Indiscriminate Data Poisoning Attacks on Neural Networks*

Yiwei Lu, Gautam Kamath†, Yaoliang Yu
{yiwei.lu, gckamath, yaoliang.yu}@uwaterloo.ca
University of Waterloo

Abstract

Data poisoning attacks, in which a malicious adversary aims to influence a model by injecting "poisoned" data into the training process, have attracted significant recent attention. In this work, we take a closer look at existing poisoning attacks and connect them with old and new algorithms. By choosing an appropriate loss function for the attacker and optimizing with algorithms that exploit second-order information, we design poisoning attacks that are effective on neural networks. We present efficient implementations by parameterizing the attacker and allowing simultaneous and coordinated generation of tens of thousands of poisoned points, in contrast to existing methods that generate poisoned points one by one. We further perform extensive experiments that empirically explore the effect of data poisoning attacks on deep neural networks. Our paper set up a new benchmark on the possibility of performing indiscriminate data poisoning attacks on modern neural networks.

1 Introduction

Adversarial attacks have repeatedly demonstrated critical vulnerabilities in modern machine learning (ML) models [25, 33, 21]. As ML systems are deployed in increasingly important settings, significant effort has been levied in understanding attacks and defenses towards *robust* machine learning.

In this paper, we focus on *data poisoning attacks*. ML models require a large amount of data to achieve good performance, and thus practitioners frequently gather data by scraping content from the web [11, 35]. This gives rise to an attack vector, in which an adversary may manipulate part of the training data by injecting poisoned samples. For example, an attacker can *actively* manipulate datasets by sending corrupted samples directly to a dataset aggregator such as a chatbot, a spam filter, or user profile databases; the attacker can also *passively* manipulate datasets by placing poisoned data on the web and waiting for collection. Moreover, in *federated learning*, adversaries can also inject malicious data into a diffuse network [29, 22].

A spectrum of such data poisoning attacks exists in the literature, including *targeted*, *indiscriminate* and *backdoor* attacks (see Appendix A for a detailed comparison). We focus on indiscriminate attacks for image classification, where the attacker aims at decreasing the overall test accuracy of a model by adding a small portion of poisoned points. Current indiscriminate attacks are most effective against convex models [2, 18, 19, 30], and several defenses have also been proposed [31, 6]. However, existing poisoning attacks are less adequate against more complex non-convex models, especially deep neural networks, either due to their formulation being inherently tied to convexity or computational limitation. For example, most prior attacks generate poisoned points sequentially.

*GK and YY are listed in alphabetical order.

†Supported by an NSERC Discovery Grant, an unrestricted gift from Google, and a University of Waterloo startup grant.

ML Safety Workshop, 36th Conference on Neural Information Processing Systems (NeurIPS 2022).

Thus, when applied to deep models or large datasets, these attacks quickly become computationally infeasible. To our knowledge, a systematic analysis on poisoning deep neural works is still largely missing—a gap we aim to fill in this work.

To address this difficult problem, we design more versatile data poisoning attacks by formulating the problem as a bilevel optimization problem, in which the attacker crafts some poisoned points with the aim of decreasing the test accuracy, while the defender optimizes its model on the poisoned training set. We exploit second-order information and apply the Total Gradient Descent Ascent (TGDA) algorithm to address the attacker's objective, even on non-convex models.

Moreover, we address computational challenges by proposing an efficient architecture for poisoning attacks, where we parameterize the attacker as a separate network rather than optimizing the poisoned points directly. By applying TGDA to update the attacker model directly, we are able to generate tens of thousands of poisoned points simultaneously in one pass, potentially even in a coordinated way.

In this work, we make the following contributions:

(1) We construct a new data poisoning attack based on TGDA that incorporates second-order optimization. In comparison to prior data poisoning attacks, ours is significantly more effective and runs at least an order of magnitude faster.
(2) We propose an efficient attack architecture, which enables a more efficient, clean-label attack.
(3) We conduct experiments to demonstrate the effectiveness of our attack on neural networks and its advantages over previous methods.

2 Total Gradient Descent Ascent Attack

In this section, we formulate the indiscriminate attack and introduce our attack algorithm.

