Journal of Signal Processing Systems (2023) 95:1425–1437
https://doi.org/10.1007/s11265-023-01895-3

Stealthy Energy Consumption-oriented Attacks on Training Stage in Deep Learning

Wencheng Chen[1] · Hongyu Li[1]

Received: 18 May 2023 / Revised: 29 August 2023 / Accepted: 21 September 2023 / Published online: 11 October 2023

Abstract
Deep Learning as a Service (DLaaS) is rapidly developing recently to enable applications including self-driving, face recognition, and natural language processing for small enterprises. However, DLaaS can also introduce enormous computing power consumption at the service ends. Existing works focus on the optimization of the training process such as using low-cost chips or optimizing the training settings for better energy efficiency. In this paper, we revisit this issue from an adversary perspective which attempts to maliciously make victims waste more training efforts without being noticed. In particular, we propose a novel attack targeting enlarging the training costs stealthily via poisoning the training data. By adopting the Projected Gradient Descent (PGD) method to generate poisoned samples, we show that attackers can significantly increase the training costs by as much as 88% in both the white-box scenario and the black-box scenario with a very tiny influence on the model's accuracy.

Keywords Deep learning · Energy attack · Energy consumption · Carbon footprint

1 Introduction

Recently, applications enabled by deep Learning (DL) is widely used in various applications, such as face recognition [1], knowledge graphs [2], recommendation systems [3], and self-driving cars [4]. Various DNN architectures [5–9] have been proposed to improve the performance of DNN models. However, when these DNN models become more capable, their model architectures also become more complex with significantly more parameters. Thus, it is more and more difficult for a local computer or server to train these high-performance but sophisticated models [10]. A more promising approach recently is to rely on Deep Learning as a Service (DLaaS) provided by existing cloud computing vendors to train or deploy these DNN models such as Amazon Sagemaker [11], Google Cloud ML Engine [12], and Microsoft Azure ML Studio [13].

✉ Wencheng Chen
wenchengchen@bupt.edu.cn

Hongyu Li
l543306408@bupt.edu.cn

[1] School of Information and Communication Engineering, Beijing University of Posts and Telecommunications, Beijing 100876, China

While DL users or small enterprises can enjoy the convenience of using the DLaaS service to train large-scale models, the amount of the carbon footprint during the model training phase is also rapidly increasing. For instance, as pointed out in [14], training DL models can introduce significant carbon emissions which is even comparable with civil aviation. Particularly, the carbon emissions of training a state-of-the-art Transformer model [15] with neural architecture search (NAS) equal the carbon emissions of five cars in their lifetimes [16]. Recent works [16] have pointed out the environmental influence of training large-scale models if all the research only focuses on some metrics and does not consider their training efficiency.

Existing approaches [17] focus on optimizing the training process to save energy costs and decrease carbon emissions under benign circumstances. Some other approaches are proposed to design more efficient chips such as Tensor Processing Units (TPUs) [18]. All these approaches assume that the training process is trustworthy. However, the training process via DLaaS can be vulnerable if we re-visit this process from an adversarial perspective. For instance, recent research has proven that the models have a risk of being attacked during the training or the inference phase [19, 20]. By poisoning the training dataset, the adversary can launch a backdoor attack [21–23]. In the inference phase, adversarial examples (AEs) can deceive the model [24–26]. Considering the extensive and unreliable sources of the training dataset, we believe that it is possible to poison the training dataset while targeting energy costs. The adversary can poison the training dataset to launch an energy attack that aims to maliciously increase energy consumption in the training phase. Note that many small enterprises use DLaaS but this kind of energy attack may significantly increase the training cost to disturb their model training. On the other hand, such kind of malicious energy consumption will lead to an unnecessary carbon footprint increase. Moreover, such kind of energy attack can be very stealthy if the adversary performs such attacks but does not decrease the final trained model's accuracy. Thus, it is necessary to explore and understand the possibility of raising energy consumption by malicious attacks from an adversary's viewpoint.

In this paper, we review the security of DNN models' training from a novel but adversarial aspect targeting energy consumption. We show that there are security issues with energy consumption in the training phase. We propose a practical methodology to launch the energy-oriented attack and we can increase the training time to maliciously increase energy consumption. If the adversary can get the training dataset and replace part of the benign samples with poisoned samples before the training process, it will cost the victim more money and time to train a model. The algorithm to generate poisoned samples is based on the Projected Gradient Descent (PGD) algorithm [27]. Meanwhile, these poisoned samples are close to the replaced benign samples. It is hard for the victims to notice the attack. The experimentation is conducted on the CIFAR-10 dataset [28], and we experiment with different architectures of the attacked model, including ResNet-50, ResNet-18 [29], and VGG [30]. The victim has various methods to end the training. Therefore, we select two commonly used methods to evaluate our attack methodology respectively in two cases. The experiment results indicate this kind of attack exists in both the white-box scenario and the black-box scenario. Moreover, the accuracy decrease meant of the attacked model will not exceed 1%.

This paper has the following contributions: (1) We propose an attack methodology to generate the poisoned samples and launch a successful energy attack; (2) We show that it is feasible to increase the training time by poisoning the training dataset using crafted poisoned samples. Meanwhile, the accuracy decrease meant the attacked model will not exceed 1%, and the poisoned samples are close to the replaced benign samples. This study warns that the training dataset is susceptible to energy attacks by an adversary.

The remainder of the paper is structured as follows. Section 2 lists the background of our research. Section 3 presents an overview of our attack methodology. We show the specific attack results of two cases respectively in Section 5 and Section 6. Section 7 discuss future directions. Section 8 concludes our work.

2 Research Background

2.1 Deep Learning as a Service

With the increasing demand for using large-scale DNN models, cloud computing vendors provide the DLaaS, e.g., Amazon Sagemaker, Cloud ML Engine, and Microsoft Azure ML Studio. DLaaS can be used to train or deploy a model in the server as shown in Fig. 1. It is more convenient to use DLaaS than training and deploying models in local machines because DLaaS can shorten the training time and improve the inference speed.

There are different forms of DLaaS according to flexibility and convenience. Which form to choose depends on the users' requirements. First, the DLaaS can be provided in a server, which has good flexibility but little convenience [31]. If the users are experts in DL, they only need a bare server equipped with powerful processors and GPGPUs (General Purpose computing on Graphics Processing Units) to accomplish their work. With this server, experienced users can take control of the training phase and deploying phase in detail. But they have to write their codes from scratch. Second, the DLaaS can be provided in a series of prepared utilities to keep the balance between flexibility and convenience. If the users are ordinary developers, they may want to ease their development processing by using utilities already written by professional developers. In this condition, the users do not need to write all the code. They can use some interfaces in the platform to solve their problems. Third, the DLaaS can be provided in the auto-learning form, which is convenient for the users but has poor flexibility. If the users have no experience in DL, they can only upload the dataset and

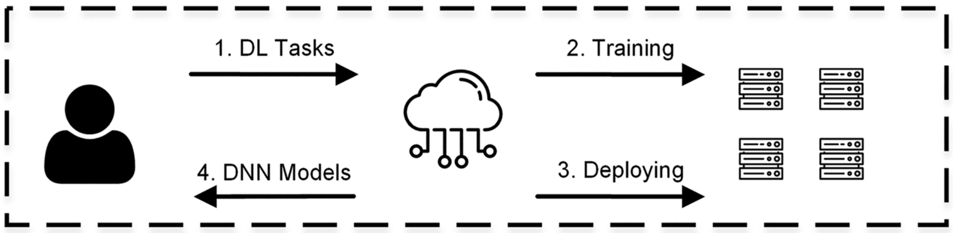

Figure 1 The processing of a DLaaS service.

choose the task without further operation. The DLaaS will automatically train the model and deploy it on the server.

