Trustworthy Machine Learning with Data in the Wild

by

Yiwei Lu

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2025

© Yiwei Lu 2025

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner: Sébastien Gambs
Professor, Computer Science
University of Quebec in Montreal

Supervisors: Yaoliang Yu
Associate Professor, Computer Science
University of Waterloo

Sun Sun
Senior Research Officer
National Research Council, Canada

Internal Members: Gautam Kamath
Assistant Professor, Computer Science
University of Waterloo

Hongyang Zhang
Assistant Professor, Computer Science
University of Waterloo

Internal-External Member: Elliot Creager
Assistant Professor, Electrical and Computer Engineering
University of Waterloo

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

This thesis presents my original research work, synthesized from four published papers: three conference articles and one journal article, as listed below. As the *first author* on all four publications, I was primarily responsible for contributing the core ideas, implementing the proposed methods, and drafting the manuscripts. My co-authors include my supervisors, who provided invaluable guidance in brainstorming, refining ideas, and polishing my writing, as well as close collaborators and undergraduate research associates under my supervision who contributed to various aspects of the work.

- Y. Lu, G. Kamath, and Y. Yu (2022). "Indiscriminate Data Poisoning Attacks on Neural Networks". *Transactions on Machine Learning Research*, which corresponds to Chapter 2.
- Y. Lu, G. Kamath, and Y. Yu (2023). "Exploring the limits of model-targeted indiscriminate data poisoning attacks". In: *International Conference on Machine Learning*. PMLR, which corresponds to Chapter 3.
- Y. Lu, M. Y. Yang, G. Kamath, and Y. Yu (2024). "Indiscriminate Data Poisoning Attacks on Pre-trained Feature Extractors". In: *Secure and Trustworthy Machine Learning*. IEEE, which corresponds to Chapter 4.
- Y. Lu, M. Y. Yang, Z. Liu, G. Kamath, and Y. Yu (2024). "Disguised Copyright Infringement of Latent Diffusion Model". In: *International Conference on Machine Learning*. PMLR, which corresponds to Chapter 5.

Other projects during my PhD During my PhD studies, I have published several papers that, while not directly incorporated into this thesis, cover topics that run parallel to my central research focus. These publications provide key insights into my broader research vision and deserve acknowledgment here (sorted by publication date):

- Y. Lu, G. Zhang, S. Sun, H. Guo, and Y. Yu (2023). "f-MICL: Understanding and Generalizing InfoNCE-based Contrastive Learning". *Transactions on Machine Learning Research*.
- Y. Lu, Y. Yu, X. Li, and V. Partovi Nia (2024). "Understanding Neural Network Binarization with Forward and Backward Proximal Quantizers". In: *Advances in Neural Information Processing Systems*. Vol. 36.
- M. Pawelczyk, J. Z. Di, Y. Lu, G. Kamath, A. Sekhari, and S. Neel (2025). "Machine Unlearning Fails to Remove Data Poisoning Attacks". In: *International Conference on Learning Representations*.

Abstract

Recent advances in machine learning (ML) have been largely fueled by models trained on extensive internet-collected datasets. While this approach has yielded remarkable capabilities, it introduces critical vulnerabilities: training data can be untrustworthy, containing harmful content or becoming susceptible to *data poisoning attacks*. In such scenarios, model behavior can be maliciously altered, resulting in reduced test performance for classification models or the replication of copyrighted materials for generative models. This thesis examines the influence of untrusted training data on machine learning training dynamics through two crucial perspectives: the ML developer's lens, focusing on model integrity, and the data owner's viewpoint, addressing privacy and copyright concerns.

Specifically, this thesis analyzes the impact of data in the wild from both theoretical and empirical perspectives. The first part formulates data poisoning attacks (specifically, accuracy degradation attacks) as bi-level optimization problems, also known as Stackelberg games. It provides a viable algorithm to poison modern machine learning models, particularly neural networks, which demonstrate significantly greater robustness to such attacks compared to traditional linear models.

The second part investigates this robustness distinction and develops a principled theoretical framework for understanding the effectiveness boundaries of data poisoning attacks across various scenarios. Given some clean training data, a target model, and malicious parameter objectives, this theoretical tool determines the minimum amount of poisoned data required to achieve these parameters, thereby quantifying the fundamental limits of data poisoning attacks.

Building upon the understanding of data poisoning attacks in supervised settings (i.e., classification tasks), this thesis further examines their threats in two realistic machine learning pipelines. The third part presents the first comprehensive analysis of data poisoning attacks against pre-trained feature extractors—components frequently utilized for various downstream ML tasks, such as adapting large models to medical data. This analysis reveals that drastic domain shifts can significantly increase ML models' vulnerability to data poisoning attacks, necessitating more robust countermeasures.

The final section examines the role of harmful data in generative models, specifically focusing on advanced latent diffusion models for text-to-image generation tasks. Copyright infringement concerns arise when such models produce outputs substantially similar to copyrighted training data. This section introduces a novel scenario termed "disguised copyright infringement" incurred by targeted data poisoning attacks, providing a thorough description of potential attack vectors and corresponding defensive strategies.

Acknowledgements

Five years ago, as I drove along seemingly endless roads toward an unfamiliar city named Waterloo, I carried nothing but a brave heart, a craving for knowledge, and sincere hope for a bright future. Now standing at the journey's end, I find myself gratified to have achieved the goals I set out to accomplish five years ago. Yet I never anticipated how winding this road would be, nor the remarkable vistas I would discover along the way.

Beginning my PhD during the pandemic, I found myself alone at the starting point of this journey: no peers, no classmates, no friends. Luckily, the cloudy road saw its first sunlight at the pandemic's end in 2022, with the publication of my first academic paper, now the opening chapter of this thesis. The momentum that began during those darkest times carried forward, and I was fortunate to achieve several more milestones in the three years that followed.

However, when I look back, I realize I was never actually alone. There were so many hands supporting me. They appeared in countless forms: thoughtful Slack messages, warm conversations in a pub, comforting family calls every Friday night, cups of handmade iced lattes, and gentle furry hugs. It is impossible to acknowledge all these supports on a single page, but I shall endeavor to express my gratitude to those who made this journey possible.

There are people who served as lightkeepers on this journey, whose guidance extended far beyond mere supervision. Prof. Yaoliang Yu was the master who demonstrated every technique, the coach who handed me water and towels during every break, and the first person to leap into the ring after each victory. Countless times I approached him in defeat, nearly ready to quit, only to walk out of his office with a confident smile and renewed determination. Dr. Sun Sun provided me with the warmest guidance during my darkest times, shared in my disappointments, and stood firmly beside me when confronting challenges. Prof. Gautam Kamath, though not my official supervisor, was present at every turn in my journey, serving as a tutor, a leader, and a friend who helped me navigate every major decision. He not only showed me how to be a good researcher but also exemplified how to approach work with passion, treat others with kindness, and embrace life with a spirit of adventure.

I was also extremely fortunate to receive invaluable guidance from numerous professors and tutors throughout my doctoral journey. I extend my sincere gratitude to my thesis committee members: Prof. Hongyang Zhang, Prof. Elliot Creager, and Prof. Sébastien Gambs, whose insightful feedback and thoughtful questions significantly strengthened this work. I am equally indebted to Prof. Florian Kerschbaum, Dr. Hongyu Guo, Dr. Vahid Partovi Nia, Prof. Dan Brown, Prof. Xi He, Prof. Seth Neel, Prof. Freda Shi, Prof. Victor Zhong, Prof. Adam Dziedzic, and Prof. Franziska Boenisch, who, though not on my committee, generously shared their expertise and provided meaningful guidance at critical junctures.

I am also profoundly grateful to my peers: Yihan Wang, Haoye Lu, Dihong Jiang, Guojun Zhang, Jimmy Di, Vasisht Duddu, Shufan Zhang, Jingjing Wang, Zeou Hu, Spencer

Szabados, Saber Malekmohammadi, and Martin Pawelczyk. I am also extremely fortunate to work with several amazing undergraduate researchers: Matthew Y.R. Yang, Robert (Zuoqiu) Liu, and William Xu. Their collaborative spirit during research discussions, their understanding during moments of struggle, and their shared excitement in times of breakthrough created a community of support that transformed solitary work into a collective endeavor.

Support also flowed from across the North American continent and across the Pacific, from my beloved hometown. There is a cherished Chinese saying, "When parents are present, one should not travel far". I have carried the weight of not honoring this wisdom, having been absent from my family for seven years. During our countless calls, I would often project strength, yet my mother could invariably sense my slightest emotional tremors. I know she silently bore rivers of tears when I struggled thousands of miles away. It was the foundation of resilience they cultivated within me that sustained my journey. I have been equally blessed by my parents-in-law, who, despite initially being strangers to me, entrusted me with their daughter and embraced me as their own son. Their unwavering prayers illuminated my path during the darkest moments and guided me toward steadfast purpose.

My deepest gratitude is reserved for the one who has been my constant companion, Yuan Ruan. She is the apple of my eye, the everlasting star in my sky, and the most beautiful soul I have ever encountered, my soulmate and truest friend. She has devoted her precious youth to walking beside me through bitter snowstorms, moments of quiet desolation, and times of profound struggle. Never once did she lament that our life together lacked fulfillment. Instead, she discovered joy in transforming our modest space into a home, in nurturing our daily existence with care, and in bestowing upon me the most precious gift this world has to offer: the sanctuary of family and unconditional love.

Finally, I must acknowledge my loyal companion, Hunter, who has been a source of unconditional joy throughout this journey. In the loneliest moments of academic solitude, he offered silent understanding without judgment, reminding me to occasionally step away from my desk and appreciate the simple pleasures of life. Those furry hugs mentioned earlier came from this faithful friend, whose playful spirit lightened even the most stressful days. The countless hours he spent curled at my feet while I worked on this thesis deserve recognition, for in those moments of quiet companionship, I found both comfort and clarity.

Dedication

To Yuan Ruan, who made every step worthwhile, and to the memory of Emily, whose tiny paws left indelible prints on our hearts.