Formulation. In data poisoning attacks, there are two parties, namely the attacker L and the defender F. The defender aims at minimizing its loss function f under data poisoning:

$$\mathbf{w}_* = \mathbf{w}_*(\mathcal{D}_p) \in \arg\min_{\mathbf{w}} \ \mathcal{L}(\mathcal{D}_{tr} \cup \mathcal{D}_p, \mathbf{w}), \tag{1}$$

while the attacker aims at maximizing a different loss function ℓ on the validation set $\mathcal{D}_v$:

$$\mathcal{D}_{p_*} \in \arg\max_{\mathcal{D}_p} \mathcal{L}(\mathcal{D}_v, \mathbf{w}_*), \tag{2}$$

where the loss function $\mathcal{L}(\cdot)$ can be any task-dependent target criterion, e.g., the cross-entropy loss. Thus we have arrived at the bilevel optimization problem [23, 17, 19]):

$$\max_{\mathcal{D}_p} \mathcal{L}(\mathcal{D}_v, \mathbf{w}_*), \text{ s.t. } \mathbf{w}_* \in \arg\min_{\mathbf{w}} \mathcal{L}(\mathcal{D}_{tr} \cup \mathcal{D}_p, \mathbf{w}). \tag{3}$$

Previous approaches. While the inner minimization can be solved via gradient descent, the outer maximization problem is non-trivial as the dependence of $\mathcal{L}(\mathcal{D}_v, \mathbf{w}_*)$ on $\mathcal{D}_p$ is *indirectly* through the parameter $\mathbf{w}$ of the poisoned model. Thus, *applying simple algorithms (e.g., Gradient Descent Ascent) directly will result in zero gradient*. Nevertheless, we can rewrite the desired derivative using the chain rule: $\frac{\partial \mathcal{L}(\mathcal{D}_v, \mathbf{w}_*)}{\partial \mathcal{D}_p} = \frac{\partial \mathcal{L}(\mathcal{D}_v, \mathbf{w}_*)}{\partial \mathbf{w}_*} \frac{\partial \mathbf{w}_*}{\partial \mathcal{D}_p}$. The difficulty lies in computing $\frac{\partial \mathbf{w}_*}{\partial \mathcal{D}_p}$, i.e., measuring how much the model parameter $\mathbf{w}$ changes with respect to the poisoned points $\mathcal{D}_p$. Various approaches compute $\frac{\partial \mathbf{w}_*}{\partial \mathcal{D}_p}$ by solving this problem exactly via KKT conditions [2, 18], or approximately using gradient ascent [23].

TGDA attack. However, we can avoid such calculation using the Total gradient descent ascent (TGDA) algorithm [7, 8]: TGDA takes a total gradient ascent step for the attacker and a gradient descent step for the defender:

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \eta_t \mathsf{D}_{\mathbf{x}} \ell(\mathbf{x}_t, \mathbf{w}_t), \tag{4}$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \nabla_{\mathbf{w}} f(\mathbf{x}_t, \mathbf{w}_t) \tag{5}$$

where $\mathsf{D}_{\mathbf{x}} := \nabla_{\mathbf{x}} \ell - \nabla_{\mathbf{wx}} f \cdot \nabla_{\mathbf{ww}}^{-1} f \cdot \nabla_{\mathbf{w}} \ell$ is the total derivative of ℓ with respect to $\mathbf{x}$, which implicitly measures the change of $\mathbf{w}$ with respect to $\mathcal{D}_p$.

We thus apply the total gradient descent ascent algorithm and call this the **TGDA attack**. Avoiding computing $\frac{\partial \mathbf{w}_*}{\partial \mathcal{D}_p}$ enables us to parameterize $\mathcal{D}_p$ and generate points indirectly by treating L as a

separate model. Namely that $\mathcal{D}_p = \mathsf{L}_\theta(\mathcal{D}'_{tr})$, where θ is the model parameter and $\mathcal{D}'_{tr}$ is part of the training set to be poisoned. Thus, we have arrived a poisoning attack that generates $\mathcal{D}_p$ in a batch rather than individually, which greatly improves the attack efficiency.

3 An Efficient Attack architecture

In this section, we discuss how we use the TGDA attack to efficiently perform data poisoning attacks. We observe that existing data poisoning attacks have **two limitations**:
(1) Existing attacks work under subtly different assumptions, on, for example, the attacker's knowledge, the attack formulation, and the training set size. These inconsistencies result in unfair comparisons between methods (see Table 3 in Appendix B for a detailed comparison). Thus we set an experimental protocol for generalizing existing attacks in Appendix B and benchmarking data poisoning attacks for systematic analysis in the future.

(2) They approach the problem by optimizing individual points directly, thus having to generate poisoned points one by one. We design a new poisoning scheme that allows simultaneous and coordinated generation of $\mathcal{D}_p$ in batches requiring only one pass that involves three stages:

Pretrain: The goals of the attacker L are to: reduce the test accuracy (i.e., successfully attack) and generate $\mathcal{D}_p$ that is close to $\mathcal{D}_{tr}$ (i.e., thwart potential defenses). The attacker achieves the first objective during the attack by optimizing ℓ. However, ℓ does not enforce that the distribution of the poisoned points will resemble those of the training set. To this end, we pretrain L to reconstruct $\mathcal{D}_{tr}$, producing a parameter vector θ_{pre}. This process is identical to training an autoencoder.
For the defender, we assume that F is fully trained to convergence. Thus we perform standard training on $\mathcal{D}_{tr}$ to acquire F with $\mathbf{w}_{pre}$.