2.2 Energy Efficiency

Even for a big cloud computing vendor, i.e., Google, it takes a high cost to improve the performance of the large-scale models in some tasks. For example, Google AI Brain proposes a Transformer model to translate between English and German which takes 150,000 dollars and 32,000 TPU hours but only to improve the 0.1 scores inaccuracy [14]. We cannot ignore the energy consumption during this phase. Therefore, it is necessary to investigate how to improve energy efficiency.

We can use two approaches to save energy: the software approach and the hardware approach. The first approach is to improve the training and inference algorithms, e.g., pruning [32], distilling [33], and quantization [34]. Pruning is to discard part of neural networks, which has less impact on the final output, to decrease the computing. This can be used in the inference phase to reduce energy consumption and keep accuracy in the meantime. Distilling is to use a small-scale model to fit a large-scale model. The small-scale model has fewer parameters but has similar accuracy to the large-scale model. Quantization is to change the data type of parameters from high precision to low precision or mixed precision. We can neither choose to lower the parameter precision before the training nor save the model with a low precision after the training. The second approach is to use Application Specific Integrated Circuit (ASIC) chips rather than GPGPUs to train and deploy the DNN models [35]. We use general-purpose chips, i.e., GPGPUs, to train and deploy our model in most cases, while ASIC chips are more efficient than GPGPUs [18]. These ASIC chips are good at accelerating DNN models, which can offer higher computing speed can cost less energy. Nowadays, there are many ASIC chips to accomplish training and inference tasks, including Google's TPU [18], Huawei's Davinci NPU [36], and Cambricon's DianNao NPU [37].

2.3 Energy Attacks

Present researches on the energy consumption of DNN models are mainly from the perspective of energy saving [38–40]. Because there exist security problems in the DNN models, e.g., backdoor attacks, and AEs, we consider this energy problem from the adversary's perspective. As an adversary, we aim at increasing the energy consumption in the DLaaS without influencing the accuracy of DNN models. The attack process in the DLaaS scenario shows in Fig. 2. Users train their models by applying training services provided by cloud computing vendors. During the training phase, the adversary may attack the training dataset to increase the training time. Furthermore, it will take the user more money to accomplish the training tasks. This kind of attack usually is difficult to find. The users don't have enough knowledge about the training phase. The cloud computing vendors [41] cannot inspect the training dataset due to security or privacy regulations. Therefore, the adversary can attack both the users and the cloud computing vendors without being found.

Although there are already several methods to launch energy attacks in the training and inference phases, all these methods have disadvantages. In the training phase, we can attack the training process to make it take more time to train a DNN model. The first attack method is to modify the initialization matrixes in the DNN [42]. This paper designs different initialization matrixes for Convolutional Neural Networks (CNN) [43] and fully connected networks to stop the DNN from training. But, this method is easy to find due to the dramatic fall in model accuracy. Meanwhile, it lacks feasibility in the real world because it is hard to change initialization matrixes. The second attack method is to change the order of the samples in the training dataset [44]. This paper proves that if we change the order of samples to some special orders, it will disturb the training phase. However, in most situations, there is a shuffle in the training dataset during the training phase. In the inference phase, we can generate poisoned samples to increase the inference time. The attack method is to modify a sample to

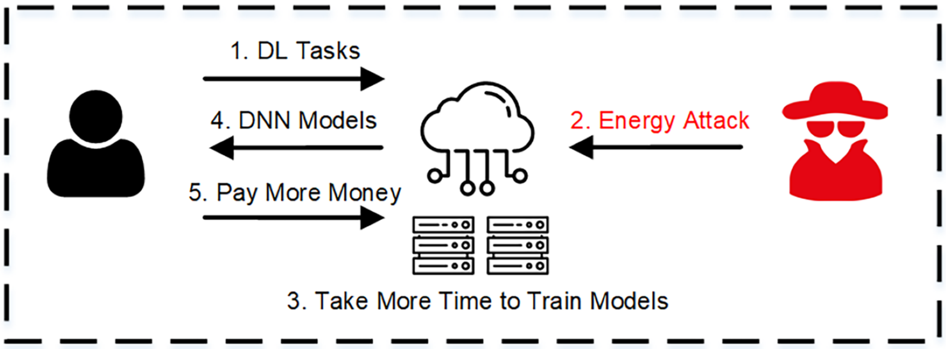

Figure 2 The processing of an adversary maliciously attacking a DLaaS service.

have output in uniform distribution for each exit in the multi-exit model [27]. The original purpose of a multi-exit model is to save energy when inference a sample. If a sample's output distribution meets the threshold in some exit of the multi-exit model, it can early exit. But, the adversary can attack the output distribution to uniform distribution to stop the sample from early exiting. However, this method can only attack the multi-exit architecture model and cannot be used for attacking different models.

Despite these methods can successfully launch the energy attack, they all have their disadvantages. We need an attack method that can adapt the different architectures of DNN models and have the feasibility to launch in the real world. Meanwhile, it is necessary to keep the attack hard to find.

3 Design and Experimentation

In this section, we first define the threat model from an adversary's viewpoint. Then we present our intuition and the proposed attack methodology. Finally, our experimentation of the attack is presented.

3.1 Threat Model

Before diving into our specific attack methodology, we first define the threat model, including adversarial goals, adversarial capability, and adversary knowledge. Being aware of the threat model helps us clearly understand the situation from an adversary's viewpoint. We define the threat model in three parts as described below.

Adversarial Goals The adversary attempts to increase the victim's cost in the training time by poisoning the benign training dataset. The poison method is to replace part of the benign samples in the benign training dataset with the poisoned samples. To keep hidden from the victim's notice, poisoned samples generated by the adversary should be as close as the replaced benign ones. The similarity between replaced benign samples and poisoned samples can be measured using l_2 norm metrics. Moreover, the accuracy of the poisoned model can not drop too much than innocent models to guarantee the attack's stealthiness.

Adversarial Capability The adversary can not attack the whole training system. During the attack process, the adversary can only get the benign training dataset and poison it with crafted poisoned samples. All other training steps are safe.

Adversary's Knowledge How much information the adversary knows depends on the attack scenario. There are two types of attack scenarios: the white-box scenario and the black-box scenario. In the former type, the adversary knows the architecture of the model that the victim plans to train. While in the latter type, the adversary cannot get the model architecture. We assume that the other information, such as training algorithms, hyper-parameters, and model parameters, are unknown to the adversary.

3.2 Attack Methodology

Our intuition is to increase the difficulty for a model to learn the necessary features for classification but not affect the model's final accuracy. Specifically, our assumption is that a sample's probability distribution in the model inference phase may connect to the sample's feature. When the model gives a high-confidence value in the sample's label class, it means this sample's feature is clear enough for the model to give an easy prediction. On the contrary, a uniform probability distribution in the prediction results represents that the sample's feature is challenging to learn. The real datasets are always formed by those high-confidence samples and low-confidence samples while the former ones can help the model training to rapidly locate important features to classify.

Therefore, our attack methodology is to generate poisoned samples whose probability distributions are close to a uniform distribution to increase the difficulty of a model to learn. With more such difficult samples mixed in the training dataset, more training efforts (e.g. epochs) will be needed for a model to achieve a target accuracy score.

