The results of all methods on CIFAR-10 are shown in Table 1. Our method achieves a 100% ASR with the fewest $\mathrm{N_{flip}}$ for all the bit-widths and architectures. FT modifies the maximum number of bits among all methods since there is no limitation of parameter modifications. Due to the absence of the training data, the PA-ACC of FT is also poor. These results indicate that fine-tuning the trained DNN as an attack method is infeasible. Although T-BFA flips the second-fewest bits under three cases, it fails to achieve a higher ASR than GDA and FSA. In terms of PA-ACC, TA-LBF is comparable to other methods. Note that the PA-ACC of TA-LBF significantly outperforms that of GDA, which is the most competitive *w.r.t.* ASR and $\mathrm{N_{flip}}$ among all the baseline methods. The PA-ACC of GDA is relatively poor, because it does not employ auxiliary samples. Achieving the highest ASR, the lowest $\mathrm{N_{flip}}$, and the comparable PA-ACC demonstrates that our optimization-based method is more superior than other heuristic methods (TBT, T-BFA and GDA).

Results on ImageNet. The results on ImageNet are shown in Table 1. It can be observed that GDA shows very competitive performance compared to other methods. However, our method obtains the highest PA-ACC, the fewest bit-flips (less than 8), and a 100% ASR in attacking ResNet. For VGG, our method also achieves a 100% ASR with the fewest $\mathrm{N_{flip}}$ for both bit-widths. The $\mathrm{N_{flip}}$ results of our method are mainly attributed to the cardinality constraint on the number of bit-flips. Moreover, for our method, the average PA-ACC degradation over four cases on ImageNet is only 0.06%, which demonstrates the stealthiness of our attack. When comparing the results of ResNet and VGG, an interesting observation is that all methods require significantly more bit-flips for VGG. One reason is that VGG is much wider than ResNet. Similar to the claim in (He et al., 2020), increasing the network width contributes to the robustness against the bit-flip based attack.

Table 2: Results of all attack methods against the models with defense on CIFAR-10 and ImageNet (bold: the best; underline: the second best). The mean and standard deviation of PA-ACC and $\mathrm{N_{flip}}$ are calculated by attacking the 1,000 images. Our method is denoted as **TA-LBF**. $\Delta\mathrm{N_{flip}}$ denotes the increased $\mathrm{N_{flip}}$ compared to the corresponding result in Table 1.

Defense	Dataset	Method	ACC (%)	PA-ACC (%)	ASR (%)	$\mathrm{N_{flip}}$	$\Delta\mathrm{N_{flip}}$
Piece-wise Clustering	CIFAR-10	FT	91.01	84.06$\pm$3.56	99.5	1893.55$\pm$68.98	386.04
		TBT		<u>87.05$\pm$1.69</u>	52.3	254.20$\pm$10.22	**7.50**
		T-BFA		85.82$\pm$1.89	98.6	<u>45.51$\pm$9.47</u>	35.60
		FSA		86.61$\pm$2.51	98.6	246.11$\pm$75.36	60.60
		GDA		84.12$\pm$4.77	**100.0**	52.76$\pm$16.29	25.93
		TA-LBF		**87.30$\pm$2.74**	**100.0**	**18.93$\pm$7.11**	<u>13.36</u>
	ImageNet	FT	63.62	43.44$\pm$2.07	92.2	762267.56$\pm$52179.46	484843.27
		TBT		63.07$\pm$0.04	81.8	1184.14$\pm$30.30	606.74
		T-BFA		62.82$\pm$0.27	90.1	273.56$\pm$191.29	248.99
		FSA		<u>63.26$\pm$0.21</u>	99.5	729.94$\pm$491.83	288.73
		GDA		63.14$\pm$0.48	**100.0**	<u>107.59$\pm$31.15</u>	<u>89.05</u>
		TA-LBF		**63.52$\pm$0.14**	**100.0**	**51.11$\pm$4.33**	**43.74**
Larger Model Capacity	CIFAR-10	FT	94.29	86.46$\pm$2.84	**100.0**	2753.43$\pm$188.27	1245.92
		TBT		89.72$\pm$2.99	89.5	366.90$\pm$12.09	120.20
		T-BFA		**91.16$\pm$1.42**	98.7	<u>17.91$\pm$4.64</u>	<u>8.00</u>
		FSA		90.70$\pm$2.37	98.5	271.27$\pm$65.18	85.76
		GDA		89.83$\pm$3.02	**100.0**	48.96$\pm$21.03	22.13
		TA-LBF		<u>90.96$\pm$2.63</u>	**100.0**	**8.79$\pm$2.44**	**3.22**
	ImageNet	FT	71.35	63.51$\pm$1.29	**100.0**	507456.61$\pm$34517.04	230032.32
		TBT		71.12$\pm$0.04	99.9	1138.34$\pm$44.23	560.94
		T-BFA		70.84$\pm$0.30	88.9	40.23$\pm$27.29	15.66
		FSA		**71.30$\pm$0.04**	**100.0**	449.70$\pm$106.42	8.49
		GDA		**71.30$\pm$0.05**	**100.0**	<u>20.01$\pm$6.04</u>	<u>1.47</u>
		TA-LBF		**71.30$\pm$0.04**	**100.0**	**8.48$\pm$2.52**	**1.11**

4.3 Resistance to Defense Methods

Resistance to Piece-wise Clustering. He et al. (2020) proposed a novel training technique, called piece-wise clustering, to enhance the network robustness against the bit-flip based attack. Such a training technique introduces an additional weight penalty to the inference loss, which has the effect of eliminating close-to-zero weights (He et al., 2020). We test the resistance of all attack methods to the piece-wise clustering. We conduct experiments with the 8-bit quantized ResNet on CIFAR-10 and ImageNet. Following the ideal configuration in (He et al., 2020), the clustering coefficient, which is a hyper-parameter of piece-wise clustering, is set to 0.001 in our evaluation. For our method, the initial k is set to 50 on ImageNet and the rest settings are the same as those in Section 4.1. Besides the three metrics in Section 4.1, we also present the number of increased $\mathrm{N_{flip}}$ compared to the model without defense ($i.e.$, results in Table 1), denoted as $\Delta\mathrm{N_{flip}}$.