Attack: We generate poisoned points using the TGDA attack. We assume that the attacker can inject ϵN poisoned points, where $N = |\mathcal{D}_{tr}|$ and ϵ is the power of the attacker, measured as a fraction of the training set size.
(a) Initialization: We take the pretrained model L with parameter θ_{pre} and F with pretrained parameter $\mathbf{w}_{pre}$ as initialization of the two networks; the complete training set $\mathcal{D}_{tr}$; a validation set $\mathcal{D}_v$ and part of the training set as initialization of the poisoned points $\mathcal{D}_p^0 = \mathcal{D}_{tr}[0 : \epsilon N]$.
(b) TGDA attack: We then perform TGDA attack using the updates in Section 2. We use total gradient ascent to update attacker parameter θ of L, and gradient descent to update defender parameter $\mathbf{w}$ of F.
(c) Label Information: We specify that $\mathcal{D}_p^0 = \{x_i, y_i\}_{i=1}^{\epsilon N}$. Prior works (e.g., [19, 23]) optimize x to produce x_p, and perform a label flip on y to produce y_p . This approach neglects label information during optimization. In contrast, we **fix** $y_p = y$, and concatenate x and y to $\mathcal{D}_p^0 = \{x_i; y_i\}_{i=1}^{\epsilon N}$ as input to L. Thus we generate poisoned points by considering the label information. We emphasize that we do not optimize or change the label during the attack, but merely use it to aid the construction of the poisoned x_p. Thus, our attack belongs to the clean label category.

Testing: Finally, we discuss how we measure the effectiveness of an attack. In a realistic setting, the testing procedure should be identical to the pretrain procedure, such that we can measure the effectiveness of $\mathcal{D}_p$ fairly. The consistency between pretrain and testing is crucial as the model F is likely to underfit with fewer training steps.

4 Experiments

We evaluate our TGDA attack on various models for image classification tasks and show the efficacy of our method for poisoning neural networks. Specifically, our results confirm that (1) we can attack neural networks with improved efficiency and efficacy using the TGDA attack. (2) The poisoned points are visually similar to clean data, making the attack intuitively resistant to defenses.

See Appendix C for a detailed list of our experimental settings (hardware and package, model details, dataset split, pretrain hyperparameters and complete baseline methods).

4.1 Comparison with Benchmarks

MNIST. We compare our attack with the Back-gradient and the Label flip attacks with $\epsilon = 3\%$ on MNIST in Table 1. Since the Back-gradient attack relies on generating poisoned points sequentially,

Table 1: The attack accuracy/accuracy drop (%) and attack running time (hours) on the MNIST dataset. We only record the attack running time since pretrain and testing time are fixed across different methods. As the label flip attack does not involve optimization, its running time is always 0. Our attack outperforms the Back-gradient attack in terms of both effectiveness and efficiency across three different models.

Target Model	Clean	**Label Flip**		**Back-gradient**		**TGDA(ours)**	
	Accuracy	Accuracy/Drop	Running time	Accuracy/Drop	Running time	Accuracy/Drop	Running time
LR	92.35	90.83 / 1.52	0 hrs	89.82 / 2.53	27 hrs	89.56 / **2.79**	1.1 hrs
NN	98.04	97.99 / 0.05	0 hrs	97.67 / 0.37	239 hrs	96.54 / **1.50**	15 hrs
CNN	99.13	99.12 / 0.01	0 hrs	99.02 / 0.09	2153 hrs	98.02 / **1.11**	75 hrs

Table 2: The attack accuracy/accuracy drop (%) and attack running time (hours) on CIFAR-10.

	Clean	Label Flip	MetaPoison	TGDA(ours)
Accuracy/Drop	69.44	68.99 / 0.45	68.14/1.13	65.15 / **4.29**
Running time	0 hrs	0 hrs	75hrs	346 hrs

we cannot adapt it into our unified architecture and run their code directly for comparison. For the label flip attack, we flip the label according to the rule that $y \leftarrow 10 - y$ as there are 10 classes.

We observe that label flip attack, though very efficient, is not effective against neural networks. Although [23] shows empirically that the Back-gradient attack is effective when attacking subsets of MNIST (1,000 training samples, 5,000 testing samples), we show that the attack is much less effective on the full dataset. We also observe that the complexity of the target model affects the attack effectiveness significantly. Specifically, we find that neural networks are generally more robust against indiscriminate data poisoning attacks, among which, the CNN architecture is even more robust. Overall, our method outperforms the baseline methods across the three target models. Moreover, with our unified architecture, we significantly reduce the running time of poisoning attacks by more than an order of magnitude.