The process of building a poisoned dataset can be illustrated as follows. We need to build a poisoned dataset that can maliciously increase the training efforts but will not affect the model accuracy on classifying the clean samples. This can not only lead to malicious attack results but also guarantee the stealthiness of our attack. Thus, the core idea of our attack is to modify part of the training dataset to make it malicious. In order to build such a poisoned dataset, our approach is to first use a part of this dataset to train a proxy model. This proxy model can then be used to tell the confidence score of the rest dataset by inference. Then, the attack methodology consists of two parts: generating poisoned samples and poisoning the benign training dataset. The algorithm of generating poisoned samples based on the PGD algorithm [45] is inspired by [27]. We modify it as described in Eq. 1.

$$p^{t+1} = \prod_{||p||_{\infty<\epsilon}}(p^t + \alpha sgn(\nabla_p L(F(x+p), \widetilde{y}))) \tag{1}$$

Here, x is the attacked benign sample. p is the perturbation. $x + p$ is the poisoned sample. t is the current iteration times. $\prod$ is the projection operation. ϵ is the maximum perturbation range. α is the step size. F is the model. L is a loss function between the output probability distribution and the targeted distribution, and $\widetilde{y}$ is

```
Input: The surrogate model $F$, benign sample $x \in \mathbb{R}^{c\times h\times w}$, hyper-parameters including the maximum perturbation $\varepsilon$, the step size $\alpha$, the maximal iteration number $T$ and the target label probability $\widetilde{y}$
Output: The poisoned sample $x + p$
1: Randomly initialize the perturbation $p \in \mathbb{R}^{c\times h\times w}$.
2: $t \leftarrow 0$.
3: while $t < T$ do
4:     $g^t \leftarrow \nabla_p L(F(x + p^t), \widetilde{y})$            ▷ Fetch the gradient
5:     $g^t \leftarrow sgn(g^t)$                                  ▷ PGD step
6:     $p^{t+1} \leftarrow \prod_{||p||_\infty<\varepsilon}(p^t + \alpha g^t)$
7:     $t \leftarrow t + 1$
8: end while
9: return $x + p$
```

Algorithm 1 Crafting poisoned samples

the targeted distribution. In each iteration, we attempt to push $F(x+p)$ towards distribution $\widetilde{y}$. The $\prod$ insures that the l_∞ norm of p can not exceed ε. Note the initial PGD is used to generate adversarial examples which the target label is usually chosen by the attackers. Using initial version PGD can only guarantee a high confidence to predict the sample as the target label but will not affect the confidence distribution of the other labels. However, our goal is different since we do not seek to modify the predicted label to cause any misclassification. On the contrary, we want to generate a poisoned sample that can be correctly classified by the model with more similarity to other labels. According to the goal above, we modify the target of the PGD function to generate our poisoned samples. We list this poison process in the following algorithm.

An attack result is shown in Fig. 3. The image of the truck in the first row is a replaced benign sample. Its probability distribution is concentrated. The truck image in the second row is a crafted poisoned sample. It has a uniform distribution. Our algorithm of generating poisoned samples can successfully attack the probability distribution to targeted distribution and make poisoned samples close to the replaced benign samples. The l_2 distance between x and $x+p$ in this example is 2.0.

4 Experiment Configuration

Our server is equipped with an Intel i7-9700K CPU, two NVIDIA RTX 2080Ti GPUs, and a memory of 32GB, with Ubuntu 18.04, CUDA 10.0, and cuDNN 7.6.0. The framework of our code is PyTorch 1.10.

4.1 Attack Details

In our experiments, we mix the benign training dataset with the poisoned samples. We assume that the victims will use this training dataset from a third-party source and then train their own models. Figure 4 shows the overview of our method, which consists of three phases. We detail the three phases as follows.

In phase 1: the attack phase, the adversary gets the benign training dataset, where is used to train a surrogate model and generate poisoned samples targeting more energy consumption. In the white-box scenario, the architecture of the victim's model is accessible to the adversary. The adversary can train a model, which has the same architecture as the victim's model. Then the adversary attacks it with our attack methodology above. In the black-box scenario, it is impossible to know what the victim's model architecture is for the adversary.

The only solution is to train a different model and count on its transferable ability. According to whether the attacked model has seen the attacked benign samples in the training dataset, there are two methods to attack. **The first method** only takes part of benign samples to train the attacked model, and we use the rest of them to generate poisoned samples. In **the second method**, the model has seen the whole benign training dataset during the training, and we attack this model with the whole benign training dataset to generate poisoned samples. We only choose correctly classified samples to attack in either method. The detailed algorithm for generating poisoned samples is described in Eq. 1. We experiment with these two methods to explore the influence that the attacked benign samples are in the training dataset of the attacked model. Each method experiments in the white-box scenario and the black-box scenario.

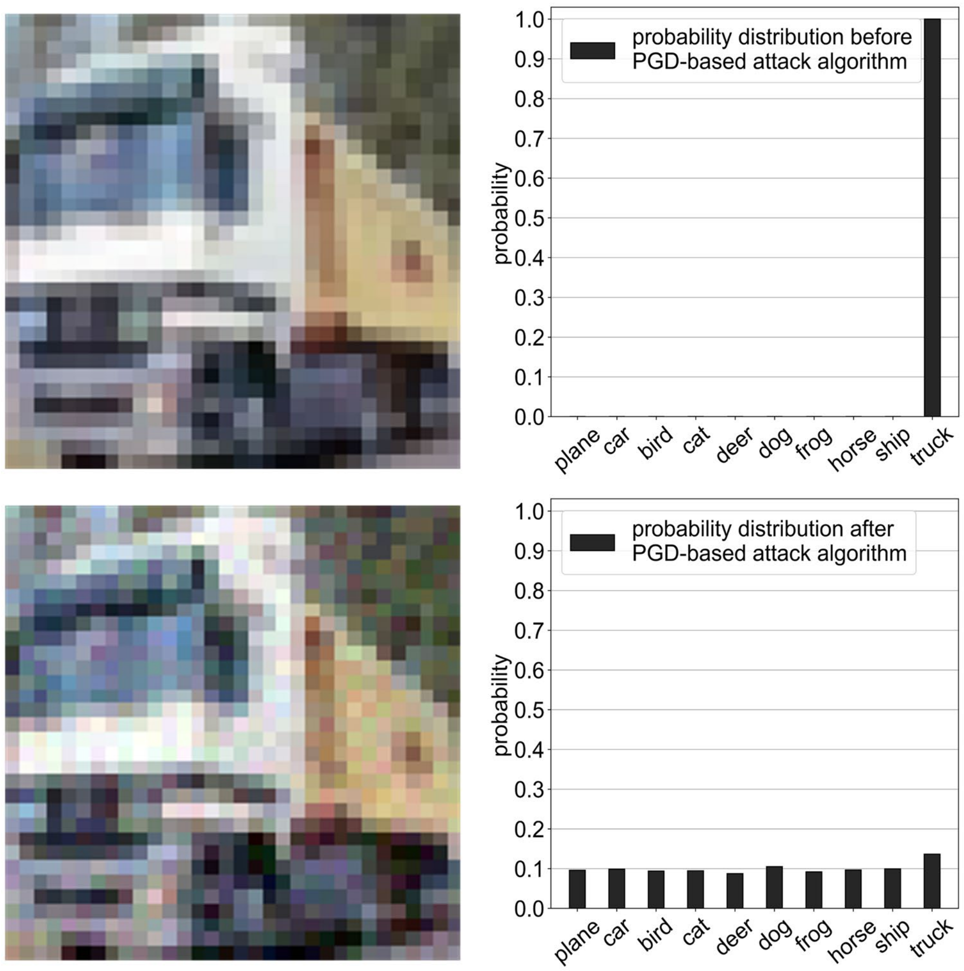

Figure 3 An example of the attack result.