Table of Contents

Examining Committee ii

Author's Declaration iii

Statement of Contributions iv

Abstract v

Acknowledgements vi

Dedication viii

List of Figures xiv

List of Tables xix

1 Introduction 1

1.1 Motivation 1

1.2 Trustworthy ML with Data in the Wild 2

1.2.1 The Scope 2

1.2.2 Fundamental Properties of Data Poisoning 3

1.2.3 Outsourced Data in Various ML Pipelines 5

1.3 Outline and Contributions 6

2 Indiscriminate Data Poisoning Attacks on Neural Networks 9

2.1 Preface 9

2.2 Introduction 9

2.3 Background . . . 11
2.3.1 Power of an attacker . . . 11
2.3.2 Objective of an attacker . . . 12
2.4 Total Gradient Descent Ascent Attack . . . 13
2.4.1 Preliminaries on Stackelberg Game . . . 13
2.4.2 On Data Poisoning Attacks . . . 14
2.5 Implementation . . . 17
2.5.1 Current Limitations . . . 17
2.5.2 A more efficient attack architecture . . . 19
2.6 Experiments . . . 21
2.6.1 Experimental Settings . . . 21
2.6.2 Comparison with Benchmarks . . . 22
2.6.3 Ablation Studies . . . 24
2.6.4 Visualization of attacks . . . 27
2.6.5 Transferability of the TGDA attack . . . 27
2.6.6 Against Defenses: . . . 28
2.7 Conclusions . . . 29

3 Exploring the Limits of Model-Targeted Indiscriminate Data Poisoning Attacks 31
3.1 Preface . . . 31
3.2 Introduction . . . 31
3.3 Background . . . 33
3.4 Theoretical Results . . . 34
3.4.1 Model Poisoning Reachability . . . 35
3.4.2 Scalar Output Linear Models . . . 36
3.4.3 Multiple Output Linear Models . . . 39
3.5 Gradient Canceling Attack . . . 40
3.6 Experiments . . . 43
3.6.1 Experimental Settings . . . 44
3.6.2 How Competitive Is Gradient Canceling (GC)? . . . 44
3.6.3 Predicting Poisoning Reachability Using τ . . . 45
3.6.4 Does GC Remain Effective Against Defenses? . . . 46
3.7 Conclusion and Future Work . . . 47

4 Indiscriminate Data Poisoning Attacks on Pre-trained Feature Extractors **49**

4.1 Preface . . . 49
4.2 Introduction . . . 50
4.3 Background . . . 52
4.4 Input Space Attacks . . . 53
4.4.1 Problem Setting . . . 53
4.4.2 TGDA input space attack . . . 54
4.4.3 GC input space attack . . . 55
4.4.4 UE (EMN) input space attack . . . 56
4.5 Feature Targeted Attack . . . 56
4.5.1 GC feature space attack . . . 58
4.5.2 Inverting features . . . 58
4.5.2.1 Decoder Inversion . . . 58
4.5.2.2 Feature Matching . . . 59
4.6 Experiments . . . 60
4.6.1 Experimental setting . . . 61
4.6.2 Fine-tuning . . . 62
4.6.2.1 Input space attacks . . . 62
4.6.2.2 Feature targeted attacks . . . 65
4.6.3 Transfer Learning . . . 66
4.6.3.1 Input space attacks . . . 66
4.6.3.2 Feature targeted attacks . . . 67
4.6.4 Unlearnable Examples . . . 68
4.7 Conclusions and Limitations . . . 68

5 Disguised Copyright Infringement of Latent Diffusion Models **71**

5.1 Preface . . . 71
5.2 Introduction . . . 72
5.3 Background . . . 74
5.3.1 Diffusion Models . . . 74

5.3.2 Data Poisoning attacks . . . 74
5.3.3 Textual Inversion . . . 75
5.4 Generating Disguises . . . 75
5.4.1 Feature Matching Attack . . . 75
5.4.2 Disguised Copyrighted Image Generation . . . 77
5.5 Revealing Disguises . . . 78
5.5.1 Experimental settings . . . 79
5.5.2 Concepts learned on disguises . . . 80
5.5.3 Visual appearance of disguises . . . 82
5.5.4 Detection . . . 83
5.5.5 A broader definition of access . . . 83
5.6 Conclusion . . . 84

6 Conclusion **86**

References **88**

APPENDICES **98**

A Appendix of Chapter 2 **99**
A.1 Indiscriminate data poisoning attacks . . . 99
A.1.1 Non-zero-sum setting . . . 99
A.1.2 Zero-sum Setting . . . 101
A.1.3 Zero-sum Setting with Target parameters . . . 101
A.1.4 Training-data-only attack . . . 103
A.2 Unlearnable Examples . . . 104
A.2.1 Stackelberg Game on Unlearnable Examples . . . 104
A.2.2 Comparison with Indiscriminate Data Poisoning Attacks . . . 105
A.3 Other solvers than TGDA . . . 106

B Appendix of Chapter 3 **108**
B.1 Proofs . . . 108
B.2 Data poisoning as measure optimization . . . 111
B.3 Additional Experiments . . . 112
B.3.1 Additional implementation details . . . 112
B.3.2 More on Parameter Corruption . . . 113
B.3.3 Least-square Regression . . . 113
B.3.4 Comparison with Gradient Matching . . . 114
B.3.5 More on Figure 3.3 . . . 115
B.3.6 Scaling **w** . . . 115
B.3.7 Simulating Different Target Parameters . . . 118
B.3.8 GC against Defenses . . . 118
B.3.9 Visualization of Poisoned Images . . . 120
B.3.10 Comparison with Replacing Attack . . . 120
B.4 Selecting Target Parameters . . . 122

C Appendix of Chapter 4 **123**
C.1 Perturbations introduced by feature matching . . . 123
C.2 Additional results on transfer learning . . . 123
C.2.1 Vision transformer as encoder . . . 123
C.2.2 Image encoder trained with CLIP . . . 124
C.2.3 Transferring to CIFAR-100 . . . 125
C.2.3.1 Transferring to PatchCamelyon . . . 125

List of Figures

2.1 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. 19

2.2 Our experimental protocol benchmarks data poisoning attacks. (1) Pretrain: the attacker and the defender are first trained on $\mathcal{D}_{tr}$ to yield a good autoencoder/classifier respectively. (2) During the attack, the attacker generates the optimal θ^* (thus $\mathcal{D}_p$) w.r.t $\mathcal{D}_v$ and the the optimal $\mathbf{w}^*$; the defender generates optimal $\mathbf{w}^*$ w.r.t $\mathcal{D}'_{tr} = \mathcal{D}_{tr} \cup \mathcal{D}_p$ and the optimal θ^* (which mimics testing). 20

2.3 We visualize the poisoned data generated by the TGDA attack with/without pretraining the leader L on the MNIST dataset. 24

2.4 Accuracy drop induced by our TGDA poisoning attack and baseline methods versus ε_d (left three figures: MNIST; right two figures: CIFAR-10). Attack efficacy increases modestly with ε_d. Note that when $\varepsilon_d = 1$, only 50% of the training set is filled with poisoned data. 25

2.5 We visualize the poisoned data generated by the TGDA attack and report the magnitude of perturbation (left: CIFAR-10; right: MNIST). 29

3.1 Logistic regression on the 2d OR dataset that verifies the transitioning threshold τ in Corollary 1. **Left**: τ w.r.t. target models $\mathbf{w} \in \mathbb{R}^2$, which all achieve 0 accuracy; **Middle**: accuracy drop due to the gradient canceling attack in Section 3.5. Indeed, poisoning successfully induces the target model $\mathbf{w}$ as long as $\varepsilon_d \geq \tau$; **Right**: norm of gradient w.r.t. model $\mathbf{w}$ over the mixed distribution χ, with ε_d the relative proportion of poisoned data. In general, the closer ε_d gets above τ, the smaller the gradient norm, which is an indication of the target model being more achievable through data poisoning. 37

3.2 We run experiments on logistic regression to verify the transition threshold τ in Corollary 1. **Left**: accuracy difference between GC and GradPC on 10-d Gaussian dataset; **Middle**: norm of the gradient over the mixed dataset χ on 10-d Gaussian dataset; **Right**: norm of the gradient over the mixed dataset χ on MNIST-17. 43

3.3 The learning curve for running GC on MNIST with different target models $\mathbf{w}$ and ε_w. We fix $\varepsilon_d = 1$, and the curves indicate the decrease of the gradient $\|\mathbf{g}(\chi)\|$ w.r.t. GC epoch. We confirm that GC fails to achieve $\mathbf{w}$ when $\varepsilon_d < \tau$. 45

4.1 An illustration of our threat model:(top row) we acquire the weights of a feature extractor f with contrastive learning methods and optimizing w.r.t. the InfoNCE loss; (bottom row) we inject poisoned samples to the training dataset on downstream applications (image classification in this example) to perturb the linear head only. We examine two scenarios in this chapter: (1) fine-tuning where pre-training and downstream tasks share the same training set; and (2) transfer learning where the downstream task is performed on a different dataset. 50

4.2 We visualize some clean training samples of CIFAR-10 (which serve as initialization to the attacks) in the first row, and poisoned samples generated by GC input space attacks (which induce an accuracy drop of 29.54%) for $\varepsilon_d = 0.03$ in the second row. The poisoned images show that GC input space attack generates images with no semantic meaning if no explicit constraints are posed. Clean images and their corresponding poisoned ones are chosen randomly. 53

4.3 An illustration of the three stages of feature-targeted attacks: (1) obtaining the target linear head parameter $\boldsymbol{\omega}$ with GradPC; (2) acquiring poisoned features ζ with GC feature-space attack; (3) invert ζ back to the input space using feature matching or decoder inversion (decoder inversion requires training an autoencoder with a fixed encoder f). 55

4.4 Here we visualize original test images (first row), and images reconstructed by an autoencoder with fixed ResNet-18 feature extractor learned by MoCo (second row), the same autoencoder trained end-to-end (third row), and the autoencoder with skip connections, i.e., a U-Net (fourth row). 57