The results of the resistance to the piece-wise clustering of all attack methods are shown in Table 2. It shows that the model trained with piece-wise clustering can improve the number of required bit-flips for all attack methods. However, our method still achieves a 100% ASR with the least number of bit-flips on both two datasets. Although TBT achieves a smaller $\Delta\mathrm{N_{flip}}$ than ours on CIFAR-10, its ASR is only 52.3%, which also verifies the defense effectiveness of the piece-wise clustering. Compared with other methods, TA-LBF achieves the fewest $\Delta\mathrm{N_{flip}}$ on ImageNet and the best PA-ACC on both datasets. These results demonstrate the superiority of our method over other methods when attacking models trained with piece-wise clustering.

Resistance to Larger Model Capacity. Previous studies (He et al., 2020; Rakin et al., 2020b) observed that increasing the network capacity can improve the robustness against the bit-flip based attack. Accordingly, we evaluate all attack methods against the models with a larger capacity using the 8-bit quantized ResNet on both datasets. Similar to the strategy in (He et al., 2020), we increase the model capacity by varying the network width ($i.e.$, $2\times$ width in our experiments). All settings of our method are the same as those used in Section 4.1.

The results are presented in Table 2. We observe that all methods require more bit-flips to attack the model with the $2\times$ width. To some extent, it demonstrates that the wider network with the same architecture is more robust against the bit-flip based attack. However, our method still achieves a 100% ASR with the fewest $\mathrm{N_{flip}}$ and $\Delta\mathrm{N_{flip}}$. Moreover, when comparing the two defense methods, we find that piece-wise clustering performs better than the model with a larger capacity in terms of $\Delta\mathrm{N_{flip}}$. However, piece-wise clustering training also causes the accuracy decrease of the original model ($e.g.$, from 92.16% to 91.01% on CIFAR-10). We provide more results in attacking models with defense under different settings in **Appendix F**.

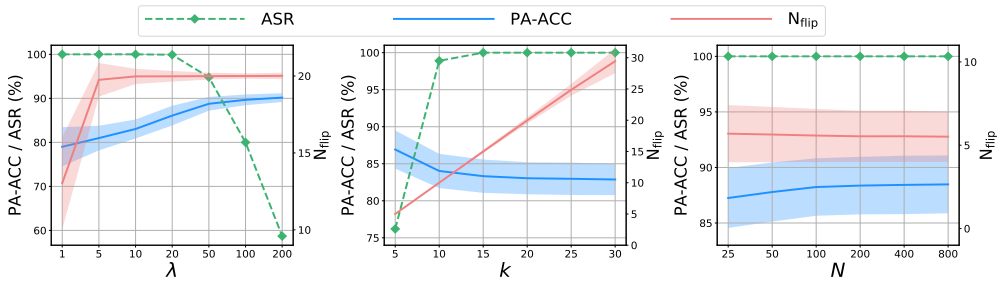

Figure 2: Results of TA-LBF with different parameters λ, k, and the number of auxiliary samples N on CIFAR-10. Regions in shadow indicate the standard deviation of attacking the 1,000 images.

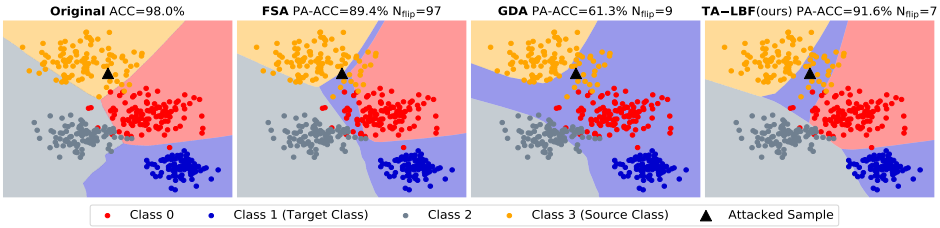

Figure 3: Visualization of decision boundaries of the original model and the post attack models. The attacked sample from Class 3 is misclassified into the Class 1 by FSA, GDA, and our method.

4.4 Ablation Study

We perform ablation studies on parameters λ and k, and the number of auxiliary samples N. We use the 8-bit quantized ResNet on CIFAR-10 as the representative for analysis. We discuss the attack performance of TA-LBF under different values of λ while k is fixed at 20, and under different values of k while λ is fixed at 10. To analyze the effect of N, we configure N from 25 to 800 and keep other settings the same as those in Section 4.1. The results are presented in Fig. 2. We observe that our method achieves a 100% ASR when λ is less than 20. As expected, the PA-ACC increases while the ASR decreases along with the increase of λ. The plot of parameter k presents that k can exactly limit the number of bit-flips, while other attack methods do not involve such constraint. This advantage is critical since it allows the attacker to identify limited bits to perform an attack when the budget is fixed. As shown in the figure, the number of auxiliary samples less than 200 has a marked positive impact on the PA-ACC. It's intuitive that more auxiliary samples can lead to a better PA-ACC. The observation also indicates that TA-LBF still works well without too many auxiliary samples.

4.5 Visualization of Decision Boundary

To further compare FSA and GDA with our method, we visualize the decision boundaries of the original and the post attack models in Fig. 3. We adopt a four-layer Multi-Layer Perceptron trained with the simulated 2-D Blob dataset from 4 classes. The original decision boundary indicates that the original model classifies all data points almost perfectly. The attacked sample is classified into Class 3 by all methods. Visually, GDA modifies the decision boundary drastically, especially for Class 0. However, our method modifies the decision boundary mainly around the attacked sample. Althoug FSA is comparable to ours visually in Fig. 3, it flips $10\times$ bits than GDA and TA-LBF. In terms of the numerical results, TA-LBF achieves the best PA-ACC and the fewest $\mathrm{N}_{\mathrm{flip}}$. This finding verifies that our method can achieve a successful attack even only tweaking the original classifier.