Since not all the attacks are successful, we select part of the attack results, which have correct classification results and the close probability distributions in the inference phase to the targeted distribution, as our poisoned samples.

In phase 2: the poison phase, poisoned samples replace part of benign samples in the benign training dataset in a replacement ratio. The replacement ratio means the number of replaced samples in the poisoned training dataset to the number in the poisoned training dataset. The exact value of the replacement ratio relies on the adversary, and we experiment with different replacement ratios to investigate their difference. During the replacement, we replace a benign sample with the poisoned sample generated by itself. The adversary successfully attacks the benign training dataset into a poisoned training dataset.

In phase 3: the training phase, the victim uses the poisoned dataset, consisting of poisoned samples and benign samples, to train a model. Considering poisoned samples are close to the replaced benign samples, the victim can not tell the difference between them. Because these poisoned samples are hard to learn, it will take the victim more time and money to train a model. Meanwhile, the accuracy of the model will not decrease significantly. It is hard for the victim to notice the existence of the adversary. Generally, there are many means for the victim to decide when to end training the model. The adversary can not disturb the victim's decision. Thus, we choose two commonly used means: an early stopping algorithm and stopping when some metrics meet a threshold, to examine the attack results. In the early stopping algorithm, if some metrics do not improve or have little change over a given number of epochs, the training will be stopped. The minimum change to qualify as an improvement is δ, b is the baseline value for the monitored quantity, and the given number of epochs is patience. The second stopping algorithm is to stop the training when some metrics meet a threshold, and t is the threshold to end the model training. The specific experiment results are presented in the following two cases.

4.2 Experiment Setup

Dataset We adopt two popular vision datasets in our experiments, CIFAR-10 and SVHN. For CIFAR-10, it has 60,000 images of 32×32 size. These images have ten classes in

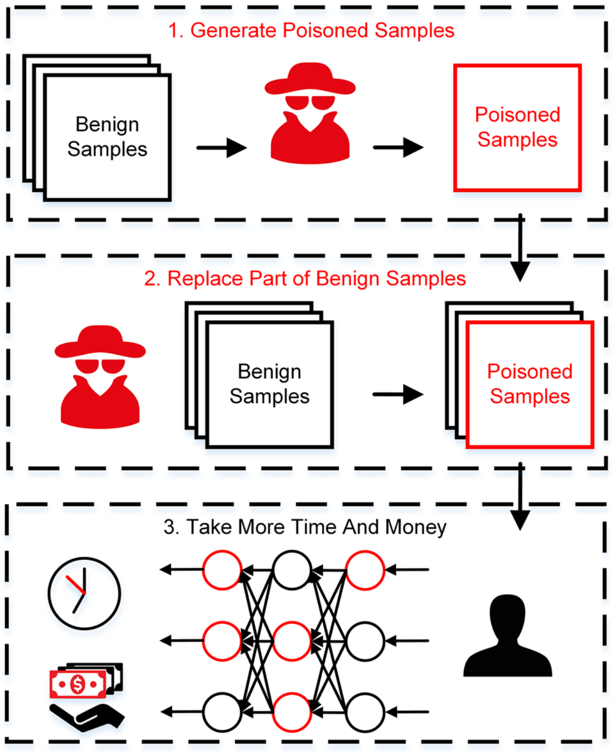
1. Generate Poisoned Samples
Benign Samples
Poisoned Samples
2. Replace Part of Benign Samples
Benign Samples
Poisoned Samples
3. Take More Time And Money

Figure 4 An overview of the experimentation. The experiment consists of three phases. In the first phase, the adversary gets the benign training dataset and generates poisoned samples on this dataset. Then the adversary replaces part of the benign samples in the benign training dataset with poison. The replacement ratio is variable. Finally, the victim uses this poisoned training dataset and chooses an algorithm to end the training. If the attack is successful, it will take the victim more time and money to train a model.

total. We divide it into the training dataset and test dataset with a 5:1 ratio. The other dataset called SVHN, which is collected for the task of identifying digits, is composed of 73,257 samples for training and 26,032 samples for validation. All samples of SVHN are 32×32 RGB images of printed digits (from 0 to 9) cropped from real-world pictures. The adversary takes advantage of benign samples to craft the poisoned samples and then replaces them with their corresponding malicious versions under different poison rates. The victim is assumed to accept the released poisoned dataset and train their models with it. The test dataset is used to measure the model's accuracy. There are two methods to generate poisoned samples as described in Section 3. In the first method, we use 80% images of the training split (e.g., 40,000 samples in CIFAR-10) to build up a surrogate model and generate poisoned samples by attacking the surrogate model with the rest 20% images. The second method uses all images of the training set to train a model, and we attack this model with the same set of images to get the poisoned samples.

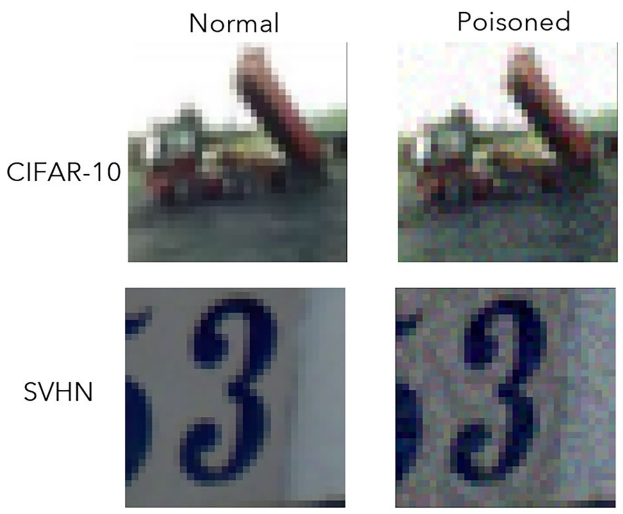

Figure 5 The examples of clean images and the corresponding poisoned images.

Hyper-Parameter Configuration Our parameters of the PGD attack algorithm are set as follows: ϵ is 0.2; α is 0.001, and the maximum iteration time is 200. Because the poisoned samples will fail to be correctly classified if the $\widetilde{y}$ are too close to a uniform distribution. In the first method, $\widetilde{y}$ is a probability distribution where the correct class corresponds to 0.15, and the others are $\frac{1-0.15}{9}$. In the second method, we set the probability of the correct class as 0.2 and the other classes' probability $\frac{1-0.2}{9}$. We select ResNet-50 as the surrogate model and the target model of the white-box scenario. For the black-box scenario, the poisoned samples are generated by attacking ResNet-50. The attacking results are evaluated on two other kinds of models. We employ SGD as the training optimizer.

4.3 Visual Effects

We list the visual results of the two datasets as shown in Fig. 5. Note that our poisoned samples are different compared with the adversarial examples that aim to mislead the model's classification. These poisoned samples can still be correctly classified by either human eyes or by the models which makes them very stealthy.

5 Case Study 1: Training with Early Stopping

In this section, we give the first case study by considering the victim applying the early stopping algorithm to end the training phase at an early stage. We present attack results on training efficiency by evaluating three models and different data poison rates.