CIFAR-10. We compare our attack with the Label flip attack and the MetaPoison attack with $\epsilon = 3\%$ on CIFAR-10 in Table 2. We omit comparison with the Back-gradient attack as it is too computationally expensive to run on CIFAR-10. We observe that the TGDA attack is very effective at poisoning the CNN architecture, but the running time becomes infeasible on larger models (e.g., ResNet). Also, MetaPoison is a more efficient attack (meta-learning with 2 unrolled steps are much quick than calculating total gradient), but since its original objective is to perform targeted attack, its application on indiscriminate attack is not effective. Moreover, the difference between the efficacy of the TGDA attack on MNIST and CIFAR-10 suggests that indiscriminate attacks may be dataset dependent, with MNIST being harder to poison than CIFAR-10.

4.2 Visualization of attacks

We visualize some poisoned points $\mathcal{D}_p$ generated by the TGDA attack in Figure 1. The poisoned samples against NN and CNN are visually very similar with $\mathcal{D}_{tr}$, which provides heuristic evidence that the TGDA attack may be robust against data sanitization algorithms. We further quantitatively evaluate the robustness of TGDA attack agaisnt data sanitization algorithms in Appendix C.

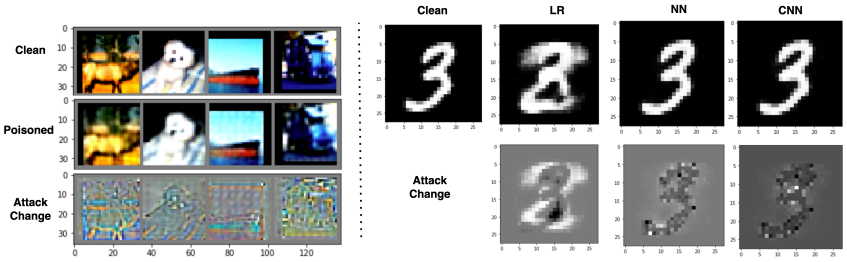

Figure 1: We visualize the poisoned data generated by the TGDA attack (left: CIFAR-10; right: MNIST).

References

[1] Hojjat Aghakhani, Dongyu Meng, Yu-Xiang Wang, Christopher Kruegel, and Giovanni Vigna. Bullseye polytope: A scalable clean-label poisoning attack with improved transferability. arXiv preprint arXiv:2005.00191, 2020.

[2] Battista Biggio, Blaine Nelson, and Pavel Laskov. Support vector machines under adversarial label noise. In *Proceedings of the Asian Conference on Machine Learning (ACML)*, pages 97–112, 2011.

[3] Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines. In *Proceedings of the 29th International Conference on Machine Learning (ICML)*, pages 1467–1474, 2012.

[4] Mengjie Chen, Chao Gao, and Zhao Ren. Robust covariance and scatter matrix estimation under Huber's contamination model. *The Annals of Statistics*, 2018.

[5] Li Deng. The MNIST database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[6] Ilias Diakonikolas, Gautam Kamath, Daniel M. Kane, Jerry Li, Jacob Steinhardt, and Alistair Stewart. Sever: A robust meta-algorithm for stochastic optimization. In *Proceedings of the 36th International Conference on Machine Learning*, pages 1596–1606, 2019.

[7] Yu. G. Evtushenko. Iterative methods for solving minimax problems. *USSR Computational Mathematics and Mathematical Physics*, 14(5):52–63, 1974.

[8] Tanner Fiez, Benjamin Chasnov, and Lillian J Ratliff. Implicit learning dynamics in Stackelberg games: Equilibria characterization, convergence analysis, and empirical study. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2020.

[9] Liam Fowl, Ping-yeh Chiang, Micah Goldblum, Jonas Geiping, Arpit Bansal, Wojtek Czaja, and Tom Goldstein. Preventing unauthorized use of proprietary data: Poisoning for secure dataset release. arXiv preprint arXiv:2103.02683, 2021.

[10] Liam Fowl, Micah Goldblum, Ping-yeh Chiang, Jonas Geiping, Wojtek Czaja, and Tom Goldstein. Adversarial examples make strong poisons. arXiv preprint arXiv:2106.10807, 2021.

[11] Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. The pile: An 800gb dataset of diverse text for language modeling. arXiv preprint arXiv:2101.00027, 2020.