5 Conclusion

In this work, we have presented a novel attack paradigm that the weights of a deployed DNN can be slightly changed via bit flipping in the memory, to give a target prediction for a specific sample, while the predictions on other samples are not significantly influenced. Since the weights are stored as binary bits in the memory, we formulate this attack as a binary integer programming (BIP) problem, which can be effectively and efficiently solved by a continuous algorithm. Since the critical bits are determined through optimization, the proposed method can achieve the attack goals by flipping a few bits, and it shows very good performance under different experimental settings. To further illustrate the differences between our TA-LBF with existing methods, we also provide a detailed comparison in **Appendix G**.

REFERENCES

Michel Agoyan, Jean-Max Dutertre, Amir-Pasha Mirbaha, David Naccache, Anne-Lise Ribotta, and Assia Tria. How to flip a bit? In *IOLTS*, pp. 235–239, 2010.

Naveed Akhtar and Ajmal Mian. Threat of adversarial attacks on deep learning in computer vision: A survey. *IEEE Access*, 6:14410–14430, 2018.

Stephen Boyd, Neal Parikh, and Eric Chu. *Distributed optimization and statistical learning via the alternating direction method of multipliers*. Now Publishers Inc, 2011.

Jakub Breier, Xiaolu Hou, Dirmanto Jap, Lei Ma, Shivam Bhasin, and Yang Liu. Practical fault attack on deep neural networks. In *CCS*, pp. 2204–2206, 2018.

Yair Carmon, Aditi Raghunathan, Ludwig Schmidt, John C. Duchi, and Percy Liang. Unlabeled data improves adversarial robustness. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pp. 11190–11201, 2019.

Yinpeng Dong, Hang Su, Baoyuan Wu, Zhifeng Li, Wei Liu, Tong Zhang, and Jun Zhu. Efficient decision-based black-box adversarial attacks on face recognition. In *CVPR*, pp. 7714–7722, 2019.

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

Jonathan Eckstein and Dimitri P Bertsekas. On the douglas—rachford splitting method and the proximal point algorithm for maximal monotone operators. *Mathematical Programming*, 55(1-3):293–318, 1992.

Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust physical-world attacks on deep learning visual classification. In *CVPR*, pp. 1625–1634, 2018.

E Gi Gol'shtein and NV Tret'yakov. Modified lagrangians in convex programming and their generalizations. In *Point-to-Set Maps and Mathematical Programming*, pp. 86–97. Springer, 1979.

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

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

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pp. 770–778, 2016.

Zhezhi He, Adnan Siraj Rakin, Jingtao Li, Chaitali Chakrabarti, and Deliang Fan. Defending and harnessing the bit-flip based adversarial weight attack. In *CVPR*, pp. 14095–14103, 2020.

Sanghyun Hong, Pietro Frigo, Yiğitcan Kaya, Cristiano Giuffrida, and Tudor Dumitraș. Terminal brain damage: Exposing the graceless degradation in deep neural networks under hardware fault attacks. In *USENIX Security Symposium*, pp. 497–514, 2019.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *CVPR*, pp. 2704–2713, 2018.

Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping bits in memory without accessing them: An experimental study of dram disturbance errors. *ACM SIGARCH Computer Architecture News*, 42(3):361–372, 2014.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. *Technical report*, 2009.

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

Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Fine-pruning: Defending against backdooring attacks on deep neural networks. In *RAID*, pp. 273–294, 2018a.

Yannan Liu, Lingxiao Wei, Bo Luo, and Qiang Xu. Fault injection attack on deep neural network. In *ICCAD*, pp. 131–138, 2017a.

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

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

Yuntao Liu, Yang Xie, and Ankur Srivastava. Neural trojans. In *ICCD*, pp. 45–48, 2017b.

Yuntao Liu, Ankit Mondal, Abhishek Chakraborty, Michael Zuzak, Nina Jacobsen, Daniel Xing, and Ankur Srivastava. A survey on neural trojans. In *ISQED*, 2020b.

Szymon Migacz. 8-bit inference with tensorrt. In *GPU technology conference*, 2017.

Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. In *CVPR*, pp. 1765–1773, 2017.

Jonathan Pan. Blackbox trojanising of deep learning models: Using non-intrusive network structure and binary alterations. *arXiv preprint arXiv:2008.00408*, 2020.

Adnan Siraj Rakin, Zhezhi He, and Deliang Fan. Bit-flip attack: Crushing neural network with progressive bit search. In *ICCV*, pp. 1211–1220, 2019.

Adnan Siraj Rakin, Zhezhi He, and Deliang Fan. Tbt: Targeted neural network attack with bit trojan. In *CVPR*, pp. 13198–13207, 2020a.

Adnan Siraj Rakin, Zhezhi He, Jingtao Li, Fan Yao, Chaitali Chakrabarti, and Deliang Fan. T-bfa: Targeted bit-flip adversarial weight attack. *arXiv preprint arXiv:2007.12336*, 2020b.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

Aniruddha Saha, Akshayvarun Subramanya, and Hamed Pirsiavash. Hidden trigger backdoor attacks. In *AAAI*, pp. 11957–11965, 2020.

Bodo Selmke, Stefan Brummer, Johann Heyszl, and Georg Sigl. Precise laser fault injections into 90 nm and 45 nm sram-cells. In *CARDIS*, pp. 193–205. Springer, 2015.

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

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

Brandon Tran, Jerry Li, and Aleksander Madry. Spectral signatures in backdoor attacks. In *NeurIPS*, pp. 8000–8010, 2018.

Victor Van Der Veen, Yanick Fratantonio, Martina Lindorfer, Daniel Gruss, Clémentine Maurice, Giovanni Vigna, Herbert Bos, Kaveh Razavi, and Cristiano Giuffrida. Drammer: Deterministic rowhammer attacks on mobile platforms. In *CCS*, pp. 1675–1689, 2016.

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 *IEEE S&P*, pp. 707–723, 2019.