5.1 Evaluation on Training Efficiency

In the first attack method, we successfully generated 9,000 poisoned samples out of 10,000 benign images in CIFAR-10 and 13,932 poisoned samples out of 14,652 benign images in SVHN. We use different poison rates: 2%, 10%, and 18%, to replace benign samples in the benign training dataset with poisoned samples, where poison rate means the percentage of poisoned samples in the poisoned training dataset. In the white-box scenario, ResNet-50 is trained with the poisoned dataset. In the black-box scenario, we evaluate the effectiveness of the crafted samples on two other kinds of model architecture, ResNet-18 or VGG-13 for CIFAR-10 and ResNet-34 and VGG-16 for SVHN. Table 1 displays the attack results in the white-box scenario and the black-box scenario. We compare the training epochs under different replacement ratios with the training epochs in the innocent dataset. The final model accuracy is listed in the accuracy column. In most cases, our attack methodology can make it take more epochs to train a model.

In the second attack method, we managed to obtain 49,500 poisoned samples out of 50,000 benign samples in CIFAR-10 and 66,735 poisoned samples out of 73,257 benign samples in SVHN. In order to compare with the first methodology, we also choose three replacement ratios: 2%, 10%, and 18%, respectively. The poison ratio is limited by the samples selected to train the proxy model. Note that we first train a local proxy model with partial training dataset (clean dataset) such that we can determine how to generate poisoned samples in the rest of the clean training dataset. Moreover, training this proxy model must use most of the training dataset to guarantee the model is capable to help generate the poisoned samples. In our setting, we use 80% clean training datasets to train this proxy model such that the maximum poison ratio is the 20% (i.e. the rest training dataset). Thus, we set 2%, 10%, and 18% as three poison ratio settings. We give the attack

Table 1 The attack results of the 1st method using early stopping. The hyper-parameters for CIFAR-10 are set as follows: ResNet-50, $\delta = 0.001$, $b = 90$, patience = 60; ResNet-18, $\delta = 0.01$, $b = 90$, patience = 40; VGG-13, $\delta = 0.001$, $b = 90$, patience = 60. For SVHN, we set $\delta = 0.001$, $b = 90$, patience = 40 for all models.

Dataset	Architecture	Poison rate	Epoch change (%)	Accuracy (%)
CIFAR-10	ResNet-50	2%	7.04	94.92
		10%	14.53	94.72
		18%	28.19	94.45
	ResNet-18	2%	5.19	94.48
		10%	−5.19	94.37
		18%	9.91	94.05
	VGG-13	2%	22.84	93.53
		10%	33.18	93.32
		18%	39.65	93.17
SVHN	ResNet-50	2%	16.66	95.11
		10%	9.52	95.12
		18%	16.66	95.14
	ResNet-34	2%	24.49	94.20
		10%	5.44	94.33
		18%	26.53	94.21
	VGG-16	2%	40.61	94.37
		10%	50.25	94.42
		18%	28.43	94.30

Table 2 The attack results of the 2nd method using early stopping. The hyper-parameters for CIFAR-10 are set as follows: ResNet-50, $\delta = 0.001$, $b = 90$, patience = 60; ResNet-18, $\delta = 0.01$, $b = 90$, patience = 60; VGG-13, $\delta = 0.001$, $b = 90$, patience = 60. For SVHN, we set $\delta = 0.001$, $b = 90$, patience = 40 for all models.

Dataset	Architecture	Poison rate	Epoch change (%)	Accuracy (%)
CIFAR-10	ResNet-50	2%	5.29	95.04
		10%	30.40	94.40
		18%	32.15	94.30
	ResNet-18	2%	31.03	94.31
		10%	16.81	94.28
		18%	18.10	93.99
	VGG-13	2%	31.47	93.42
		10%	3.88	93.36
		18%	35.34	93.37
SVHN	ResNet-50	2%	26.19	94.99
		10%	32.14	95.10
		18%	34.52	95.01
	ResNet-34	2%	24.49	94.34
		10%	33.33	94.18
		18%	15.65	93.89
	VGG-16	2%	32.99	94.21
		10%	39.09	94.01
		18%	45.69	94.08

Table 3 The attack results on CIFAR-10 of the 2nd method using early stopping: ResNet-50, $\delta = 0.001$, $b = 90$, patience = 60; ResNet-18, $\delta = 0.001$, $b = 90$, patience = 40; VGG-13, $\delta = 0.001$, $b = 90$, patience = 60. The targeted probability of the label is 0.4.

Architecture	Poison rate	Epoch change (%)	Accuracy (%)
ResNet-50	2%	22.47	95.07
	10%	5.73	94.55
	18%	22.47	94.36
ResNet-18	2%	14.98	94.29
	10%	−7.69	94.47
	18%	11.76	94.09
VGG-13	2%	3.88	93.42
	10%	41.38	93.52
	18%	40.09	93.35

Table 4 The attack results on CIFAR-10 of the 2nd method using early stopping: ResNet-50, δ = 0.01, b = 90, patience = 60; ResNet-18, δ = 0.01, b = 90, patience = 40; VGG-13, δ = 0.005, b = 90, patience = 50. The targeted probability of the label is 0.6.

Architecture	Poison rate	Epoch change (%)	Accuracy (%)
ResNet-50	2%	34.36	94.72
	10%	37.00	94.87
	18%	11.45	94.20
ResNet-18	2%	4.17	94.47
	10%	−12.74	94.18
	18%	10.38	93.87
VGG-13	2%	4.95	93.56
	10%	38.29	93.40
	18%	−10.36	93.15

results in the white-box scenario and the black-box scenario in Tables 2, 3 and 4. The second method takes more epochs than the first method.

5.2 Ablation Study

There is a decrease in the accuracy of the attacked model. Taking the results on CIFAR-10 as an example, when attacking with the second method in the white-box scenario, the model accuracy drops off 0.7% than the unattacked model with the replacement ratio of 18%. We attempt to study whether we could increase the training time but not decrease the model accuracy by using different $\widetilde{y}$. In this study, the targeted probability of the correct class of each sample is set to 0.4 or 0.6. We apply the second attack method to attack the model. Tables 5 and 6 show the specific results. These models attacked by the new method have higher accuracy, while the cost epochs become less.

In our experimentation, the poison rate has a positive correlation to the attack performance. In both the white-box scenario and the black-box scenario, our attack methods are valid. However, the attack results are influenced by the parameters of the early-stopping algorithm. It is hard to find a group of fixed parameters valid for all conditions. Meanwhile, there is a trade-off between the cost epochs and the final model accuracy in most cases. Which side to take depends on the situation that the adversary wants to attack. If the attack needs to be invisible, it is better to select a small poison rate value and a highly targeted probability of the label, which leads to a higher accuracy but fewer cost epochs. For the contrary situation, a big poison rate and a small targeted probability are preferred. We notice some attack results are not stable, and it may have a connection to the robustness of the model. This phenomenon implies a small group of poisoned samples can help the model learn a better result.

6 Case Study 2: Targeted Accuracy Training

In this section, we consider another commonly used approach to end the training. It will stop training the model after some metrics reach pre-defined thresholds. Note that all empirical settings are the same as the ones in Section 4.2.