4.5 Here we visualize clean images (first row), and poisoned samples returned by the feature matching algorithm with $\beta = 0.25, 0.1, 0.05$ respectively from the second to the fourth row. 63

4.6 The histogram of the (L_∞) magnitude of 1500 ($\varepsilon_d = 0.03$) poisoned points generated by feature matching feature matching on transfer learning (red) and fine-tuning (blue) with $\varepsilon_d = 0.03$ for $\beta = 0.25$ (left), $\beta = 0.1$ (right). The plot shows that while for fine-tuning the poisoned points are roughly in the same range, for transfer learning, $\approx 10\%$ of the poisoned samples are anomalies. 64

4.7 We plot poisoned images generated by feature matching on transfer learning for $\varepsilon_d = 0.03$. We show clean images and poisoned images for $\beta = 0.25$ and $\beta = 0.1$ on rows 1,2,3 respectively. Columns 1-4 are images within the normal range, and columns 5-8 are abnormal poisoned images with a bigger magnitude. 65

4.8 We plot poisoned images generated by EMN input space attack. We show clean images and poisoned images for fine-tuning (which incurs an accuracy drop of 7.55%) and transfer learning (incurring an accuracy drop of 7.88%) on rows 1,2,3, respectively. 69

5.1 An overview of conventional (with *direct access* to copyrighted material) and disguised (with *indirect access*) copyright infringement for latent diffusion models. For *direct access*, training an LDM-based model on copyrighted material x_c and reproducing x_c is subject to copyright infringement. For *indirect access*, one trains the same model on disguised samples x_d, which are drastically different from x_c, but are still able to reproduce x_c during inference. 72

5.2 An illustration on the algorithm to generate disguises. We aim to optimize the loss $\mathcal{L}$ consisting of an input space constraint that measures the distance between the base image x_b and the disguise x_d in the input space, and a feature space constraint that measures the distance between the copyrighted x_c and x_d in the feature space. 76

5.3 The disguised symbol on textual inversion. The first row from left to right: Column (1) the designated copyrighted symbol; Columns (2)-(5) four disguises x_d generated with different x_b. The second row: images generated by textual inversion after training on the above x_d. 78

5.4 **(a)** Disguised copyrighted content on textual inversion. The first row: the designated copyrighted image x_c (*The Sunflowers* by Vincent van Gogh); the second row: three disguises x_d generated with different x_b; the third row: images generated by textual inversion after training on the above x_d. **(b)** Disguised copyrighted style on textual inversion. The first row: the designated copyrighted style x_c (in the style of *The Starry Night* by Vincent van Gogh); the second row: disguises x_d generated with different x_b; the third row: images generated by textual inversion after training on the above x_d. 79

5.5 The effect of tuning the hyperparameter α on disguises (Rows 1-2: symbol; Rows 3-4: style). Rows 1 and 3 show disguises generated with different α; Rows 2 and 4 show the images generated by textual inversion by training on the disguises above. Note that the criterion on the input and feature threshold may not be satisfied in this series of experiments and we only train the disguise generation algorithm until convergence. 81

5.6 Results for the encoder-decoder examination. The first row shows the disguises x_d and the second row includes the output of the autoencoder: $\mathcal{D}(\mathcal{E}(x_d))$. Columns j-l, m-o, and p-r show results for symbol, content, and style, respectively. The autoencoder reveals the copyrighted information contained in the disguises. 82

B.1 Here we visualize the OR dataset. 113

B.2 Here we run the Gradient Canceling algorithm on linear regression on a 2D Gaussian dataset. The first row displays all figures in the same scale for better comparison; and the second row shows the upper figures in their original scale for better viewing. (1) The left five figures show the poisoned points generated with different ε_d. When ε_d is smaller, the poisoned points are farther from the data distribution. (2) The algorithm always generates the target parameter (the prediction) regardless of ε_d. 114

B.3 Here we run the Gradient Matching algorithm on linear regression on a 2D Gaussian dataset. The first row displays the poisoned points generated with different ε_d; and the second row shows the different target parameters generated by the algorithm with different ε_d. 114

B.4 The learning curve for running GC on MNIST with different target models and ε_d. Note that we fix ε_w for each model and print the respective τ, and the loss indicates $\|\mathbf{g}(\chi)\|$. The figure again confirms that GC cannot achieve $\mathbf{w}$ if $\varepsilon_d < \tau$. 115

B.5 We plot the training curves of GC on the toy example (see equation B.9). The colored curves represent GC attack on $\mathbf{w} = \begin{bmatrix} 0 & 2 \ln 2 \end{bmatrix}$ with $\varepsilon = 0.52 < \tau \approx 0.67$ under different learning rates; the black curve represents GC attack on the scaled target parameter $\mathbf{w}/1.1 \approx \begin{bmatrix} 0 & 1.82 \ln 2 \end{bmatrix}$ with $\varepsilon = 0.52 > \tau \approx 0.51$ under learning rate 1. 118

B.6 We visualize some poisoned images generated by the GC attack on the MNIST dataset. The first row shows the clean samples, the second row shows the poisoned samples; the third row displays the perturbation. . . . 120

B.7 We visualize some poisoned images generated by the GC attack on the CIFAR-10 dataset. The first row shows the clean samples, the second row shows the poisoned samples; the third row displays the perturbation. . . . 121

C.1 Here we visualize perturbations introduced by the feature matching algorithm with $\beta = 0.25, 0.1, 0.05$ respectively from the first to the third row. . 124

C.2 Example images from the PatchCamelyon dataset. 125

List of Tables

2.1 Summary of existing poisoning attack algorithms, evaluations, and their respective code. 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 (Biggio et al. 2011), P-SVM: PoisonSVM attack (Biggio et al. 2011), Min-max attack (Steinhardt et al. 2017), KKT attack (Koh, Steinhardt, et al. 2022), i-Min-max: improved Min-max attack (Koh, Steinhardt, et al. 2022), MT: Model Targeted attack (Suya et al. 2021), BG: Back-gradient attack (Muñoz-González, Biggio, et al. 2017). 18

2.2 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. We take three different runs for TGDA to get the mean and the standard deviation. Our attack outperforms the Min-max, i-Min-max, and Back-gradient attacks in terms of both effectiveness and efficiency across neural networks. 23

2.3 The attack accuracy/accuracy drop (%) and attack running time (hours) on CIFAR-10. Note that TGDA experiments are performed on 8 GPUs for parallel training. We take three different runs for TGDA and W. R. Huang et al. 2020 to get the mean and the standard deviation. 24

2.4 Comparing the TGDA attack with different orders: attacker as the leader and defender as the leader in terms of test accuracy/accuracy drop(%). Attacks are more effective when the attacker is the leader. 25

2.5 Comparing the TGDA attack with different formulations: non-zero-sum and zero-sum in terms of test accuracy/accuracy drop (%). Non-zero-sum is more effective at generating poisoning attacks. 25

2.6 Comparing the TGDA attack with/without pretraining the attacker L in terms of test accuracy/accuracy drop (%). Pretraining strongly improves attack efficacy. 26

2.7 Comparing different numbers of steps of the attacker (m) and defender (n) in terms of test accuracy/accuracy drop (%). Many attacker steps and a single defender step produce the most effective attacks. 26

2.8 Comparing the TGDA attack with partial warm-start (our original setting) and cold-start in terms of test accuracy/accuracy drop (%). Cold-start training is less effective overall. 27

2.9 Transferability experiments on MNIST. 27

2.10 Comparison with pGAN on MNIST with loss defense. 28

2.11 TGDA attack on MNIST with Influence/Sever/MaxUp defense. 28

3.1 The attack accuracy/accuracy drop (%) on MNIST. 33

3.2 The attack accuracy/accuracy drop (%) on MNIST, CIFAR-10 and Tiny-ImageNet. We perform GC based on the target parameters generated by GradPC. Our attack significantly outperforms TGDA and Gradient Matching. 43

3.3 Accuracy drop (%) of Gradient Canceling (w/wo clipping) on MNIST against Sever defense (+ indicates the accuracy increased by the defense). GC-c: GC with clipped output; GC-d: GC after defense; GC-cd: GC-c after defense. 46

3.4 The Certified Accuracy (CA) (%) of DPA and Accuracy drop (%) of Gradient Canceling ($\varepsilon_d = 0.008$) on MNIST against the DPA defense (+ indicates the accuracy increased by the defense). 46

4.1 Fine-tuning results: comparison between input space attacks (TGDA and GC input space attacks with/without constraints on poisoned samples) and feature targeted attacks (Decoder inversion and Feature Matching with different β) w.r.t. different ε_d. The white background indicates methods that generate target parameters or features; blue indicates input space attacks and pink indicates feature targeted attacks. 61

4.2 Transfer learning results: comparison between input space attacks (TGDA and GC input space attacks with/without constraints on poisoned samples) and feature targeted attacks (Decoder inversion and Feature Matching with different β) w.r.t. different ε_d. The white background indicates methods that generate target parameters or features; blue indicates input space attacks and pink indicates feature targeted attacks. 64

4.3 Comparison between feature matching and feature matching after removing anomaly points (feature matching-r). 67

4.4 Comparison between UE (EMN) in the context of poisoning a fixed feature extractor with linear head and training end-to-end (with the same initialization). We report the results on both fine-tuning and transfer learning. . 68

A.1 Indiscriminate data poisoning attacks: the attack accuracy/accuracy drop (%) and attack running time (hours) on CIFAR-10. 106

A.2 Unlearnable examples: model accuracy (%) under different unlearnable percentages on CIFAR-10 with ResNet-18 model. Percentage of unlearnable examples is defined as $\frac{|\mathcal{D}_p|}{|\mathcal{D}_{tr}+\mathcal{D}_p|}$. 106

B.1 The attack accuracy/accuracy drop (%) on the MNIST dataset. 113

B.2 The GC attack accuracy drop (%) on MNIST when scaling $\mathbf{w}$ by half. . . . 116