Baoyuan Wu and Bernard Ghanem. ℓ_p-box admm: A versatile framework for integer programming. *IEEE transactions on pattern analysis and machine intelligence*, 41(7):1695–1708, 2018.

Chulin Xie, Keli Huang, Pin-Yu Chen, and Bo Li. Dba: Distributed backdoor attacks against federated learning. In *ICLR*, 2019.

Cihang Xie, Jianyu Wang, Zhishuai Zhang, Zhou Ren, and Alan Yuille. Mitigating adversarial effects through randomization. *ICLR*, 2018.

Weilin Xu, David Evans, and Yanjun Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. *NDSS*, 2017.

Fan Yao, Adnan Siraj Rakin, and Deliang Fan. Deephammer: Depleting the intelligence of deep neural networks through targeted chain of bit flips. In *USENIX Security Symposium*, pp. 1463–1480, 2020.

Pu Zhao, Siyue Wang, Cheng Gongye, Yanzhi Wang, Yunsi Fei, and Xue Lin. Fault sneaking attack: A stealthy framework for misleading deep neural networks. In *ACM DAC*, pp. 1–6. IEEE, 2019.

A UPDATE $\hat{\boldsymbol{b}}$ BY GRADIENT DESCENT

In this section, we derive the gradient of L *w.r.t.* $\hat{\boldsymbol{b}}$, which is adopted to update $\hat{\boldsymbol{b}}^{r+1}$ by gradient descent (see Eq. (10)). The derivation consists of the following parts.

Derivation of $\partial\mathcal{L}_1(\hat{\boldsymbol{b}})/\partial\hat{\boldsymbol{b}}$. For clarity, here we firstly repeat some definitions,

$$\mathcal{L}_1(\hat{\mathbf{B}}_s, \hat{\mathbf{B}}_t) = \max\big(m - p(\boldsymbol{x}; \boldsymbol{\Theta}, \hat{\mathbf{B}}_t) + \delta, 0\big) + \max\big(p(\boldsymbol{x}; \boldsymbol{\Theta}, \hat{\mathbf{B}}_s) - m + \delta, 0\big), \tag{12}$$

$$p(\boldsymbol{x}; \boldsymbol{\Theta}, \hat{\mathbf{B}}_i) = [h(\hat{\mathbf{B}}_{i,1}); h(\hat{\mathbf{B}}_{i,2}); ...; h(\hat{\mathbf{B}}_{i,C})]^\top g(\boldsymbol{x}; \boldsymbol{\Theta}), \tag{13}$$

$$h(\boldsymbol{v}) = (-2^{Q-1} \cdot v_Q + \sum_{i=1}^{Q-1} 2^{i-1} \cdot v_i) \cdot \Delta^l. \tag{14}$$

Then, we obtain that

$$\frac{\partial p(\boldsymbol{x}; \boldsymbol{\Theta}, \hat{\mathbf{B}}_i)}{\partial \hat{\mathbf{B}}_i} = [g_1(\boldsymbol{x}; \boldsymbol{\Theta}) \cdot (\frac{\nabla h(\hat{\mathbf{B}}_{i,1})}{\partial \hat{\mathbf{B}}_{i,1}})^\top; ...; g_C(\boldsymbol{x}; \boldsymbol{\Theta}) \cdot (\frac{\nabla h(\hat{\mathbf{B}}_{i,C})}{\nabla \hat{\mathbf{B}}_{i,C}})^\top], \tag{15}$$

where $\frac{\nabla h(\boldsymbol{v})}{\nabla \boldsymbol{v}} = [2^0; 2^1, \ldots, 2^{Q-2}; -2^{Q-1}] \cdot \Delta^l$ is a constant, and here l indicates the last layer; $g_j(\boldsymbol{x}; \boldsymbol{\Theta})$ denotes the j-th entry of the vector $g(\boldsymbol{x}; \boldsymbol{\Theta})$. Utilizing (15), we have

$$\begin{aligned} \frac{\partial \mathcal{L}_1(\hat{\mathbf{B}}_s, \hat{\mathbf{B}}_t)}{\partial \hat{\mathbf{B}}_s} &= \begin{cases} \frac{\partial p(\boldsymbol{x}; \boldsymbol{\Theta}, \hat{\mathbf{B}}_s)}{\partial \hat{\mathbf{B}}_s}, & \text{if } p(\boldsymbol{x}; \boldsymbol{\Theta}, \mathbf{B}_s) > m - \delta \\ \mathbf{0}, & \text{otherwise} \end{cases}, \\ \frac{\partial \mathcal{L}_1(\hat{\mathbf{B}}_s, \hat{\mathbf{B}}_t)}{\partial \hat{\mathbf{B}}_t} &= \begin{cases} -\frac{\partial p(\boldsymbol{x}; \boldsymbol{\Theta}, \hat{\mathbf{B}}_t)}{\partial \hat{\mathbf{B}}_t}, & \text{if } p(\boldsymbol{x}; \boldsymbol{\Theta}, \mathbf{B}_t) < m + \delta \\ \mathbf{0}, & \text{otherwise} \end{cases}. \end{aligned} \tag{16}$$

Thus, we obtain that

$$\frac{\partial \mathcal{L}_1(\hat{\boldsymbol{b}})}{\partial \hat{\boldsymbol{b}}} = \big[\text{Reshape}\big(\frac{\partial \mathcal{L}_1(\hat{\mathbf{B}}_s, \hat{\mathbf{B}}_t)}{\partial \hat{\mathbf{B}}_s}\big); \text{Reshape}\big(\frac{\partial \mathcal{L}_1(\hat{\mathbf{B}}_s, \hat{\mathbf{B}}_t)}{\partial \hat{\mathbf{B}}_t}\big)\big], \tag{17}$$

where Reshape$(\cdot)$ elongates a matrix to a vector along the column.