6.1 Evaluation on Training Efficiency

The poisoned samples generated by the two attack methods are the same as the former ones in case 1. Because the victim's stopping threshold is unknown, we set different values of t to evaluate the performance of our attack methodology. Tables 5 and 6 show the attack results on CIFAR-10 of the first method and the second method respectively. The considered parameter t varies from 86 to 93. For SVHN, we noticed that the model can easily obtain an acceptable accuracy (e.g., 90%) but has trouble reaching a higher accuracy (e.g., 94%+), for which we mainly focus on the higher values of t for SVHN in Tables 7 and 8. The replacement ratios in our attack methods are 2%, 10%, and 18%. The values in the tables are the change percentages of the cost epochs compared with the innocent model training phase. To determine the influence of the different poison rates in a visual manner, we illustrate the cost epochs of training ResNet-18 with the CIFAR-10 dataset under the first attack method in Fig. 6.

Table 5 The results on CIFAR-10 of epoch cost with an accuracy threshold (ACC_t) for different poison rates: the 1st method in case 2.

Architecture	Poison rate	$ACC_t = 86$	$ACC_t = 87$	$ACC_t = 88$	$ACC_t = 89$	$ACC_t = 90$	$ACC_t = 91$	$ACC_t = 92$	$ACC_t = 93$
ResNet-50 (ACC=94.89%)	2%	8.33%	−18.75%	12.50%	**37.50%**	19.23%	18.42%	21.43%	−2.97%
	10%	0.00%	0.00%	18.75%	**43.75%**	23.08%	13.16%	53.57%	6.93%
	18%	0.00%	12.50%	18.75%	**62.50%**	46.15%	26.32%	16.07%	17.82%
ResNet-18 (ACC=94.74%)	2%	9.09%	−18.75%	−5.56%	**21.05%**	8.33%	7.50%	20.75%	2.04%
	10%	9.09%	−25.00%	5.56%	15.79%	16.67%	−7.50%	**30.19%**	4.08%
	18%	0.00%	−6.25%	−5.56%	36.84%	**37.50%**	0.00%	28.30%	17.35%
VGG-13 (ACC=93.87%)	2%	−8.33%	7.69%	−23.81%	0.00%	**46.67%**	41.07%	21.28%	1.55%
	10%	−25.00%	15.38%	4.76%	7.69%	**26.67%**	21.43%	24.47%	6.98%
	18%	16.67%	7.69%	−9.52%	11.54%	16.67%	23.21%	10.64%	**31.78%**

The best results are in bold

Table 6 The results on CIFAR-10 of epoch cost with an accuracy threshold (ACC_t) for different poison rates: the 2nd method in case 2.

Architecture	Poison rate	$ACC_t = 86$	$ACC_t = 87$	$ACC_t = 88$	$ACC_t = 89$	$ACC_t = 90$	$ACC_t = 91$	$ACC_t = 92$	$ACC_t = 93$
ResNet-50 (ACC=94.89%)	2%	0.00%	−25.00%	25.00%	**43.75%**	11.54%	2.63%	8.93%	2.97%
	10%	0.00%	−12.50%	6.25%	**62.50%**	30.77%	28.95%	41.07%	4.95%
	18%	25.00%	−6.25%	37.50%	**87.50%**	69.23%	42.11%	48.21%	21.78%
ResNet-18 (ACC=94.74%)	2%	18.18%	−6.25%	−16.67%	−5.26%	4.17%	−10.00%	**28.30%**	14.29%
	10%	−27.27%	0.00%	−11.11%	10.53%	4.17%	10.00%	**11.32%**	4.08%
	18%	9.09%	0.00%	−11.11%	26.32%	29.17%	27.50%	**64.15%**	17.35%
VGG-13 (ACC=93.87%)	2%	0.00%	7.69%	−4.76%	−7.69%	**30.00%**	16.07%	17.02%	8.53%
	10%	−8.33%	23.08%	−19.05%	15.38%	16.67%	**50.00%**	13.83%	5.43%
	18%	25.00%	15.38%	23.81%	23.08%	50.00%	**55.36%**	24.47%	20.16%

The best results are in bold

The more data are poisoned, the longer the training time. In most cases, our attack is successful.

6.2 Ablation Study

We explore the performance of our attack method by setting the targeted probability of the correct class to 0.4 or 0.6 in the second method. The attack results are presented in Tables 9 and 10 respectively. Meanwhile, the targeted probability influences the highest model accuracy. While in most cases, the accuracy will reduce by less than 1%. A higher probability helps the attack more covert. For example, the highest model accuracy will not decrease when the targeted probability is 0.6, the attacked model is ResNet-50, and the poison rate is 10% in the second attack method. In this situation, the cost epochs increase 10% averagely in different *t*. If we further set the poison rate to 18%, we can improve the epochs increment percentage to 26% but limit the accuracy decrement to less than 0.3%. While the targeted probability is 0.2, the poison rate is 18%, and the other conditions maintain the same, the training costs more 32% epochs than the innocent situation.

The attack result connects to poison rates. More data are poisoned, and more epochs cost. In case 2, the attack results are more stable. It does not rely on *t*. No matter what *t* is, it usually takes more epochs to train a model. Thus, if the victim set thresholds to end the training, they are more vulnerable. However, when the poison rate is 2%, the attack results are not always successful. We consider it as the robustness of the model training.

7 Discussion and Future Work

There is very little related work on this topic. For instance, the method in [44] can increase the training time by maliciously reordering the samples in the training set. Such an attack, although effective, is not practical since a shuffle operation is set as default in most real-world cases.

In this paper, we have shown that by poisoning the dataset without any requirements for the order of the training set which is more practical to deploy. However, our work is still empirical work that shows the possibility of such a kind of attack. Our results of increasing the training time are highly relying on empirical factors such as the poison rate or the target accuracy which needs further improvement. Moreover, there are more criteria to stop the training process in real-world DNN training scenarios which may disable our attack.

Moreover, our attack is indeed orthogonal to these energy optimization methods. With different training devices or server configurations, our attack can significantly increase the training epoch numbers for the target accuracy which will lead to a malicious energy increase. In this paper, we try to focus on the model layer attack. Focusing on the model

Table 7 The results on SVHN of epoch cost corresponding to different accuracy thresholds (ACC_t): the 1st method in case 2.

Architecture	Poison rate	$ACC_t = 91$	$ACC_t = 92$	$ACC_t = 93$	$ACC_t = 94$
ResNet-50 (ACC=95.16%)	2%	0.00%	0.00%	56.25%	−3.70%
	10%	7.69%	0.00%	25.00%	14.81%
	18%	7.69%	28.57%	37.50%	−3.70%
ResNet-18 ACC=94.36%)	2%	−14.29%	−5.88%	−10.53%	213.04%
	10%	0.00%	0.00%	−10.53%	221.74%
	18%	−7.14%	−5.88%	−5.26%	856.52%
VGG-13 ACC=94.38%)	2%	−7.69%	−22.22%	5.00%	51.79%
	10%	−7.69%	5.55%	5.00%	50.00%
	18%	7.69%	−11.11%	0.00%	80.36%

Table 8 The results on SVHN of epoch cost corresponding to different accuracy thresholds (ACC_t): the 2nd method in case 2.

Architecture	Poison rate	$ACC_t = 91$	$ACC_t = 92$	$ACC_t = 93$	$ACC_t = 94$
ResNet-50 (ACC=95.16%)	2%	15.38%	21.43%	**62.50%**	7.41%
	10%	0.00%	7.14%	**50.00%**	18.52%
	18%	0.00%	−7.14%	**37.50%**	−11.11%
ResNet-18 (ACC=94.36%)	2%	7.14%	−11.76%	−10.53%	**126.09%**
	10%	14.28%	−5.88%	0.00%	**39.13%**
	18%	21.43%	0.00%	−5.26%	**973.91%**
VGG-13 (ACC=94.38%)	2%	−7.69%	0.00%	5.00%	**117.86%**
	10%	15.38%	0.00%	5.00%	**407.14%**
	18%	−7.69%	5.55%	20.00%	**301.79%**

The best results are in bold

Table 9 The results on the CIFAR-10 dataset of epoch cost with an accuracy threshold (ACC_t) for different poison rates: the 2nd method with 0.4 as the targeted probability of the label in case 2.