B.3 Simulating TGDA attack ($\varepsilon_d = 1$) with Gradient Canceling attack on the MNIST dataset. 119

B.4 The accuracy drop (%) Gradient Canceling attack (w/wo clipping) introduces on MNIST with Influence/Sever/MaxUp defense (+ indicates the accuracy increased by defenses). GC: original Gradient Canceling attack; GC-c: GC with clipped output; GC-d: GC after defense; GC-cd: GC-c after defense. 119

B.5 Gradient Canceling attack ($\varepsilon_d = 0.03$) with adding-only vs replacing-only on the MNIST dataset. 121

C.1 Transfer learning results on vision transformers: comparison between input space attacks (GC input space attacks with/without constraints on poisoned samples) and feature matching after removing anomaly points (feature matching-r) w.r.t. different ε_d. 123

C.2 Transfer learning results on CLIP image encoder: comparison between input space attacks (GC input space attacks with/without constraints on poisoned samples) and feature matching after removing anomaly points (feature matching-r). 124

C.3 Transfer learning results on CIFAR-100: comparison between input space attacks (GC input space attacks with/without constraints on poisoned samples) and feature matching after removing anomaly points (feature matching-r). 125

C.4 Transfer learning results on PatchCamelyon: comparison between input space attacks (GC input space attacks with/without constraints on poisoned samples) and feature matching after removing anomaly points (feature matching-r). 126

Chapter 1

Introduction

1.1 Motivation

Machine learning (ML) models have achieved remarkable success in recent years, especially in the domains of large language models (e.g., GPT-4 (Achiam et al. 2023), DeepSeek-V2 (A. Liu et al. 2024)), multimodal models (e.g., Gemini (Team et al. 2023)), and text-to-image generative models (e.g., Latent Diffusion (Rombach et al. 2022)). The exceptional performance of these models largely stems from the availability of vast training datasets, typically obtained by scraping extensive internet data using tools such as Common Crawl[1]. The advent of self-supervised learning methods such as CLIP (Radford et al. 2021) has enabled the efficient processing of extremely large training corpora for machine learning model development without the computational burden of intensive human annotation.

However, this seemingly innocuous data collection process harbors significant vulnerabilities. Adversaries can inject "poisoned data" into clean training sets to maliciously alter ML model behavior. Perhaps surprisingly, such attacks can be executed with alarming ease: an adversary can poison approximately 40,000 samples of LAION-5B (Schuhmann, Beaumont, et al. 2022)—a commonly used web-scale dataset that trains latent diffusion models (Rombach et al. 2022)—with a minimal budget of just 60 USD (Carlini, Jagielski, et al. 2024).

This vulnerability largely stems from the structural characteristics of web-scale datasets. For instance, to conserve storage space, LAION-5B contains domain names (URLs) rather than actual data (images). While efficient for storage, this design choice introduces a critical security flaw, as acknowledged by the dataset creators:

> "*The dataset is reliant on links to the World Wide Web. As such, we are unable to offer any guarantees of the existence of these samples. Due to the size we will also not be able to offer archives of the current state either.*"[2]

[1] `https://commoncrawl.org/`

[2] Quote from `https://laion.ai`

Consequently, domains in LAION-5B may expire over time. My independent investigation, involving random sampling of 50,000 images from the original dataset across eight iterations, revealed that approximately one-third of the links had expired on average. Adversaries can exploit this vulnerability by purchasing expired domains at minimal cost to inject arbitrary poisoned data. As a result, any subsequent client downloading this dataset after such an attack would receive a dataset substantially different from the one originally curated by data annotators.

Moreover, even without the presence of adversaries, such web-scale datasets present inherent ethical and legal problems. One notable concern is copyright infringement by text-to-image generative models. For instance, Latent Diffusion models have been found to incorporate artwork by human artists without permission and can produce substantially similar works during inference[3]. These incidents have affected numerous artists, resulting in the recent lawsuit *Anderson vs. Stability AI LTD* (Dist. Court 2023). Even more concerning, a recent investigation discovered that LAION-5B contained child sexual abuse material[4], leading to the dataset being completely removed from public access.

These problems are far from new. The formal definition of data poisoning attacks dates back to 2006 (Barreno et al. 2006), while the theoretical foundations of learning in the presence of malicious errors can be traced to 1988 (Kearns and M. Li 1988). The critical point is that these are long-standing technical challenges that appear ubiquitously across machine learning applications—from Microsoft's Twitter chatbot Tay[5] to Google's Gmail spam filter[6].

These issues pose fundamental challenges from two crucial perspectives. As ML developers, practitioners cannot confidently deploy trained models when data in the wild may compromise model integrity. Simultaneously, as data owners, we face legitimate concerns about unauthorized usage of our data by machine learning systems. The following section will outline how this thesis addresses these dual challenges through a comprehensive analytical framework.

1.2 Trustworthy ML with Data in the Wild

1.2.1 The Scope

Before addressing the aforementioned problems, I must explicitly define this thesis's scope. This work falls within the emerging field of "trustworthy ML," which broadly encompasses

[3] https://www.nytimes.com/2022/12/31/opinion/sarah-andersen-how-algorithim-took-my-work.html

[4] https://cyber.fsi.stanford.edu/news/investigation-finds-ai-image-generation-models-trained-child-abuse

[5] https://www.bbc.com/news/technology-35890188

[6] https://www.forbes.com/sites/naveenjoshi/2022/03/17/countering-the-underrated-threat-of-data-poisoning-facing-your-organization/?sh=1b634de3b5d8

the development and deployment of machine learning systems that satisfy key properties ensuring reliability, safety, and alignment with human values and societal norms. Trustworthiness in ML spans multiple dimensions, including robustness, privacy, safety, accountability, fairness, and transparency. It also intersects with various stages of ML pipelines, from data collection and training processes to downstream deployment and inference.

Rather than providing a surface-level examination of multiple trustworthiness aspects, this thesis deliberately focuses on one critical dimension: the impact of untrusted training data. Specifically, I investigate how data in the wild, particularly maliciously crafted training samples, affects the optimization of machine learning models and subsequently influences their behavior during inference. This focused approach directly addresses the challenges outlined in the previous section by providing a precise understanding of data impacts from both theoretical and empirical perspectives. The following paragraphs will elucidate these complementary dimensions of analysis.

1.2.2 Fundamental Properties of Data Poisoning

Untrusted data manifests in various forms, ranging from the copyrighted content and child sexual material mentioned previously to more subtle issues such as images with inaccurate annotations. Even meticulously curated datasets like ImageNet (J. Deng et al. 2009) contain mislabeled images. However, analyzing the impact of these diverse data problems proves challenging when approached randomly or in isolation. To systematically investigate their effects, we require a framework that characterizes the worst-case scenarios and encompasses these various issues as special cases. Fortunately, the concept of data poisoning (Barreno et al. 2006) provides precisely such a framework, allowing for comprehensive analysis of untrusted data's impact on machine learning systems. As defined by the Canadian Center for Cyber Security:

> **Data poisoning attack**: This type of attack occurs at a machine learning tool's training phase. AI tools rely heavily on accurate data for training. When poisoned (inaccurate) data is injected into the training data set, the poisoned data can lead the learning system to make mistakes[7].

Formulation Data poisoning attacks are classified into several categories based on the specific behaviors they induce in ML models. We provide a detailed categorization in Chapter 2. This thesis primarily investigates indiscriminate data poisoning attacks that aim to reduce the overall test accuracy of ML models, also known as availability attacks or accuracy-degrading attacks. Such attacks can be formulated as non-zero-sum bi-level optimization problems or Stackelberg games (Stackelberg 1934).

[7] `https://www.cyber.gc.ca/en/guidance/artificial-intelligence-itsap00040`

Difficulties While this formulation has been well-studied for linear settings (Biggio et al. 2011; Koh, Steinhardt, et al. 2022), the actual threat posed by indiscriminate attacks remains unclear for complex models such as deep neural networks—which dominate modern ML pipelines. The fundamental difficulty lies in solving the underlying Stackelberg game, where measuring the influence of poisoned data on trained models relies heavily on second-order information, particularly Hessian-inverse vector products. Such calculations scale poorly with increasing parameter counts, making their application to deeper models computationally prohibitive and consequently underexplored in current literature.

Data Poisoning against Neural Networks This thesis (specifically, Chapter 2) addresses this challenge by innovatively parameterizing the generation of poisoned data as a model, enabling the simultaneous creation of thousands of poisoned samples—a significant advancement over existing methods that generate poisoned points individually. Chapter 2 introduces the first efficient attack algorithms specifically designed for modern neural networks, facilitating the examination of data poisoning attacks against such models. However, our investigations reveal a surprising distinction between linear models and neural networks: neural networks are surprisingly robust to such attacks, with the test accuracy degradation consistently lower than the adversary's budget (concepts we will define precisely later). Moreover, this impact becomes especially limited when the poisoning set constitutes only a small fraction of the training data.

The observations above align with the conventional wisdom prevalent in the research community circa 2022 (when our paper (Lu, Kamath, et al. 2022), which forms the basis for Chapter 2, was published): that indiscriminate data poisoning attacks are essentially harmless to deep neural networks. However, this thesis argues that such conclusions may be premature, as several fundamental questions remain unresolved:

- Is the limited effectiveness of these attacks due to the inherent difficulty of the problem, or limitations in our optimization algorithms?
- Can we derive a complete theoretical characterization of the relationship between attack budget and effectiveness?
- How should we define the criteria for a "successful" attack in this context?
- What defensive strategies are appropriate against such attacks, and is the default approach of "doing nothing" actually sufficient?

Understanding the Limitations Consequently, Chapter 3 aims to provide answers to all of these questions by exploring the end goal of data poisoning attacks: changing model parameters *indirectly* through training on clean and poisoned data. Given this mindset, we can reformulate data poisoning into a two step procedure: (1) setting up the goal

parameters (i.e., a target) we would like to achieve[8]; (2) reaching this target parameter with data poisoning attacks.