Derivation of $\partial\mathcal{L}_2(\hat{\boldsymbol{b}})/\partial\hat{\boldsymbol{b}}$. For clarity, here we firstly repeat the following definition

$$\mathcal{L}_2(\hat{\mathbf{B}}_s, \hat{\mathbf{B}}_t) = \sum_{i=1}^{N} \ell\big(f(\boldsymbol{x}_i; \boldsymbol{\Theta}, \mathbf{B}_{\{1,\ldots,K\}\setminus\{s,t\}}, \hat{\mathbf{B}}_s, \hat{\mathbf{B}}_t), y_i\big), \tag{18}$$

where $f_j(\boldsymbol{x}_i; \boldsymbol{\Theta}, \mathbf{B}_{\{1,\ldots,K\}\setminus\{s,t\}}, \hat{\mathbf{B}}_s, \hat{\mathbf{B}}_t) = \text{Softmax}(p(\boldsymbol{x}_i; \boldsymbol{\Theta}, \hat{\mathbf{B}}_j))$ or $\text{Softmax}(p(\boldsymbol{x}_i; \boldsymbol{\Theta}, \mathbf{B}_j))$ indicates the posterior probability of $\boldsymbol{x}_i$ *w.r.t.* class j, and we simply denote $f(\boldsymbol{x}_i) \in [0, 1]^K$ as the

posterior probability vector of $\boldsymbol{x}_i$. $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$ denotes the auxiliary sample set. $\ell(\cdot,\cdot)$ is specified as the cross entropy loss. Then, we have

$$\frac{\partial \mathcal{L}_2(\hat{\mathbf{B}}_s, \hat{\mathbf{B}}_t)}{\partial \hat{\mathbf{B}}_s} = \sum_{i=1}^{N} \Big[\big(\mathbb{I}(y_i = s) - f_s(\boldsymbol{x}_i; \boldsymbol{\Theta}, \mathbf{B}_{\{1,\ldots,K\}\setminus\{s,t\}}, \hat{\mathbf{B}}_s, \hat{\mathbf{B}}_t)\big) \cdot \frac{\partial p(\boldsymbol{x}_i; \boldsymbol{\Theta}, \hat{\mathbf{B}}_s)}{\partial \hat{\mathbf{B}}_s} \Big], \quad (19)$$

$$\frac{\partial \mathcal{L}_2(\hat{\mathbf{B}}_s, \hat{\mathbf{B}}_t)}{\partial \hat{\mathbf{B}}_t} = \sum_{i=1}^{N} \Big[\big(\mathbb{I}(y_i = t) - f_t(\boldsymbol{x}_i; \boldsymbol{\Theta}, \mathbf{B}_{\{1,\ldots,K\}\setminus\{s,t\}}, \hat{\mathbf{B}}_s, \hat{\mathbf{B}}_t)\big) \cdot \frac{\partial p(\boldsymbol{x}_i; \boldsymbol{\Theta}, \hat{\mathbf{B}}_t)}{\partial \hat{\mathbf{B}}_t} \Big], \quad (20)$$

where $\mathbb{I}(a) = 1$ of a is true, otherwise $\mathbb{I}(a) = 0$. Thus, we obtain

$$\frac{\partial \mathcal{L}_2(\hat{\boldsymbol{b}})}{\partial \hat{\boldsymbol{b}}} = \Big[\text{Reshape}\big(\frac{\partial \mathcal{L}_2(\hat{\mathbf{B}}_s, \hat{\mathbf{B}}_t)}{\partial \hat{\mathbf{B}}_s}\big); \text{Reshape}\big(\frac{\partial \mathcal{L}_2(\hat{\mathbf{B}}_s, \hat{\mathbf{B}}_t)}{\partial \hat{\mathbf{B}}_t}\big) \Big]. \quad (21)$$

Derivation of $\partial L(\hat{\boldsymbol{b}})/\partial \hat{\boldsymbol{b}}$. According to Eq. (8), and utilizing Eqs. (17) and (21), we obtain

$$\frac{\partial L(\hat{\boldsymbol{b}})}{\partial \hat{\boldsymbol{b}}} = \frac{\partial \mathcal{L}_1(\hat{\boldsymbol{b}})}{\partial \hat{\boldsymbol{b}}} + \frac{\partial \mathcal{L}_2(\hat{\boldsymbol{b}})}{\partial \hat{\boldsymbol{b}}} + \boldsymbol{z}_1 + \boldsymbol{z}_2 + \rho_1(\hat{\boldsymbol{b}} - \boldsymbol{u}_1) + \rho_2(\hat{\boldsymbol{b}} - \boldsymbol{u}_2) + 2(\hat{\boldsymbol{b}} - \boldsymbol{b}) \cdot \big[\boldsymbol{z}_3 + \rho_3 ||\hat{\boldsymbol{b}} - \boldsymbol{b}||_2^2 - k + u_3\big]. \quad (22)$$

B Algorithm Outline

Algorithm 1 Continuous optimization for the BIP problem (5).

Input: The original quantized DNN model f with weights $\boldsymbol{\Theta}, \mathbf{B}$, attacked sample $\boldsymbol{x}$ with ground-truth label s, target class t, auxiliary sample set $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$, hyper-parameters λ, k, and δ.

Output: $\hat{\boldsymbol{b}}$.

1: Initial $\boldsymbol{u}_1^0$, $\boldsymbol{u}_2^0$, u_3^0, $\boldsymbol{z}_1^0$, $\boldsymbol{z}_2^0$, z_3^0, $\hat{\boldsymbol{b}}^0$ and let $r \leftarrow 0$;
2: **while** not converged **do**
3: Update $\boldsymbol{u}_1^{r+1}$, $\boldsymbol{u}_2^{r+1}$ and u_3^{r+1} as Eq. (9);
4: Update $\hat{\boldsymbol{b}}^{r+1}$ as Eq. (10);
5: Update $\boldsymbol{z}_1^{r+1}$, $\boldsymbol{z}_2^{r+1}$ and z_3^{r+1} as Eq. (11);
6: $r \leftarrow r + 1$.
7: **end while**