[12] Micah Goldblum, Dimitris Tsipras, Chulin Xie, Xinyun Chen, Avi Schwarzschild, Dawn Song, Aleksander Madry, Bo Li, and Tom Goldstein. Dataset security for machine learning: Data poisoning, backdoor attacks, and defenses. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[13] Chengyue Gong, Tongzheng Ren, Mao Ye, and Qiang Liu. Maxup: A simple way to improve generalization of neural network training. *arXiv preprint arXiv:2002.09024*, 2020.

[14] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. arXiv:1708.06733, 2017.

[15] Junfeng Guo and Cong Liu. Practical poisoning attacks on neural networks. In *European Conference on Computer Vision*. Springer, 2020.

[16] Hanxun Huang, Xingjun Ma, Sarah Monazam Erfani, James Bailey, and Yisen Wang. Unlearnable examples: Making personal data unexploitable. arXiv preprint arXiv:2101.04898, 2021.

[17] W Ronny Huang, Jonas Geiping, Liam Fowl, Gavin Taylor, and Tom Goldstein. Metapoison: Practical general-purpose clean-label data poisoning. In *Advances in Neural Information Processing Systems*, volume 33, pages 12080–12091, 2020.

[18] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pages 1885–1894, 2017.

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

[20] Alex Krizhevsky. Learning multiple layers of features from tiny images. tech. report, 2009.

[21] Ram Shankar Siva Kumar, Magnus Nyström, John Lambert, Andrew Marshall, Mario Goertzel, Andi Comissoneru, Matt Swann, and Sharon Xia. Adversarial machine learning-industry perspectives. In *2020 IEEE Security and Privacy Workshops (SPW)*, pages 69–75. IEEE, 2020.

[22] Lingjuan Lyu, Han Yu, and Qiang Yang. Threats to federated learning: A survey. arXiv preprint arXiv:2003.02133, 2020.

[23] Luis Muñoz-González, Battista Biggio, Ambra Demontis, Andrea Paudice, Vasin Wongrassamee, Emil C. Lupu, and Fabio Roli. Towards poisoning of deep learning algorithms with back-gradient optimization. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security (AISec)*, pages 27–38, 2017.

[24] Luis Muñoz-González, Bjarne Pfitzner, Matteo Russo, Javier Carnerero-Cano, and Emil C Lupu. Poisoning attacks with generative adversarial nets. arXiv preprint arXiv:1906.07773, 2019.

[25] Blaine Nelson, Marco Barreno, Fuching Jack Chi, Anthony D Joseph, Benjamin IP Rubinstein, Udam Saini, Charles Sutton, J Doug Tygar, and Kai Xia. Exploiting machine learning to subvert your spam filter. *LEET*, 8:1–9, 2008.

[26] Aniruddha Saha, Akshayvarun Subramanya, and Hamed Pirsiavash. Hidden trigger backdoor attacks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

[27] Avi Schwarzschild, Micah Goldblum, Arjun Gupta, John P Dickerson, and Tom Goldstein. Just how toxic is data poisoning? A unified benchmark for backdoor and data poisoning attacks. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, 2021.

[28] Ali Shafahi, W. Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. Poison frogs! Targeted clean-label poisoning attacks on neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 6103–6113, 2018.

[29] Virat Shejwalkar, Amir Houmansadr, Peter Kairouz, and Daniel Ramage. Back to the drawing board: A critical evaluation of poisoning attacks on federated learning. arXiv:2108.10241, 2021.

[30] Ilia Shumailov, Zakhar Shumaylov, Dmitry Kazhdan, Yiren Zhao, Nicolas Papernot, Murat A Erdogdu, and Ross Anderson. Manipulating SGD with data ordering attacks. arXiv:2104.09667, 2021.

[31] Jacob Steinhardt, Pang Wei Koh, and Percy Liang. Certified defenses for data poisoning attacks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 3520–3532, 2017.

[32] Fnu Suya, Saeed Mahloujifar, Anshuman Suri, David Evans, and Yuan Tian. Model-targeted poisoning attacks with provable convergence. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, pages 10000–10010, 2021.

[33] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. arXiv preprint arXiv:1312.6199, 2013.

[34] Brandon Tran, Jerry Li, and Aleksander Madry. Spectral signatures in backdoor attacks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[35] Jane Wakefield. Microsoft chatbot is taught to swear on twitter. BBC News, 2016.

[36] Da Yu, Huishuai Zhang, Wei Chen, Jian Yin, and Tie-Yan Liu. Indiscriminate poisoning attacks are shortcuts. arXiv preprint arXiv:2111.00898, 2021.

[37] Guojun Zhang, Kaiwen Wu, Pascal Poupart, and Yaoliang Yu. Newton-type methods for minimax optimization. In *ICML workshop on Beyond First-Order Methods in ML Systems*, 2021.