This formulation provides exceptional analytical utility by: (1) rigorously defining attack success through a distance measure we term "reachability"; (2) enabling the design of a novel data poisoning attack without explicitly solving the bi-level optimization problem in Chapter 2, an attack that is significantly more efficient than previous methods and achieves state-of-the-art performance in effectiveness; and (3) facilitating exploration of the attack budget's role across different model architectures. Specifically, Chapter 3 introduces a novel theoretical framework revealing a surprising phenomenon: for marginal losses, reaching a target parameter is only possible when the attack budget exceeds an *attack threshold*—a value readily calculable using clean in-house data, target parameters, and the corresponding gradient. This attack threshold provides a simple yet complete characterization of poisoning attack difficulty and limitations.

1.2.3 Outsourced Data in Various ML Pipelines

While the preceding sections provide theoretical and empirical insights into indiscriminate data poisoning attacks, our analysis thus far has been somewhat restricted, focusing solely on classification models trained in an end-to-end supervised learning fashion. In modern machine learning pipelines, however, numerous other model architectures and training regimes are commonly employed. This thesis extends our investigation to two specific ML pipelines that represent significant contemporary applications.

Data Impact on Personalized Tasks Machine learning applications in critical domains such as medical imaging, 3D vision, and scientific AI frequently encounter challenges in collecting large-scale labeled datasets. The prevalent approach employs a two-stage process: (a) utilizing massive clean unlabeled data to train a general *feature extractor*, followed by (b) fine-tuning this extractor with limited in-domain data for transfer learning. Chapter 3 investigates the threat of indiscriminate data poisoning attacks on transfer learning scenarios where the feature extractor is obtained through self-supervised learning methods, particularly contrastive learning (Lu, G. Zhang, et al. 2023).

Expanding our investigation of indiscriminate data poisoning attacks, Chapter 4 examines the vulnerability of downstream tasks that employ pre-trained feature extractors. This chapter introduces two novel attack methodologies: (1) input space attacks, which adapt existing techniques to craft poisoned data directly in the input space, and (2) feature targeted attacks, which employ a three-stage approach to overcome optimization challenges: first acquiring target parameters for the linear head, then identifying poisoned features by treating learned representations as a dataset, and finally inverting these poisoned features

[8]We note that such target parameters can be generated by directly perturbing the clean model parameters, as we specify in Chapter 3.

back to the input space. Through comprehensive experiments examining these attacks in common downstream scenarios (fine-tuning on the same dataset and transfer learning with domain adaptation), we demonstrate that transfer learning scenarios exhibit significantly higher vulnerability. Our findings further reveal that while input space attacks pose substantial threats in the absence of protective measures, feature targeted attacks demonstrate superior effectiveness when countermeasures are implemented.

Copyright Infringement in Generative Models Finally, this thesis examines generative models, with particular focus on text-to-image generative models (specifically, latent diffusion models (Rombach et al. 2022)). As briefly introduced in the motivation section, these models have already raised concerns regarding copyright infringement through their capacity to memorize copyrighted training data and subsequently reproduce substantially similar material during inference when prompted with specific queries.

Chapter 5 reveals a potentially more serious threat to data owners through targeted data poisoning attacks, demonstrating how maliciously constructed samples can circumvent current copyright law protections. Specifically, we identify a concealed form of copyright infringement where an adversary can construct a "disguise", content that appears drastically different from the copyrighted material, yet still triggers the model to reproduce the copyrighted content during generation. This scenario represents a significant vulnerability currently overlooked by existing legal frameworks.

This research not only provides a novel perspective on copyright challenges in Generative AI but also offers an auditing methodology for detecting such disguised copyright violations. These contributions are particularly crucial for legal experts and policymakers tasked with addressing the emerging challenges posed by AI-generated content in copyright enforcement.

1.3 Outline and Contributions

Each of the remaining chapters is centered around a full-length article where I am the first author. The main contributions are summarized as follows:

In Chapter 2:

- 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.
- We summarize and classify existing data poisoning attacks (specifically, indiscriminate attacks) in both theoretical formulations and experimental settings.
- We propose an efficient attack architecture, which enables a more efficient clean-label attack.

- We conduct experiments to demonstrate the effectiveness of our attack on neural networks and its advantages over previous methods.

In Chapter 3:

- We formalize the notion of model poisoning reachability as a technical tool to study model-targeted data poisoning and we derive an easily computable threshold to characterize it.
- We quantify the critical role played by the poisoning ratio ε_d and we establish a surprising phase transition for ML classifiers, explaining seemingly disparate empirical results obtained with varying ε_d.
- We perform the Gradient Canceling attack on a number of models and datasets to extensively test our results. With carefully chosen target parameters, we are able to improve existing indiscriminate data poisoning baselines.

In Chapter 4:

- We expose the threat of indiscriminate data poisoning attacks on pre-trained feature extractors and set baselines on fine-tuning and transfer learning downstream tasks, where transfer learning is generally more vulnerable to the considered attacks;
- We tailor existing attacks (e.g., TGDA, Gradient Canceling attack, and Unlearnable Examples) to poisoning fixed future extractors and empirically identify the scenarios where they succeed (e.g., without constraints) and fail (with constraints);
- We propose new attacks called the *feature targeted* (FT) attack that involves three stages to alleviate the challenge of optimizing input space attacks with constraints and observe empirical improvement.
- Finally, we examine unlearnable examples in the context of poisoning fixed feature extractors and identify that a popular UE attack becomes much less effective on the downstream tasks.

In Chapter 5:

- We challenge the current "access" criterion (that copyrighted data is directly included in the training set) and point out its insufficiency in more delicate cases of copyright infringement;
- We propose an algorithm that demonstrably crafts disguised data to conceal the content (or concepts) of copyrighted images in the training set;

- We show disguised data contain copyrighted information in the latent space, such that by finetuning them on textual inversion or DreamBooth, or training on LDM, the model reproduces copyrighted data during inference;
- We propose methods to detect such disguises, which further encourage the expansion and quantification of "access" in the context of copyright infringement.

Chapter 2

Indiscriminate Data Poisoning Attacks on Neural Networks

2.1 Preface

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 for solving sequential Stackelberg games. 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 most 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. This chapter sets a new benchmark on the possibility of performing indiscriminate data poisoning attacks on modern neural networks.

The following sections are adapted from the main paper and appendix of Y. Lu, G. Kamath, and Y. Yu (2022). "Indiscriminate Data Poisoning Attacks on Neural Networks". *Transactions on Machine Learning Research.*

2.2 Introduction

Adversarial attacks have repeatedly exposed critical vulnerabilities in modern machine learning (ML) models (Kumar et al. 2020; Nelson et al. 2008; Szegedy et al. 2014). 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 chapter, 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 (Gao et al. 2020; Wakefield 2016). 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 (Farhadkhani et al. 2022; Lyu et al. 2020; Shejwalkar et al. 2022).

A spectrum of such data poisoning attacks exists in the literature, including *targeted*, *indiscriminate* and *backdoor* attacks. 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 (Biggio et al. 2012; Koh and Liang 2017; Koh, Steinhardt, et al. 2022; Shumailov et al. 2021), and several defenses have also been proposed (Diakonikolas, Kamath, Kane, J. Li, Steinhardt, et al. 2019; Steinhardt et al. 2017). 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 limitations. For example, many prior attacks generate poisoned points sequentially. Thus, when applied to deep models or large datasets, these attacks quickly become computationally infeasible. To our knowledge, a systematic analysis of indiscriminate data poisoning attacks on 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 non-zero-sum Stackelberg game, 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, (Evtushenko 1974; Fiez et al. 2020; G. Zhang et al. 2021)) algorithm to address the attacker's objective, even on non-convex models.

We also examine the effectiveness of alternative formulations, including the simpler zero-sum setting as well as when the defender leads the optimization. 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:

- 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.

- We summarize and classify existing data poisoning attacks (specifically, indiscriminate attacks) in both theoretical formulations and experimental settings.
- We propose an efficient attack architecture, which enables a more efficient clean-label attack.
- We conduct experiments to demonstrate the effectiveness of our attack on neural networks and its advantages over previous methods.

Notation. Throughout this chapter, we denote training data as $\mathcal{D}_{tr}$, validation data as $\mathcal{D}_v$, test data as $\mathcal{D}_{test}$, and poisoned data as $\mathcal{D}_p$. We use L to denote the leader in a Stackelberg game, ℓ for its loss function, $\mathbf{x}$ for its action, and θ for its model parameters (if they exist). Similarly, we use F to denote the follower, f for its loss function, and $\mathbf{w}$ for its model parameters. Finally, we use ε_d as the poisoning budget, namely that $|\mathcal{D}_p| = \varepsilon_d |\mathcal{D}_{tr}|$.

2.3 Background

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

2.3.1 Power of an attacker

Injecting poisoned samples. Normally, without breaching the defender'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. Such a situation may be realistic when the defender gathers data from several sources, some of which may be untrusted (e.g., when scraping data from the public Internet). The goal of the attacker can be presented as:

$$\mathbf{w}_* = \mathbf{w}_*(\mathcal{D}_p) \in \operatorname*{argmin}_{\mathbf{w}} \ \mathcal{L}(\mathcal{D}_{tr} \cup \mathcal{D}_p, \mathbf{w}),$$

where $\mathbf{w}_*$ is the attacker's desired model parameter, which realizes the attacker's objectives, and $\mathcal{L}(\cdot)$ is the training loss function (of the defender). We focus on such attacks and further categorize them in the next subsection.

Perturbing training data. Some work makes the assumption that the attacker can directly change the training data $\mathcal{D}_{tr}$. This is perhaps less realistic, as it assumes the attacker has compromised the defender's database. We note that this threat model may be more applicable in an alternate setting, where the defender wishes to prevent the data

from being used downstream to train a machine learning model. This research direction focuses on so called *unlearnable examples* (Fowl, Chiang, et al. 2021; Fowl, Goldblum, et al. 2021; H. Huang et al. 2021; D. Yu et al. 2022), and has faced some criticism that it provides "a false sense of security" (Radiya-Dixit et al. 2022). We provide more details of this line of research in Appendix A.2.