C Complexity Analysis

The computational complexity of the proposed algorithm (*i.e.*, Algorithm 1) consists of two parts, the forward and backward pass. In terms of the forward pass, since $\boldsymbol{\Theta}, \mathbf{B}_{\{1,\ldots,K\}\setminus\{s,t\}}$ are fixed during the optimization, their involved terms, including $g(\boldsymbol{x}; \boldsymbol{\Theta})$ and $p(\boldsymbol{x}; \boldsymbol{\Theta}, \mathbf{B}_i)|_{i \neq s,t}$, are calculated only one time. The main cost from $\hat{\mathbf{B}}_s$ and $\hat{\mathbf{B}}_t$ is $O(2(N+1)C^2Q)$ per iteration, as there are $N+1$ samples. In terms of the backward pass, the main cost is from the update of $\hat{\boldsymbol{b}}^{r+1}$, which is $O(2(N+1)CQ)$ per iteration in the gradient descent. Since all other updates are very simple, their costs are omitted here. Thus, the overall computational cost is $O\big(T_{outer}[2(N+1)CQ \cdot (C + T_{inner})]\big)$, with T_{outer} being the iteration of the overall algorithm and T_{inner} indicating the number of gradient steps in updating $\hat{\boldsymbol{b}}^{r+1}$. As shown in Section D, the proposed method TA-LBF always converges very fast in our experiments, thus T_{outer} is not very large. As demonstrated in Section E.3, T_{inner} is set to 5 in our experiments. In short, the proposed method can be optimized very efficiently.

D Numerical Convergence Analysis

We present the numerical convergence of TA-LBF in Fig. 4. Note that $||\hat{\boldsymbol{b}} - \boldsymbol{u}_1||_2^2$ and $||\hat{\boldsymbol{b}} - \boldsymbol{u}_2||_2^2$ characterize the degree of satisfaction of the box and ℓ_2-sphere constraint, respectively. For the two

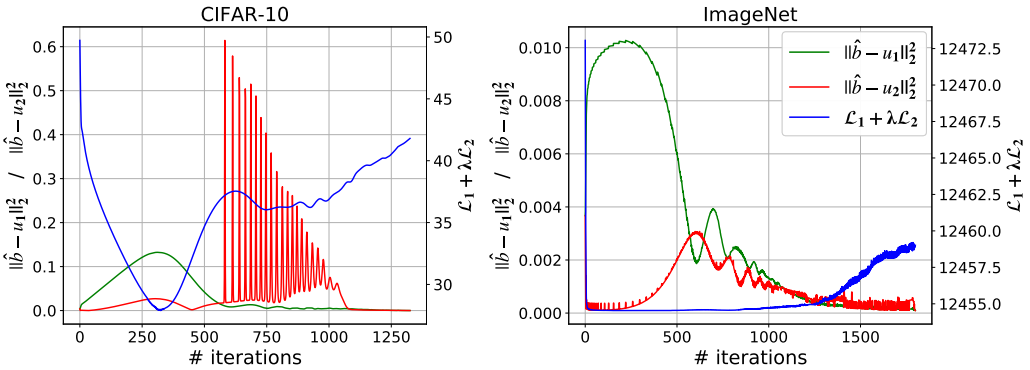

Figure 4: Numerical convergence analysis of TA-LBF *w.r.t.* the attacked sample on CIFAR-10 and ImageNet, respectively. We present the values of $||\hat{\boldsymbol{b}}-\boldsymbol{u}_1||_2^2$, $||\hat{\boldsymbol{b}}-\boldsymbol{u}_2||_2^2$ and $\mathcal{L}_1+\lambda\mathcal{L}_2$ at different iterations in attacking 8-bit quantized ResNet. Note that λ in the left figure is 100 and λ in the right figure is 10^4.

examples of CIFAR-10 and ImageNet, the values of both indicators first increase, then drop, and finally close to 0. Another interesting observation is that $\mathcal{L}_1+\lambda\mathcal{L}_2$ first decreases evidently and then increases slightly. Such findings illustrate the optimization process of TA-LBF. In the early iterations, modifying the model parameters tends to achieve the two goals mentioned in Section 3.1; in the late iterations, $\hat{\boldsymbol{b}}$ is encouraged to satisfy the box and l_2-sphere constraint. We also observe that both examples stop when meeting $||\hat{\boldsymbol{b}}-\boldsymbol{u}_1||_2^2 \leq 10^{-4}$ and $||\hat{\boldsymbol{b}}-\boldsymbol{u}_2||_2^2 \leq 10^{-4}$ and do not exceed the maximum number of iterations ($i.e.$, 2000). The numerical results demonstrate the fast convergence of our method in practice.

E Evaluation Setup

E.1 Baseline Methods

Since GDA (Liu et al., 2017a) and FSA (Zhao et al., 2019) are originally designed for attacking the full-precision network, we adapt these two methods to attack the quantized network by applying quantization-aware training (Jacob et al., 2018). We adopt the ℓ_0-norm for FSA (Liu et al., 2017a) and modification compression for GDA (Zhao et al., 2019) to reduce the number of the modified parameters. Among three types of T-BFA (Rakin et al., 2020b), we compare to the most comparable method: the 1-to-1 stealthy attack scheme. The purpose of this attack scheme is to misclassify samples of a single source class into the target class while maintaining the prediction accuracy of other samples. Besides, we take the fine-tuning (FT) of the last fully-connected layer as a basic attack and present its results. We perform attack once for each selected image except TBT (Rakin et al., 2020a) and totally 1,000 attacks on each dataset. The attack objective of TBT is that the attacked DNN model misclassifies all inputs with a trigger to a certain target class. Due to such objective, the number of attacks for TBT is equal to the number of target classes ($i.e.$, 10 attacks on CIFAR-10 and 50 attacks on ImageNet).

E.2 Target Models

According to the setting in (Rakin et al., 2020a;b), we adopt two popular network architectures: ResNet (He et al., 2016) and VGG (Simonyan & Zisserman, 2015) for evaluation. On CIFAR-10, we perform experiments on ResNet-20 and VGG-16. On ImageNet, we use the pre-trained ResNet-18[1] and VGG-16[2] network. We quantize all networks to the 4-bit and 8-bit quantization level using the layer-wise uniform weight quantization scheme, which is similar to the one involved in the Tensor-RT solution (Migacz, 2017).