Architecture	Poison rate	$ACC_t = 86$	$ACC_t = 87$	$ACC_t = 88$	$ACC_t = 89$	$ACC_t = 90$	$ACC_t = 91$	$ACC_t = 92$	$ACC_t = 93$
ResNet-50 (ACC=94.89%)	2%	−8.33%	−6.25%	12.50%	**50.00%**	19.23%	5.26%	1.79%	−9.90%
	10%	0.00%	18.75%	25.00%	25.00%	**46.15%**	15.79%	42.86%	13.86%
	18%	8.33%	18.75%	18.75%	**75.00%**	30.77%	23.68%	30.36%	3.96%
ResNet-18 (ACC=94.74%)	2%	9.09%	−18.75%	−22.22%	21.05%	20.83%	20.00%	**37.74%**	10.20%
	10%	0.00%	−6.25%	−16.67%	10.53%	**33.33%**	−12.5%	32.08%	11.22%
	18%	9.09%	−6.25%	−0.56%	21.05%	20.83%	32.50%	**47.17%**	19.39%
VGG-13 (ACC=93.87%)	2%	−8.33%	−15.38%	−4.76%	15.38%	**63.33%**	51.79%	8.51%	2.33%
	10%	8.33%	15.38%	−14.29%	15.38%	30.00%	**48.21%**	9.57%	5.43%
	18%	8.33%	**61.54%**	23.81%	0.00%	56.67%	33.93%	19.15%	12.40%

The best results are in bold

Table 10 The results on the CIFAR-10 dataset of epoch cost with an accuracy threshold (ACC_t) for different poison rates: the 2nd method with 0.6 as the targeted probability of the label in case 2.

Architecture	Poison rate	$ACC_t = 86$	$ACC_t = 87$	$ACC_t = 88$	$ACC_t = 89$	$ACC_t = 90$	$ACC_t = 91$	$ACC_t = 92$	$ACC_t = 93$
ResNet-50 (ACC=94.89%)	2%	0.00%	−6.25%	18.75%	**43.75%**	26.92%	39.47%	32.14%	1.98%
	10%	−8.33%	−18.75%	18.75%	**50.00%**	26.92%	18.42%	44.64%	0.99%
	18%	8.33%	6.25%	18.75%	50.00%	38.46%	47.37%	**57.14%**	11.88%
ResNet-18 (ACC=94.74%)	2%	9.09%	−25.00%	−22.22%	21.05%	16.67%	−2.50%	**24.53%**	6.12%
	10%	9.09%	−12.50%	−5.56%	5.26%	16.67%	15.00%	**47.17%**	22.45%
	18%	0.00%	−31.25%	−5.56%	21.05%	41.67%	67.50%	**67.92%**	22.45%
VGG-13 (ACC=93.87%)	2%	0.00%	15.38%	−4.76%	−15.38%	**33.33%**	25.00%	2.13%	2.33%
	10%	−8.33%	38.46%	−4.76%	−11.54%	**43.33%**	28.57%	11.70%	10.85%
	18%	16.67%	7.69%	0.00%	7.69%	**60.00%**	32.14%	29.79%	14.73%

The best results are in bold

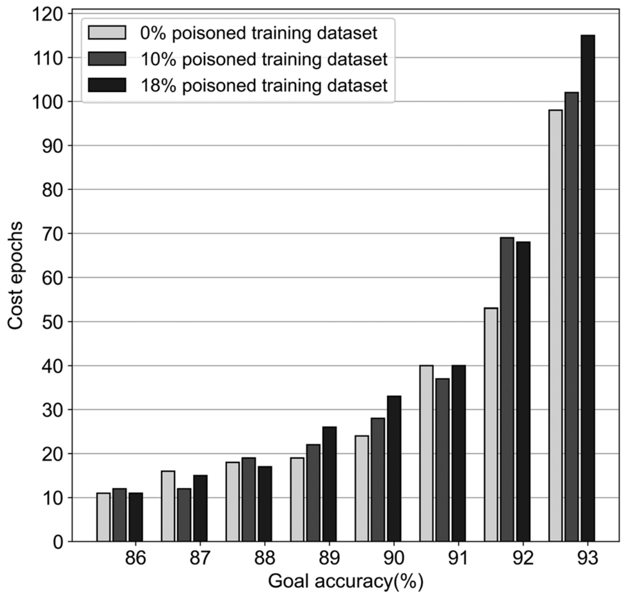

Figure 6 The attack result of the first method, targeting ResNet-18 trained on CIFAR-10. The label of the x-axis is goal accuracy. It represents the pre-defined threshold *t*. The value of each bar in the y-axis gives the cost epochs.

layer makes the attack results agnostic to the hardware or server configurations since increasing certain training epoch numbers can always increase the training efforts. We leave more practical attacks as our first future work.

For the second future work, we will focus on how to improve the attack results. We plan to generate poisoned samples that cost more epochs and do not influence the model's accuracy in the meantime. It will be hard for the victim to notice the attack. Meanwhile, we try to make our attack method more stable with the early stopping algorithm. Besides, it is valuable to experiment with other tasks to explore whether the attack works in different fields. Moreover, further experiments are needed to figure out the reasons for the energy attack. Such research will help us understand the energy attack better. Finally, it is also necessary to investigate the solutions to defend against the attack.

8 Conclusion

In this paper, we propose a novel energy attack targeting the increment of training time of a DNN model. The attack methodology consists of generating poisoned samples and poisoning the benign training dataset. In our experimentation, we evaluate our attack results with two algorithms to end the training. The attack results prove that it is feasible to increase the training time using our methodology. The future investigation will focus on improving the attack performance and the reason for the energy attack.

Funding No funding was received for conducting this study.

Data Availability Will be open-sourced.

Code Availability Will be open-sourced.

Declarations

Ethics Approval Not applicable.

Consent to Participate Yes.

Consent for Publication Yes.

Competing Interests The authors have no relevant financial or non-financial interests to disclose.