In this chapter, we focus on injecting poisoned samples as it is a more realistic attack.

2.3.2 Objective of an attacker

Data poisoning attacks can be further classified into three categories according to the adversary's objective (Cinà et al. 2022; Goldblum et al. 2023).

Targeted attack. The attacker adds poisoned data $\mathcal{D}_p$, resulting in a model $\mathbf{w}^*$ such that a particular target example from the test set is misclassified as the *base* class (Aghakhani et al. 2021; J. Guo and C. Liu 2020; Shafahi et al. 2018; Zhu et al. 2019). This topic is well studied in the literature, and we refer the reader to Schwarzschild et al. (2021) for an excellent summary of existing methods.

Backdoor attack. This attack aims at misclassifying any test input with a particular trigger pattern (X. Chen, C. Liu, et al. 2017; Gu et al. 2017; Saha et al. 2020; Tran et al. 2018). Note that backdoor attacks require access to both the training data as well as the input at inference time to plant the trigger.

Indiscriminate attack. This attack aims to induce a parameter vector $\mathbf{w}^*$ that broadly decreases the model utility. We consider image classification tasks where 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.
- Training-only attack: the attacker has access to training data $\mathcal{D}_{tr}$, the target model, and the training procedure (e.g., Biggio et al. 2012; Koh, Steinhardt, et al. 2022; Muñoz-González, Biggio, et al. 2017).
- Training-data-only attack: the attacker only has access to the training data $\mathcal{D}_{tr}$ (e.g., the label flip attack of Biggio et al. 2011).

In Appendix A.1 we give a more detailed summary of the existing indiscriminate data poisoning attacks.

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 2.6.

2.4 Total Gradient Descent Ascent Attack

In this section, we formulate the indiscriminate data poisoning problem and present our attack algorithm. We first briefly introduce the Stackelberg game and then link it to data poisoning.

2.4.1 Preliminaries on Stackelberg Game

The Stackelberg competition is a strategic game in Economics in which two parties move sequentially (Stackelberg 1934). Specifically, we consider two players, a leader L and a follower F in a Stackelberg game, where the follower F chooses $\mathbf{w}$ to best respond to the action $\mathbf{x}$ of the leader L, through minimizing its loss function f:

$$\forall \mathbf{x} \in \mathbb{X} \subseteq \mathbb{R}^d, \quad \mathbf{w}_*(\mathbf{x}) \in \operatorname*{argmin}_{\mathbf{w} \in \mathbb{W}} f(\mathbf{x}, \mathbf{w}),$$

and the leader L chooses $\mathbf{x}$ to maximize its objective function ℓ:

$$\mathbf{x}_* \in \operatorname*{argmax}_{\mathbf{x} \in \mathbb{X}} \ell(\mathbf{x}, \mathbf{w}_*(\mathbf{x})),$$

where $(\mathbf{x}_*, \mathbf{w}_*(\mathbf{x}_*))$ is known as a Stackelberg equilibrium. We note that an early work of W. Liu and Chawla (2009) already applied the Stackelberg game formulation to learning a linear discriminant function, where an adversary perturbs the whole training set first. In contrast, we consider the poisoning problem where the adversary can only add a small amount of poisoned data to the *unperturbed* training set. Moreover, instead of the genetic algorithm in W. Liu and Chawla (2009), we solve the resulting Stackelberg game using gradient algorithms that are more appropriate for neural network models. The follow-up work of W. Liu and Chawla (2010) further considered a constant-sum simplification, effectively crafting unlearnable examples (see more discussion in Appendix A.2) for support vector machines and logistic regression. Finally, we mention the early work of Dalvi et al. (2004), who essentially considered a game-theoretic formulation of adversarial training. However, the formulation of Dalvi et al. (2004) relied on the notion of Nash equilibrium where both players move simultaneously while in their implementation the attacker perturbs the whole training set w.r.t a fixed surrogate model (naive Bayes).

When $f = \ell$ we recover the zero-sum setting where the problem can be written compactly as:

$$\max_{\mathbf{x} \in \mathbb{X}} \min_{\mathbf{w} \in \mathbb{W}} \ell(\mathbf{x}, \mathbf{w}),$$

see, e.g., G. Zhang et al. (2021) and the references therein.

For simplicity, we assume $\mathbb{W} = \mathbb{R}^p$ and the functions f and ℓ are smooth, hence the follower problem is an instance of unconstrained smooth minimization.

2.4.2 On Data Poisoning Attacks

There are two possible ways to formulate data poisoning as a Stackelberg game, according to the acting order. Here we assume the attacker is the leader and acts first, and the defender is the follower. This assumption can be easily reversed such that the defender acts first. Both of these settings are realistic depending on the defender's awareness of data poisoning attacks. We will show in section 2.6 that the ordering of the two parties affects the results significantly.

Non-zero-sum formulation. In this section, we only consider the attacker as the leader as the other case is analogous. Here recall that the follower F (i.e., the defender) aims at minimizing its loss function $f = \mathcal{L}(\mathcal{D}_{tr} \cup \mathcal{D}_p, \mathbf{w})$ under data poisoning:

$$\mathbf{w}_* = \mathbf{w}_*(\mathcal{D}_p) \in \underset{\mathbf{w}}{\operatorname{argmin}} \ \mathcal{L}(\mathcal{D}_{tr} \cup \mathcal{D}_p, \mathbf{w}),$$

while the leader L (i.e., the attacker) aims at maximizing a different loss function $\ell = \mathcal{L}(\mathcal{D}_v, \mathbf{w}_*)$ on the validation set $\mathcal{D}_v$:

$$\mathcal{D}_{p_*} \in \underset{\mathcal{D}_p}{\operatorname{argmax}} \ \mathcal{L}(\mathcal{D}_v, \mathbf{w}_*),$$

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 following non-zero-sum Stackelberg formulation of data poisoning attacks (a.k.a., a bilevel optimization problem, see e.g. W. R. Huang et al. 2020; Koh, Steinhardt, et al. 2022; Muñoz-González, Biggio, et al. 2017):

$$\max_{\mathcal{D}_p} \ \mathcal{L}(\mathcal{D}_v, \mathbf{w}_*), \ \text{s.t.} \ \mathbf{w}_* \in \underset{\mathbf{w}}{\operatorname{argmin}} \ \mathcal{L}(\mathcal{D}_{tr} \cup \mathcal{D}_p, \mathbf{w}). \tag{2.1}$$

Note that we assume that the attacker can inject $\varepsilon_d N$ poisoned points, where $N = |\mathcal{D}_{tr}|$ and $\varepsilon_d \in [0, 1]$ is the power of the attacker, measured as a fraction of the training set size.

We identify that equation 2.1 is closely related to the formulation of unlearnable examples (Fowl, Chiang, et al. 2021; Fowl, Goldblum, et al. 2021; Fu et al. 2021; H. Huang et al. 2021; W. Liu and Chawla 2010; Sandoval-Segura et al. 2022; D. Yu et al. 2022):

$$\max_{\mathcal{D}_p} \ \mathcal{L}(\mathcal{D}_v, \mathbf{w}_*), \ \text{s.t.} \ \mathbf{w}_* \in \underset{\mathbf{w}}{\operatorname{argmin}} \ \mathcal{L}(\mathcal{D}_p, \mathbf{w}).$$

where $\mathcal{D}_p = \{(\mathbf{x}_i + \boldsymbol{\sigma}_i, y_i)\}_{i=1}^N$, with $\boldsymbol{\sigma}_i$ the bounded sample-wise or class-wise perturbation (i.e. $\|\boldsymbol{\sigma}_i\| \leq \varepsilon_d^\sigma$). The main differences lie in the direct modification of the training set $\mathcal{D}_{tr}$ (often all of it). In comparison, adding poisoned points to a clean training set would never result in 100 % modification in the augmented training set. This seemingly minor difference can cause a significant difference in algorithm design and performance. We direct interested readers to Appendix A.2 for details.

Previous approaches. Next, we mention three previous approaches for solving equation 2.1.

(1) A direct approach: 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 would result in zero gradients in practice*. 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 $\mathcal{D}_p$. Biggio et al. (2012) and Koh and Liang (2017) compute $\frac{\partial \mathbf{w}_*}{\partial \mathcal{D}_p}$ exactly via KKT conditions while Muñoz-González, Biggio, et al. (2017) approximate it using gradient ascent. We note that the back-gradient attack of Muñoz-González, Biggio, et al. (2017) can be understood as the k-step unrolled gradient descent ascent (UGDA) algorithm of Metz et al. (2017):

$$\begin{aligned} \mathbf{x}_{t+1} &= \mathbf{x}_t + \eta_t \nabla_{\mathbf{x}} \ell^{[k]}(\mathbf{x}_t, \mathbf{w}_t), \\ \mathbf{w}_{t+1} &= \mathbf{w}_t - \eta_t \nabla_{\mathbf{w}} f(\mathbf{x}_t, \mathbf{w}_t) \end{aligned}$$

where $\ell^{[k]}(\mathbf{x}, \mathbf{w})$ is the k-time composition, i.e., we perform k steps of gradient descent for the leader. Furthermore, W. R. Huang et al. (2020) propose to use a meta-learning algorithm for solving a similar bilevel optimization problem in targeted attack, and can be understood as running UGDA for M models and taking the average.

(2) Zero-sum reduction: Koh, Steinhardt, et al. (2022) also proposed a reduced problem of equation 2.1, where the leader and follower share the same loss function (i.e. $f = \ell$):

$$\max_{\mathcal{D}_p} \min_{\mathbf{w}} \mathcal{L}(\mathcal{D}_{tr} \cup \mathcal{D}_p, \mathbf{w}). \tag{2.2}$$

This relaxation enables attack algorithms to optimize the outer problem directly. However, this formulation may be problematic as its training objective does not necessarily reflect its true influence on test data. However, for unlearnable examples, the zero-sum reduction is feasible, and might be the only viable approach. See Appendix A.2 for more details.