[1]Downloaded from `https://download.pytorch.org/models/resnet18-5c106cde.pth`
[2]Downloaded from `https://download.pytorch.org/models/vgg16_bn-6c64b313.pth`

Table 3: Results of all attack methods against models trained with piece-wise clustering on CIFAR-10 (bold: the best; underline: the second best). We adopt different clustering coefficients, including 0.0005, 0.005, and 0.01. The mean and standard deviation of PA-ACC and $\mathrm{N_{flip}}$ are calculated by attacking the 1,000 images. Our method is denoted as **TA-LBF**. $\Delta\mathrm{N_{flip}}$ denotes the increased $\mathrm{N_{flip}}$ compared to the corresponding result in Table 1.

Clustering Coefficient	Method	ACC (%)	PA-ACC (%)	ASR (%)	$\mathrm{N_{flip}}$	$\Delta\mathrm{N_{flip}}$
0.0005	FT	91.42	84.28±3.49	**100.0**	1868.26±72.48	360.75
	TBT		**87.97±1.75**	66.1	250.30±10.97	**3.60**
	T-BFA		86.20±1.96	98.5	30.95±6.50	21.04
	FSA		87.17±2.44	98.5	222.70±56.52	37.19
	GDA		85.28±4.16	**100.0**	41.33±12.84	14.50
	TA-LBF		87.92±2.54	**100.0**	**13.47±5.34**	7.90
0.005	FT	88.03	81.08±3.61	97.9	1774.69±51.47	267.18
	TBT		82.96±2.18	12.7	246.80±16.06	**0.10**
	T-BFA		80.80±2.64	98.1	61.72±12.17	51.81
	FSA		83.10±2.75	98.4	231.66±89.21	46.15
	GDA		79.23±6.25	99.9	64.87±22.78	38.04
	TA-LBF		**83.63±3.47**	**100.0**	**25.52±11.59**	19.95
0.01	FT	85.65	78.73±3.54	98.3	1748.54±46.19	241.03
	TBT		79.86±2.04	10.1	236.50±10.93	**-10.20**
	T-BFA		76.67±3.41	98.1	55.49±11.77	45.58
	FSA		80.45±3.14	98.0	220.28±101.01	34.77
	GDA		75.33±7.83	99.7	59.17±23.63	32.34
	TA-LBF		**80.51±4.39**	**100.0**	**24.60±13.03**	19.03

E.3 Parameter Settings of TA-LBF

For each attack, we adopt a strategy for jointly searching λ and k. Specifically, for an initially given k, we search λ from a relatively large initial value and divide it by 2 if the attack does not succeed. The maximum search times of λ for a fixed k is set to 8. If it exceeds the maximum search times, we double k and search λ from the relatively large initial value. The maximum search times of k is set to 4. On CIFAR-10, the initial k and λ are set to 5 and 100. On ImageNet, λ is initialized as 10^4; k is initialized as 5 and 50 for ResNet and VGG, respectively. On CIFAR-10, the δ in $\mathcal{L}_1$ is set to 10. On ImageNet, the δ is set to 3 and increased to 10 if the attack fails. $\boldsymbol{u}_1$ and $\boldsymbol{u}_2$ are initialized as $\boldsymbol{b}$ and u_3 is initialized as 0. $\boldsymbol{z}_1$ and $\boldsymbol{z}_2$ are initialized as $\mathbf{0}$ and z_3 is initialized as 0. $\hat{\boldsymbol{b}}$ is initialized as $\boldsymbol{b}$. During each iteration, the number of gradient steps for updating $\hat{\boldsymbol{b}}$ is 5 and the step size is set to 0.01 on both datasets. Hyper-parameters (ρ_1, ρ_2, ρ_3) (see Eq. (11)) are initialized as (10^{-4}, 10^{-4}, 10^{-5}) on both datasets, and increase by $\rho_i \leftarrow \rho_i \times 1.01$, $i = 1, 2, 3$ after each iteration. The maximum values of (ρ_1, ρ_2, ρ_3) are set to (50, 50, 5) on both datasets. Besides the maximum number of iterations ($i.e.$, 2000), we also set another stopping criterion, $i.e.$, $||\hat{\boldsymbol{b}} - \boldsymbol{u}_1||_2^2 \leq 10^{-4}$ and $||\hat{\boldsymbol{b}} - \boldsymbol{u}_2||_2^2 \leq 10^{-4}$.

F More Results on Resistance to Defense Methods

F.1 Resistance to Piece-wise Clustering

We conduct experiments using the 8-bit quantized ResNet on CIFAR-10 with different clustering coefficients. We set the maximum search times of k to 5 for clustering coefficient 0.005 and 0.01 and keep the rest settings the same as those in Section 4.1. The results are presented in Table 3. As shown in the table, all values of $\mathrm{N_{flip}}$ are larger than attacking models without defense for all methods, which is similar to Table 2. Our method achieves a 100% ASR with the fewest $\mathrm{N_{flip}}$ under the three clustering coefficients. Although TBT obtains a smaller $\Delta\mathrm{N_{flip}}$ than our method, it fails to achieve a satisfactory ASR. For example, TBT achieves only a 10.1% ASR when the clustering coefficient is set to 0.01. We observe that for all clustering coefficients, piece-wise clustering reduces the original accuracy. Such a phenomenon is more significant as the clustering coefficient increases.

Table 4: Results of all attack methods against models with a larger capacity on CIFAR-10. We adopt $3\times$ and $4\times$ width networks. The mean and standard deviation of PA-ACC and $\mathrm{N_{flip}}$ are calculated by attacking the 1,000 images. $\Delta\mathrm{N_{flip}}$ denotes the increased $\mathrm{N_{flip}}$ compared to the corresponding result in Table 1.