[38] Chen Zhu, W Ronny Huang, Hengduo Li, Gavin Taylor, Christoph Studer, and Tom Goldstein. Transferable clean-label poisoning attacks on deep neural nets. In *International Conference on Machine Learning*. PMLR, 2019.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes]
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

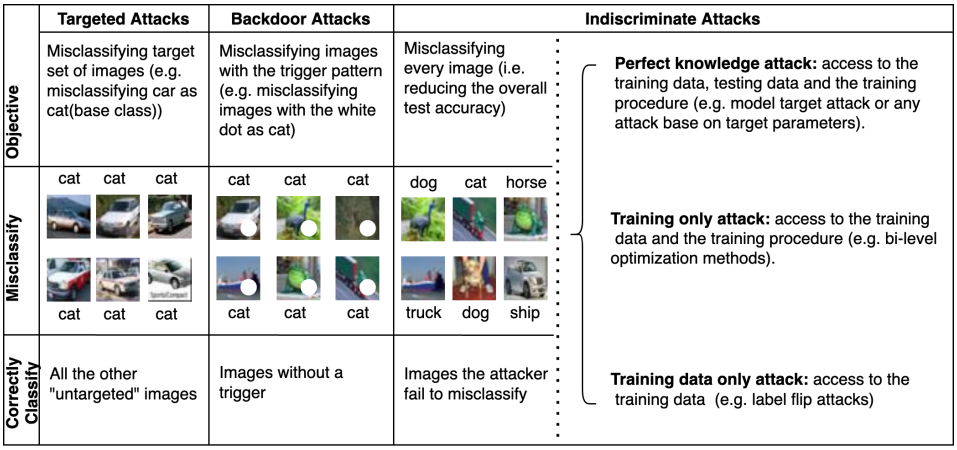

Figure 2: A taxonomy of data poisoning attacks for image classification. According to different objectives, *Target attacks* aim to misclassify a specific target set of images, *backdoor attacks* aim to misclassify any image with a specific trigger, and indiscriminate attacks aim at reducing the test accuracy in general. Among indiscriminate attacks, we can also identify three types of attacks according to the adversary's capability.

A Background on Data Poisoning Attacks

In this appendix, we categorize existing data poisoning attacks according to the attacker's *power* and *objectives*, and specify the type of attack we study in this paper.

A.1 Power of an attacker

Injecting poisoned samples. Normally, without breaching one's database (i.e., changing the existing training data $\mathcal{D}_{tr}$), an attacker can only *inject* poisoned data, actively or passively to the defender's database, such that its objective can be achieved when the model is retrained after collection. Where the goal of the attacker can be presented as:

$$\mathbf{w}_* = \mathbf{w}_*(\mathcal{D}_p) \in \arg\min_{\mathbf{w}} \ \mathcal{L}(\mathcal{D}_{tr} \cup \mathcal{D}_p, \mathbf{w}), \tag{6}$$

where $\mathbf{w}_*$ is the desired model parameter, which realizes its respective objectives. We focus on such attacks and further categorize such objectives in the next subsection.

Perturbing training data $\mathcal{D}_{tr}$. An attacker may change $\mathcal{D}_{tr}$ directly on one occasion, where the attacker is also the "defender". Such an attacker can be the dataset owner, who intends to preserve a proprietary dataset to release but prevents others from benefiting by training it [16, 36, 10, 9], thus allows modification up to the entire $\mathcal{D}_{tr}$ to make it unlearnable.

For simplicity, we refer to injecting poisoned samples as *data poisoning attacks*, and perturbing training data as *dataset protection* in this paper.

A.2 Objective of an attacker

Data poisoning attacks can be further classified into three categories according to the adversary's objective [12]. See Figure 2 for a taxonomic illustration.

Targeted attack. The attacker adds poisoned data $\mathcal{D}_p$ and acquire $\mathbf{w}^*$ such that a particular target example from the test set is misclassified as the *base* class [28, 1, 15, 38].This topic is well studied in the literature, and we refer the reader to [27] for an excellent summary of existing methods.

Backdoor attack. This attack aims at misclassifying any test input with a particular trigger pattern [14, 34, 4, 26]. Note that backdoor attacks require access to the input during inference time to plant the trigger.

Indiscriminate attack. This attack aims to acquire $\mathbf{w}^*$ by injecting poisoned data which decreases the model accuracy overall (or deny of service). We consider image classification tasks where

Table 3: Summary of existing poisoning attack algorithms. While some papers may include experiments on other datasets, we only cover vision datasets as our main focus is image classification. The attacks: Random label flip and Adversarial label flip attacks [2], P-SVM: PoisonSVM attack [2], Min-max attack [31], KKT attack [19], i-Min-max: improved Min-max attack [19], MT: Model Targeted attack [32], BG: Back-gradient attack [23].