This problem is addressed by Koh, Steinhardt, et al. (2022) with an assumption that the attacker can acquire a *target* model parameter, usually using a label flip attack which considers a much larger poisoning fraction ε_d. By adding a constraint involving the target parameter $\mathbf{w}_{tar}$, the attacker can search for poisoned points that maximize the loss ℓ while keeping a low loss on $\mathbf{w}_*^{tar}$. However, such target parameters are hard to obtain since, as we will demonstrate, non-convex models appear to be robust to label flip attacks and there are no guarantees that $\mathbf{w}_*^{tar}$ is the solution of equation 2.1.

(3) Fixed follower (model): Geiping et al. (2021) propose a gradient matching algorithm for crafting targeted poisoning attacks, which can also be easily adapted to *unlearnable examples* (Fowl, Chiang, et al. 2021). This method fixes the follower and supposes it

acquires clean parameter $\mathbf{w}$ on clean data $\mathcal{D}_{tr}$. We define a reversed function f', where f' can be the reversed cross entropy loss for classification problems (Fowl, Chiang, et al. 2021). As f' discourages the network from classifying clean samples, one can mimic its gradient $\nabla_{\mathbf{w}} f'(\mathbf{w}; \mathcal{D}_{tr})$ by adding poisoned data such that:

$$\nabla_{\mathbf{w}} f'(\mathbf{w}; \mathcal{D}_{tr}) \approx \nabla_{\mathbf{w}} f(\mathbf{w}; \mathcal{D}_{tr} \cup \mathcal{D}_p).$$

To accomplish this goal, Geiping et al. (2021) define a similarity function $\mathcal{S}$ for gradient matching, leading to the attack objective:

$$\mathcal{L} = S(\nabla_{\mathbf{w}} f'(\mathbf{w}; \mathcal{D}_{tr}), \nabla_{\mathbf{w}} f(\mathbf{w}; \mathcal{D}_{tr} \cup \mathcal{D}_p)),$$

where we minimize $\mathcal{L}$ w.r.t $\mathcal{D}_p$. This method is not studied yet in the indiscriminate attack literature, but would serve as an interesting future work.

TGDA attack. In this chapter, we solve equation 2.1 and avoid the calculation of $\frac{\partial \mathbf{w}_*}{\partial \mathcal{D}_p}$ using the Total gradient descent ascent (TGDA) algorithm (Evtushenko 1974; Fiez et al. 2020) [1]: TGDA takes a total gradient ascent step for the leader and a gradient descent step for the follower:

$$\begin{aligned} \mathbf{x}_{t+1} &= \mathbf{x}_t + \eta_t \mathsf{D}_{\mathbf{x}} \ell(\mathbf{x}_t, \mathbf{w}_t), \\ \mathbf{w}_{t+1} &= \mathbf{w}_t - \eta_t \nabla_{\mathbf{w}} f(\mathbf{x}_t, \mathbf{w}_t) \end{aligned} \tag{2.3}$$

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 $\mathcal{D}_p$. As optimizing ℓ does not involve the attacker parameter θ, we can rewrite $\mathsf{D}_{\mathbf{x}} := -\nabla_{\mathbf{wx}} f \cdot \nabla_{\mathbf{ww}}^{-1} f \cdot \nabla_{\mathbf{w}} \ell$. Here, the product $(\nabla_{\mathbf{ww}}^{-1} f \cdot \nabla_{\mathbf{w}} \ell)$ can be efficiently computed using conjugate gradient (CG) equipped with Hessian-vector products computed by auto-differentiation. As CG is essentially a *Hessian inverse-free approach* (Martens 2010), each step requires only linear time. Note that TGDA can also be treated as letting $k \to \infty$ in UGDA.

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. Therefore, we can rewrite equation 2.3 as:

$$\theta_{t+1} = \theta_t + \eta_t \mathsf{D}_\theta \ell(\theta_t, \mathbf{w}_t).$$

Thus, we have arrived at a poisoning attack that generates $\mathcal{D}_p$ in a batch rather than individually, which greatly improves the attack efficiency in Algorithm 1. Note that the TGA update does not depend on the choice of ε_d. This is a significant advantage over previous methods as the running time does not increase as the attacker is allowed a larger budget of introduced poisoned points, thus enabling data poisoning attacks on larger training sets.

[1] There are other possible solvers for equation 2.1, and we have listed them in Appendix A.3.

Algorithm 1: TGDA Attack

Input: Training set $\mathcal{D}_{tr} = \{x_i, y_i\}_{i=1}^N$, validation set $\mathcal{D}_v$, training steps T, attacker step size α, attacker number of steps m, defender step size β, defender number of steps n, poisoning fraction ε_d, L with θ_{pre} and $\ell = \mathcal{L}(\mathcal{D}_v, \mathbf{w}_*)$, F with $\mathbf{w}_{pre}$ and $f = \mathcal{L}(\mathcal{D}_{tr} \cup \mathcal{D}_p, \mathbf{w})$.

Initialize poisoned data set $\mathcal{D}_p^0 \longleftarrow \{(x'_1, y'_1), ..., (x'_{\varepsilon_d N}, y'_{\varepsilon_d N})\}$

for $t = 1, ..., T$ **do**

  for $i = 1, ..., m$ **do**

    $\theta \leftarrow \theta + \alpha \mathsf{D}_\theta \ell(\theta, \mathbf{w}_t)$ `// TGA on L`

  for $j = 1, ..., n$ **do**

    $\mathbf{w} \leftarrow \mathbf{w} - \beta \nabla_{\mathbf{w}} f(\theta, \mathbf{w})$ `// GD on F`

return model L_θ and poisoned set $\mathcal{D}_p = \mathsf{L}_\theta(\mathcal{D}_p^0)$

Necessity of Stackelberg game. Although equation 2.1 is equivalent to the bilevel optimization problem in W. R. Huang et al. (2020), Koh, Steinhardt, et al. (2022), and Muñoz-González, Biggio, et al. (2017), our sequential Stackelberg formulation is more suggestive of the data poisoning problem as it reveals the subtlety in the order of the attacker and the defender.

2.5 Implementation

In this section, we (1) discuss the limitations of existing data poisoning attacks and how to address them, (2) set an efficient attack architecture for the TGDA attack.

2.5.1 Current Limitations

We observe two limitations of existing data poisoning attacks.

Limitation 1: Inconsistent assumptions. We first summarize existing indiscriminate data poisoning attacks in Table 2.1, where we identify that such 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 somewhat unfair comparisons between methods.

Solution: 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 Section 2.3, we consider training-only attacks;

Table 2.1: Summary of existing poisoning attack algorithms, evaluations, and their respective code. 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 (Biggio et al. 2011), P-SVM: PoisonSVM attack (Biggio et al. 2011), Min-max attack (Steinhardt et al. 2017), KKT attack (Koh, Steinhardt, et al. 2022), i-Min-max: improved Min-max attack (Koh, Steinhardt, et al. 2022), MT: Model Targeted attack (Suya et al. 2021), BG: Back-gradient attack (Muñoz-González, Biggio, et al. 2017).

Attack	Dataset	Model	$\lvert\mathcal{D}_{tr}\rvert$	$\lvert\mathcal{D}_{test}\rvert$	ε_d	Code	Multiclass	Batch
Random label flip	toy	SVM	/	/	0-40%	git	✓	$\varepsilon_d\lvert\mathcal{D}_{tr}\rvert$
Adversarial label flip	toy	SVM	/	/	0-40%	git	×	$\varepsilon_d\lvert\mathcal{D}_{tr}\rvert$
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

(2) the attack formulation: in Section 2.4, we introduce three possible formulations, namely non-zero-sum, zero-sum, and zero-sum with target parameters. We will show in the experiment section that the latter two are ineffective against neural networks.

(3) 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_d of the training dataset. However, some works subsample and thus reduce the size of the training dataset. As we show in Figure 2.1, attack efficacy is *not* invariant to the size of the training set: larger training sets appear to be harder to poison. Furthermore, keeping ε_d 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_d, but also the complete training set for attacks.

Limitation 2: Running time. As discussed in Section 2.4, many existing attacks approach the problem by optimizing individual points directly, thus having to generate poisoned points one by one. Such implementation takes enormous running time (see Section 2.6) and does not scale to bigger models or datasets.

Solution: We design a new poisoning scheme that allows simultaneous and coordinated generation of $\mathcal{D}_p$ in batches requiring only one pass. Thanks to the TGDA attack in Section 2.4, we can treat L as a separate model (typically a neural network such as an autoencoder) that takes part of the $\mathcal{D}_{tr}$ as input and generates $\mathcal{D}_p$ correspondingly. Thus we fix the input and optimize only the parameters of L.

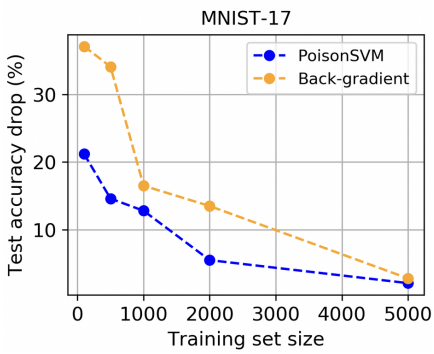


Figure 2.1: Comparing the efficacy of poisoning MNIST-17 with the PoisonSVM and Back-gradient attacks. The training set size is varied, while the ratio of the number of poisoned points to the training set size is fixed at 3%. These attacks become less effective as training set sizes increase.

2.5.2 A more efficient attack architecture

Once we have fixed the attack assumptions and poisoned data generation process, we are ready to specify the complete three-stage attack architecture, which enables us to compare poisoning attacks fairly. One can easily apply this unified framework for more advanced attacks in the future.

(1) Pretrain: The goals of the attacker L are to: (a) Reduce the test accuracy (i.e., successfully attack). (b) 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 $\mathbf{w}_{pre}$. Here we record the performance of F on $\mathcal{D}_{test}$ (denoted as $\mathtt{acc}_1$ for image classification tasks) as the benchmark we are poisoning.