Model Width	Method	ACC (%)	PA-ACC (%)	ASR (%)	$\mathrm{N_{flip}}$	$\Delta\mathrm{N_{flip}}$
$3\times$	FT	94.90	86.96$\pm$2.79	**100.0**	4002.52$\pm$281.24	2495.01
	TBT		90.67$\pm$5.23	74.1	504.70$\pm$20.44	258.00
	T-BFA		**92.18$\pm$1.14**	98.9	30.50$\pm$7.52	20.59
	FSA		91.40$\pm$2.38	99.0	342.20$\pm$79.44	156.69
	GDA		90.79$\pm$2.91	**100.0**	67.53$\pm$27.45	40.70
	TA-LBF		91.42$\pm$2.81	**100.0**	**12.29$\pm$4.18**	**6.72**
$4\times$	FT	95.02	86.94$\pm$2.78	**100.0**	4527.68$\pm$369.35	3020.17
	TBT		85.39$\pm$5.08	90.1	625.50$\pm$32.38	378.80
	T-BFA		**92.49$\pm$1.22**	99.4	19.14$\pm$5.04	9.23
	FSA		91.60$\pm$2.42	98.7	338.93$\pm$100.12	153.42
	GDA		90.76$\pm$3.00	**100.0**	66.92$\pm$40.32	40.09
	TA-LBF		90.94$\pm$3.11	**100.0**	**8.37$\pm$2.80**	**2.80**

The results also show that there is no guarantee that if the clustering coefficient is larger (e.g. 0.01), the model is more robust, which is consistent with the finding in (He et al., 2020).

F.2 Resistance to Larger Model Capacity

Besides the results of networks with a $2\times$ width shown in Section 4.3, we also evaluate all methods against models with a $3\times$ and $4\times$ width. All settings are the same as those used in Section 4.1. The results are provided in Table 4. Among all attack methods, our method is least affected by increasing the network width. Especially for the network with a $4\times$ width, our $\Delta\mathrm{N_{flip}}$ is only 2.80. The results demonstrate the superiority of the formulated BIP problem and optimization. Moreover, compared with piece-wise clustering, having a larger model capacity can improve the original accuracy, but increases the model size and the computation complexity.

G Discussions

G.1 Comparing Backdoor, Adversarial, and Weight Attack

An attacker can achieve malicious purposes utilizing backdoor, adversarial, and weight attacks. In this section, we emphasize the differences among them.

Backdoor attack happens in the training stage and requires that the attacker can tamper the training data even the training process (Liu et al., 2020b; Li et al., 2020). Through poisoning some training samples with a trigger, the attacker can control the behavior of the attacked DNN in the inference stage. For example, images with reflections are misclassified into a target class, while benign images are classified normally (Liu et al., 2020a). However, such an attack paradigm causes the accuracy degradation on benign samples, which makes it detectable for users. Besides, these methods also require to modify samples in the inference stage, which is sometimes impossible for the attacker. Many defense methods against backdoor attack have been proposed, such as the preprocessing-based defense (Liu et al., 2017b), the model reconstruction-based defense (Liu et al., 2018a), and the trigger synthesis-based defense (Wang et al., 2019).

Adversarial attack modifies samples in the inference stage by adding small perturbations that remain imperceptible to the human vision system (Akhtar & Mian, 2018). Since adversarial attack only modifies inputs while keeping the model unchanged, it has no effect on the benign samples. Similarly, recent studies have demonstrated many defense methods against adversarial attack, including the preprocessing-based defense (Xie et al., 2018), the detection-based defense (Xu et al., 2017), and the adversarial learning-based defense (Carmon et al., 2019).

Weight attack modifies model parameters in the deployment stage, which is the studied paradigm in this work. Weight attack generally aims at misleading the DNN model on the selected sample(s), while having a minor effect on other samples (Zhao et al., 2019; Rakin et al., 2020b). Many studies (Yao et al., 2020; Breier et al., 2018; Pan, 2020) have demonstrated that the DNN parameters can be modified in the bit-level in memory using fault injection techniques (Agoyan et al., 2010; Kim et al., 2014; Selmke et al., 2015) in practice. Note that the defense methods against weight attack have been not well studied. Although some defense methods (He et al., 2020) were proposed, they cannot achieve satisfactory performance. For example, our method can still achieve a 100% attack success rate against two proposed defense methods. Our work would encourage further investigation on the security of the model parameters from both attack and defense sides.

G.2 Comparing TA-LBF with Other Weight Attacks

We compare our TA-LBF with other weight attack methods, including TBT (Rakin et al., 2020a), T-BFA (Rakin et al., 2020b), GDA (Liu et al., 2017a), and FSA (Zhao et al., 2019) in this section. TBT tampers both the test sample and the model parameters. Specifically, it first locates critical bits and generates a trigger, and then flips these bits to classify all inputs embedded with the trigger to a target class. However, the malicious samples are easily detected by human inspection or many detection methods (Tran et al., 2018; Du et al., 2020). We do not modify the samples to perform TA-LBF, which makes the attack more stealthy. Rakin et al. (2020b) proposed T-BFA which misclassifies all samples (N-to-1 version) or samples from a source class (1-to-1 version) into a target class. Our method aims at misclassifying a specific sample, which meets the attacker's requirement in some scenarios. For example, the attacker wants to manipulate the behavior of a face recognition engine on a specific input. Since it affects multiple samples, T-BFA maybe not stealthy enough in attacking real-world applications. GDA (Liu et al., 2017a) and FSA (Zhao et al., 2019) modify model parameters at the weight-level rather than bit-level. They are designed for misclassifying multiple samples from arbitrary classes, which makes it infeasible for them to only modify the parameters connected to the source and target class. They modify more parameters than our method as shown in the experiments, it might be due to the reason discussed above. Besides, TBT, T-BFA, and GDA determine the critical weights to modify using heuristic strategies, while our TA-LBF adopts optimization-based methods. Although FSA applies ADMM for solving the optimization problem, it has no explicit constraint to control the number of modified parameters, which makes it intends to modify more parameters than GDA and our TA-LBF.