Attack	Dataset	Model	$\lvert\mathcal{D}_{tr}\rvert$	$\lvert\mathcal{D}_{test}\rvert$	ϵ	Code	Multiclass	Batch
Random label flip	toy	SVM	/	/	0-40%	git	✓	$\epsilon\lvert\mathcal{D}_{tr}\rvert$
Adversarial label flip	toy	SVM	/	/	0-40%	git	×	$\epsilon\lvert\mathcal{D}_{tr}\rvert$
P-SVM	MNIST-17	SVM	100	500	0-9%	git	×	1
Min-max	MNIST-17/Dogfish	SVM	60000	10000	0-30%	git	✓	1
KKT	MNIST-17/Dogfish	SVM/LR	13007/1800	2163/600	3%	git	×	1
i-Min-max	MNIST	SVM	60000	10000	3%	git	✓	1
MT	MNIST-17/Dogfish	SVM/LR	13007/1800	2163/600	/	git	✓	1
BG	MNIST	SVM, NN	1000	8000	0-6%	git	✓	1

the attacker aims to reduce the overall classification accuracy. Existing methods make different assumptions on the attacker's knowledge:

- Perfect knowledge attack: the attacker has access to both training and test data ($\mathcal{D}_{tr}$ and $\mathcal{D}_{test}$), the target model, and the training procedure (e.g., the min-max attack of [19]).
- Training-only attack: the attacker has access to training data $\mathcal{D}_{tr}$, the target model, and the training procedure (e.g., [23, 3]).
- Training-data-only attack: the attacker only has access to the training data $\mathcal{D}_{tr}$ (e.g., the label flip attack of [2]).

In this work, we focus on training-only attacks because perfect knowledge attacks are not always feasible due to the proprietary nature of the test data, while existing training-data-only attacks are weak and often fail for deep neural networks, as we show in Section 4.

B Experimental Protocol

We first summarize existing indiscriminate data poisoning attacks in Table 3, where we identify that such attacks work under subtly different assumptions, on, for example, the attacker's knowledge and the training set size. These inconsistencies result in unfair comparisons between methods .

Thus we set an experimental protocol for generalizing existing attacks and benchmarking data poisoning attacks for systematic analysis in the future. Here we fix three key variants: (1) the attacker's knowledge: as discussed in Appendix A, we consider training-only attacks;

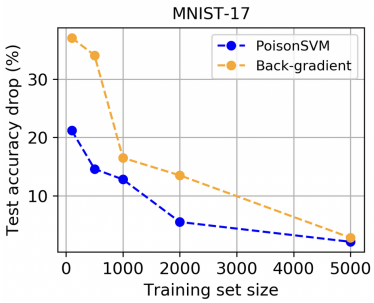

Figure 3: Comparing the efficacy of poisoning MNIST-17 with the PoisonSVM and Back-gradient attacks. The training set size is varied, while the ratio of the number of poisoned points to the training set size is fixed at 3%. These attacks become less effective as training set sizes increase.

(2) the dataset size: existing works measure attack efficacy with respect to the size of the poisoned dataset, where size is measured as a *fraction* ϵ of the training dataset. However, some works subsample and thus reduce the size of the training dataset. As we show in Figure 3, attack efficacy is *not* invariant to the size of the training set: larger training sets appear to be harder to poison. Furthermore, keeping ϵ fixed, a smaller training set reduces the number of poisoned data points and thus the time required for methods that generate points sequentially, potentially concealing a prohibitive runtime for poisoning the full training set. Thus we consider not only a fixed ϵ, but also the complete training set for attacks.

Table 4: Comparison with pGAN on MNIST with loss defense.

Method	**TGDA (w/wo defense)**			**pGAN(w/wo defense)**		
Target Model	LR	NN	CNN	LR	NN	CNN
Accuracy Drop (%)	2.79/2.56	1.50/1.49	1.11/1.104	2.52/2.49	1.09/1.07	0.74/0.73

C Additional Experiments

C.1 Experimental Settings

Hardware and package: Experiments were run on a cluster with `T4` and `P100` GPUs. The platform we use is PyTorch. Specifically, autodiff can be easily implemented using `torch.autograd`. As for the total gradient calculation, we follow [37] and apply conjugate gradient for calculating Hessian-vector products.

Dataset: We consider image classification on the MNIST [5] (60,000 training and 10,000 test images), and CIFAR-10 [20] (50,000 training and 10,000 test images) datasets. We are not aware of prior work that performs indiscriminate data poisoning on a dataset more complex than MNIST or CIFAR-10, and, as we will see, even these settings give rise to significant challenges in designing efficient and effective attacks. Indeed, some prior works consider only a simplified subset of MNIST (e.g., binary classification on 1's and 7's, or subsampling the training set to 1,000 points) or CIFAR-10 (e.g., binary classification on dogs and fish). In contrast, we set a benchmark by using the full datasets for multiclass classification.