References

1. Wang, M., & Deng, W. (2020). Deep face recognition: a survey. *Neurocomputing*.
2. Shi, C., Ding, J., Cao, X., Hu, L., Wu, B., & Li, X. (2021). Entity set expansion in knowledge graph: A heterogeneous information network perspective. *Frontiers of Computer Science, 15*(1), 1–12.
3. Fu, Z., Gao, H., Guo, W., Jha, S. K., Jia, J., Liu, X., Long, B., Shi, J., Wang, S., & Zhou, M. (2020). Deep Learning for Search and Recommender Systems in Practice. In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 3515–3516.
4. Qiu, H., Qiu, M., & Lu, R. (2019). Secure v2x communication network based on intelligent pki and edge computing. *IEEE Network, 34*(2), 172–178.
5. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778.
6. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. arXiv preprint arXiv:1706.03762.
7. Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.
8. Zeng, W., Ren, X., Su, T., Wang, H., Liao, Y., Wang, Z., Jiang, X., Yang, Z., Wang, K., Zhang, X., et al. (2021). Pangu-α: Large-scale autoregressive pretrained chinese language models with auto-parallel computation. arXiv preprint arXiv:2104.12369.
9. Qiu, H., Zheng, Q., Memmi, G., Lu, J., Qiu, M., & Thuraisingham, B. (2020). Deep residual learning-based enhanced jpeg compression in the internet of things. *IEEE Transactions on Industrial Informatics, 17*(3), 2124–2133.
10. Chen, J., & Ran, X. (2019). Deep learning with edge computing: A review. *Proceedings of the IEEE, 107*(8), 1655–1674.
11. Joshi, A. V. (2020). Amazon's machine learning toolkit: Sagemaker. In: *Machine Learning and Artificial Intelligence*, pp. 233–243. Springer.
12. Ciaburro, G., Ayyadevara, V. K., & Perrier, A. (2018). Hands-On Machine Learning on Google Cloud Platform: Implementing Smart and Efficient Analytics Using Cloud ML Engine. Packt Publishing Ltd.
13. Barga, R., Fontama, V., & Tok, W. H. (2015). Introducing microsoft azure machine learning. In: *Predictive Analytics with Microsoft Azure Machine Learning*, pp. 21–43. Springer.
14. Henderson, P., Hu, J., Romoff, J., Brunskill, E., Jurafsky, D., & Pineau, J. (2020). Towards the systematic reporting of the energy

and carbon footprints of machine learning. *Journal of Machine Learning Research, 21*(248), 1–43.
15. So, D., Le, Q., & Liang, C. (2019). The evolved transformer. In: *International Conference on Machine Learning*, pp. 5877–5886. PMLR.
16. Strubell, E., Ganesh, A., & McCallum, A. (2019). Energy and policy considerations for deep learning in NLP. arXiv preprint arXiv:1906.02243.
17. Wang, Y., Ding, C., Li, Z., Yuan, G., Liao, S., Ma, X., Yuan, B., Qian, X., Tang, J., Qiu, Q., et al. (2018). Towards ultra-high performance and energy efficiency of deep learning systems: an algorithm-hardware co-optimization framework. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32.
18. Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., et al. (2017). In-datacenter performance analysis of a tensor processing unit. In: *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pp. 1–12.
19. Qiu, H., Dong, T., Zhang, T., Lu, J., Memmi, G., & Qiu, M. (2020). Adversarial attacks against network intrusion detection in IoT systems. *IEEE Internet of Things Journal*.
20. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., & Fergus, R. (2013). Intriguing properties of neural networks. arXiv preprint arXiv:1312.6199.
21. Li, Y., Wu, B., Jiang, Y., Li, Z., & Xia, S.-T. (2020). Backdoor learning: A survey. arXiv preprint arXiv:2007.08745.
22. Zhai, T., Li, Y., Zhang, Z., Wu, B., Jiang, Y., & Xia, S.-T. (2021). Backdoor attack against speaker verification. In: *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2560–2564. IEEE.
23. Qiu, H., Zeng, Y., Guo, S., Zhang, T., Qiu, M., & Thuraisingham, B. (2021). Deepsweep: An evaluation framework for mitigating dnn backdoor attacks using data augmentation. In: *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, pp. 363–377.
24. Carlini, N., & Wagner, D. (2017). Towards evaluating the robustness of neural networks. In: *2017 IEEE Symposium on Security and Privacy (sp)*, pp. 39–57. IEEE.
25. Athalye, A., Carlini, N., & Wagner, D. (2018). Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In: *International Conference on Machine Learning*, pp. 274–283. PMLR.
26. Qiu, H., Zeng, Y., Zheng, Q., Guo, S., Zhang, T., & Li, H. (2021). An efficient preprocessing-based approach to mitigate advanced adversarial attacks. *IEEE Transactions on Computers*.
27. Hong, S., Kaya, Y., Modoranu, I.-V., & Dumitraş, T. (2020). A Panda? No, It's a Sloth: Slowdown Attacks on Adaptive Multi-Exit Neural Network Inference. arXiv preprint arXiv:2010.02432.
28. Çalik, R. C., & Demirci, M. F. (2018). Cifar-10 image classification with convolutional neural networks for embedded systems. In: *2018 IEEE/ACS 15th International Conference on Computer Systems and Applications (AICCSA)*, pp. 1–2. IEEE.
29. Szegedy, C., Ioffe, S., Vanhoucke, V., & Alemi, A. A. (2017). Inception-v4, inception-resnet and the impact of residual connections on learning. In: *Thirty-first AAAI Conference on Artificial Intelligence*.
30. Zhang, Q., Bai, C., Liu, Z., Yang, L. T., Yu, H., Zhao, J., & Yuan, H. (2020). A gpu-based residual network for medical image classification in smart medicine. *Information Sciences, 536*, 91–100.
31. Hassan, M. M., Gumaei, A., Alsanad, A., Alrubaian, M., & Fortino, G. (2020). A hybrid deep learning model for efficient intrusion detection in big data environment. *Information Sciences, 513*, 386–396.
32. He, Y., Liu, P., Wang, Z., Hu, Z., & Yang, Y. (2019). Filter pruning via geometric median for deep convolutional neural networks acceleration. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4340–4349.
33. Hinton, G., Vinyals, O., & Dean, J. (2015). Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531.
34. Han, S., Mao, H., & Dally, W. J. (2015). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149.
35. Wang, C., Gong, L., Yu, Q., Li, X., Xie, Y., & Zhou, X. (2016). DLAU: A scalable deep learning accelerator unit on FPGA. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 36*(3), 513–517.
36. Liao, H., Tu, J., Xia, J., & Zhou, X. (2019). Davinci: A scalable architecture for neural network computing. In: *2019 IEEE Hot Chips 31 Symposium (HCS)*, pp. 1–44. *IEEE Computer Society*.
37. Chen, T., Du, Z., Sun, N., Wang, J., Wu, C., Chen, Y., & Temam, O. (2014). Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. *ACM SIGARCH Computer Architecture News, 42*(1), 269–284.
38. Acun, B., Murphy, M., Wang, X., Nie, J., Wu, C.-J., & Hazelwood, K. (2021). Understanding training efficiency of deep learning recommendation models at scale. In: *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 802–814. IEEE.
39. Zhang, L., & Suganthan, P. N. (2016). A survey of randomized algorithms for training neural networks. *Information Sciences, 364*, 146–155.
40. Akita, R., Yoshihara, A., Matsubara, T., & Uehara, K. (2016). Deep learning for stock prediction using numerical and textual information. In: *2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS)*, pp. 1–6. IEEE.
41. Qiu, H., Noura, H., Qiu, M., Ming, Z., & Memmi, G. (2019). A user-centric data protection method for cloud storage based on invertible DWT. *IEEE Transactions on Cloud Computing*.
42. Grosse, K., Trost, T. A., Mosbach, M., Backes, M., & Klakow, D. (2019). On the security relevance of weights in deep learning. arXiv preprint arXiv:1902.03020.
43. Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., Liu, T., Wang, X., Wang, G., Cai, J., et al. (2018). Recent advances in convolutional neural networks. *Pattern Recognition, 77*, 354–377.
44. Shumailov, I., Shumaylov, Z., Kazhdan, D., Zhao, Y., Papernot, N., Erdogdu, M. A., & Anderson, R. (2021). Manipulating SGD with data ordering attacks. arXiv preprint arXiv:2104.09667.
45. Madry, A., Makelov, A., Schmidt, L., Tsipras, D., & Vladu, A. (2017). Towards deep learning models resistant to adversarial attacks. arXiv preprint arXiv:1706.06083.