Training and validation set: During the attack, we need to split the clean training data to the training set $\mathcal{D}_{tr}$ and validation set $\mathcal{D}_v$. Here we split the data to 70% training and 30% validation, respectively. Thus, for the MNIST dataset, we have $|\mathcal{D}_{tr}| = 42000$ and $|\mathcal{D}_v| = 18000$. For the CIFAR-10 dataset, we have $|\mathcal{D}_{tr}| = 35000$ and $|\mathcal{D}_v| = 15000$.

Attacker models and Defender models: (1) For the attacker model, for MNIST dataset: we use a three-layer neural network, with three fully connected layers and leaky ReLU activations; for CIFAR-10 dataset, we use an autoencoder with three convoluational layers and three conv transpose layers. The attacker takes the concatenation of the image and the label as the input, and generates the poisoned points. (2) For the defender, we examine three target models for MNIST: Logistic Regression, a neural network (NN) with three layers and a convolutional neural network (CNN) with two convolutional layers, maxpooling and one fully connected layer; and only the CNN model for CIFAR-10 (as CIFAR-10 contains RBG images).

Hyperparameters: (1) Pretrain: we use a batch size of 1,000 for MNIST and 256 for CIFAR-10, and optimize the network using our own implementation of gradient descent with `torch.autograd`. We choose the learning rate as 0.1 and train for 100 epochs. (2) Attack: for the attacker, we choose $\alpha = 0.01$, $m = 1$ by default; for the defender, we choose $\beta = 0.1$, $n = 20$ by default. We set the batch size to be 1,000 for MNIST; 256 for CIFAR10 and train for 200 epochs, where the attacker is updated using total gradient ascent and the defender is updated using gradient descent. We follow [37] and implement TGA using conjugate gradient. We choose the poisoning fraction $\epsilon = 3\%$ by default. Note that choosing a bigger ϵ will not increase our running time, but we choose a small ϵ to resemble the realistic setting in which the attacker is limited in their access to the training data. (3) Testing: we choose the exact same setting as pretrain to keep the defender's training scheme consistent.

Baselines: There is a spectrum of data poisoning attacks in the literature. However, due to their attack formulations, only a few attacks can be directly compared with our method. For instance, the Poison SVM [2] and KKT [19] attacks can only be applied to convex models for binary classification; the Min-max [19] and the Model targeted [32] attacks can be only applied to convex models. Thus we compare with two baseline methods that can attack neural networks: the Back-gradient attack [23] and the Label flip attack [2]. Moreover, it is also possible to apply certain targeted attack method (e.g., MetaPoison [17]) in the context of the indiscriminate attack. Thus we compare with MetaPoison on CIFAR-10 under our unified architecture. We follow [17] and choose $K = 2$ unrolled inner steps, 60 outer steps and an ensemble of 24 inner models.

Table 5: TGDA attack on MNIST with MaxUp defense.

Method	**TGDA (w/wo defense)**		
Target Model	LR	NN	CNN
Accuracy Drop (%)	2.79/2.77	1.50/1.50	1.11/1.11

C.2 Against Defenses:

To further evaluate the robustness of TGDA attack against data sanitization algorithms:

(1) We perform the loss defense [19] by removing 3% of training points with the largest loss. We compare with pGAN [24], which include constraint on the similarity between clean and poisoned sample thus inherently robust against defenses. In Table 4, we observe that although we do not add explicit constraint on detectability in our loss function, our method still reaches comparable robustness against such defenses with pGAN.

(2) We examine the robustness of our TGDA attack against strong data augmentations, e.g., the MaxUp defense[3] [13]. In a nutshell, MaxUp generates a set of augmented data with random perturbations and then aims at minimizing the worst case loss over the augmented data. Such training technique addresses overfitting and serves as a valid defense against adversarial examples. However, it is not clear if MaxUp is a good defense against indiscriminate data poisoning attacks. Thus, we implement MaxUp under our testing protocol, where we add random perturbations to the training and the poisoned data, i.e., $\{\mathcal{D}_{tr} \cup \mathcal{D}_p\}$, and then minimize the worst case loss over the augmented set. We report the results in Table 5, where we observe that even though MaxUp is a good defense against adversarial examples, it is not readily an effective defense against indiscriminate data poisoning attacks. Part of the reason we believe is that in our formulation the attacker anticipates the retraining done by the defender, in contrast to the adversarial example setting.

[3]We follow implementation in `https://github.com/Yunodo/maxup`