(2) Attack: We generate poisoned points using the TGDA attack. We assume that the attacker can inject $\varepsilon_d N$ poisoned points, where $N = |\mathcal{D}_{tr}|$ and ε_d is the power of the attacker, measured as a fraction of the training set size. We summarize the attack procedure in Figure 2.2.

Initialization: We take the pretrained model L with parameter θ_{pre} and F with pre-trained 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 $\mathcal{D}_p^0 = \mathcal{D}_{tr}[0 : \varepsilon_d N]$.

TGDA attack: In this chapter, we run the TGDA attack to generate poisoned data. But it can be changed to any suitable attack for comparison.

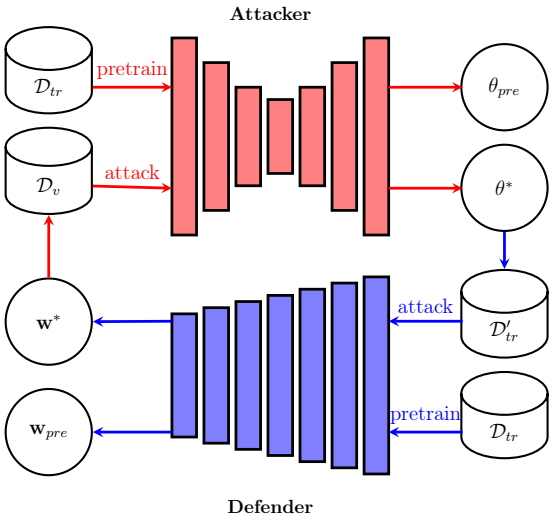


Figure 2.2: Our experimental protocol benchmarks data poisoning attacks. (1) Pretrain: the attacker and the defender are first trained on $\mathcal{D}_{tr}$ to yield a good autoencoder/classifier respectively. (2) During the attack, the attacker generates the optimal θ^* (thus $\mathcal{D}_p$) w.r.t $\mathcal{D}_v$ and the the optimal $\mathbf{w}^*$; the defender generates optimal $\mathbf{w}^*$ w.r.t $\mathcal{D}'_{tr} = \mathcal{D}_{tr} \cup \mathcal{D}_p$ and the optimal θ^* (which mimics testing).

Specifically, we follow Algorithm 1 and perform m steps of TGA updates for the attacker, and n steps of GD updates for the defender in one pass. We discuss the role of m and n in Section 2.6.

Note that previous works (e.g., Koh, Steinhardt, et al. 2022; Muñoz-González, Biggio, et al. 2017) choose $n = 1$ by default. However, we argue that this is not necessarily appropriate. When a system is deployed, the model is generally trained until convergence rather than for only a single step. Thus we recommend choosing a much larger n (e.g., $n = 20$ in our experiments) to better resemble the testing scenario.

Label Information: We specify that $\mathcal{D}_p^0 = \{x_i, y_i\}_{i=1}^{\varepsilon_d N}$. Prior works (e.g., Koh, Steinhardt, et al. 2022; Muñoz-González, Biggio, et al. 2017) optimize x to produce x_p, and perform a label flip on y to produce y_p (more details in Appendix A.1). This approach neglects label information during optimization.

In contrast, we **fix** $y_p = y$, and concatenate x and y to $\mathcal{D}_p^0 = \{x_i; y_i\}_{i=1}^{\varepsilon_d N}$ 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 can be categorized as clean label.

(3) 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.

Given the final θ, we produce the poisoned points $\mathcal{D}_p = \mathsf{L}_\theta(\mathcal{D}_p^0)$ and train F from

scratch on $\mathcal{D}_{tr} \cup \mathcal{D}_p$. Finally, we acquire the performance of F on $\mathcal{D}_{test}$ (denoted as $\texttt{acc}_2$ for image classification tasks). By comparing the discrepancy between pretrain and testing $\texttt{acc}_1 - \texttt{acc}_2$ we can evaluate the efficacy of an indiscriminate data poisoning attack.

2.6 Experiments

We evaluate our TGDA attack on various models for image classification tasks and show the efficacy of our method for poisoning neural networks. In comparison to existing indiscriminate data poisoning attacks, we show that our attack is superior in terms of both effectiveness and efficiency.

Specifically, our results confirm the following: (1) By applying the Stackelberg game formulation and incorporating second-order information, we can attack neural networks with improved efficiency and efficacy using the TGDA attack. (2) The efficient attack architecture further enables the TGDA attack to generate $\mathcal{D}_p$ in batches. (3) The poisoned points are visually similar to clean data, making the attack intuitively resistant to defenses.

2.6.1 Experimental Settings

Hardware and package: Experiments were run on a cluster with `T4` and `P100` GPUs. The platform we use is PyTorch (Paszke et al. 2019). Specifically, autodiff can be easily implemented using `torch.autograd`. As for the total gradient calculation, we follow G. Zhang et al. (2021) and apply conjugate gradient for calculating Hessian-vector products.

Dataset: We consider image classification on MNIST (L. Deng 2012) (60,000 training and 10,000 test images), and CIFAR-10 (Krizhevsky 2009) (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 into 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 and ResNet-18 (K. He, X. Zhang, et al. 2016) 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 G. Zhang et al. (2021) and implement TGA using conjugate gradient. We choose the poisoning fraction $\varepsilon_d = 3\%$ by default. Note that choosing a bigger ε_d will not increase our running time, but we choose a small ε_d 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. See Table 2.1 in Appendix A.1 for a complete summary. For instance, the Poison SVM (Biggio et al. 2011) and KKT (Koh, Steinhardt, et al. 2022) attacks can only be applied to convex models for binary classification; the Min-max (Steinhardt et al. 2017) and the Model targeted (Suya et al. 2021) attacks can only be applied to convex models. However, it is possible to modify Min-max (Steinhardt et al. 2017) and i-Min-max (Koh, Steinhardt, et al. 2022) attacks to attack neural networks. Moreover, we compare with two baseline methods that can originally attack neural networks: the Back-gradient attack (Muñoz-González, Biggio, et al. 2017) and the Label flip attack (Biggio et al. 2011). It is also possible to apply certain targeted attack methods (e.g., W. R. Huang et al. 2020) in the context of indiscriminate attacks. Thus we compare with HuangGFTG20 on CIFAR-10 under our unified architecture. We follow W. R. Huang et al. 2020 and choose $K = 2$ unrolled inner steps, 60 outer steps, and an ensemble of 24 inner models.

2.6.2 Comparison with Benchmarks

MNIST. We compare our attack with the Min-max, i-Min-max, Back-gradient and the Label flip attacks with $\varepsilon_d = 3\%$ on MNIST in Table 2.2. Since the Min-max, i-Min-max, and Back-gradient attack relies on generating poisoned points sequentially, 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 $y \leftarrow 10 - y$, as there are 10 classes in MNIST.

We observe that the label flip attack, though very efficient, is not effective against

Table 2.2: 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. We take three different runs for TGDA to get the mean and the standard deviation. Our attack outperforms the Min-max, i-Min-max, and Back-gradient attacks in terms of both effectiveness and efficiency across neural networks.

Model	Clean	Label Flip		Min-max		i-Min-max		BG		TGDA(ours)	
	Acc	Acc/Drop	Time	Acc/Drop	Time	Acc/Drop	Time	Acc/Drop	Time	Acc/Drop	Time
LR	92.35	90.83/1.52	0 hrs	89.80/ 2.55	0.7 hrs	89.56/**2.79**	19 hrs	89.82/2.53	27 hrs	89.56/$\mathbf{2.79}_{\pm 0.07}$	1.1 hrs
NN	98.04	97.99/0.05	0 hrs	98.07/-0.03	13.0 hrs	97.82/0.22	73 hrs	97.67/0.37	239 hrs	96.54/$\mathbf{1.50}_{\pm 0.02}$	15.0 hrs
CNN	99.13	99.12/0.01	0 hrs	99.55/-0.42	63.0 hrs	99.05/0.06	246 hrs	99.02/0.09	2153 hrs	98.02/$\mathbf{1.11}_{\pm 0.01}$	75.0 hrs

neural networks. Min-max attack, due to its zero-sum formulation, does not work on neural networks. i-Min-max attack is effective against LR, but performs poorly on neural networks where the assumption of convexity fails. Although Muñoz-González, Biggio, et al. (2017) show 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.

CIFAR-10. We compare our attack with the Label flip attack and the MetaPoison attack with $\varepsilon_d = 3\%$ on CIFAR-10 in Table 2.3. We omit comparison with the Back-gradient attack as it is too computationally expensive to run on CIFAR-10. We observe that running the TGDA attack following Algorithm 1 directly is computationally expensive on large models (e.g., ResNet, K. He, X. Zhang, et al. 2016). However, it is possible to run TGDA on such models by slightly changing Algorithm 1: we split the dataset into 8 partitions and run TGDA separately on different GPUs. This simple trick enables us to poison deeper models and we find it works well in practice. We observe that the TGDA attack is very effective at poisoning the CNN and the ResNet-18 architectures, Also, MetaPoison is a more efficient attack (meta-learning with two unrolled steps is much quicker than calculating the total gradient), but since its original objective is to perform targeted attacks, its application to indiscriminate attacks 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.

Table 2.3: The attack accuracy/accuracy drop (%) and attack running time (hours) on CIFAR-10. Note that TGDA experiments are performed on 8 GPUs for parallel training. We take three different runs for TGDA and W. R. Huang et al. 2020 to get the mean and the standard deviation.

Model	Clean	**Label Flip**		**MetaPoison**		**TGDA(ours)**	
	Acc	Acc/Drop	Time	Acc/Drop	Time	Acc/Drop	Time
CNN	69.44	68.99/0.45	0 hrs	$68.14/1.13_{\pm 0.12}$	35 hrs	$65.15/4.29_{\pm 0.09}$	42 hrs
ResNet-18	94.95	94.79/0.16	0 hrs	$92.90/2.05_{\pm 0.07}$	108 hrs	$89.41/5.54_{\pm 0.03}$	162 hrs

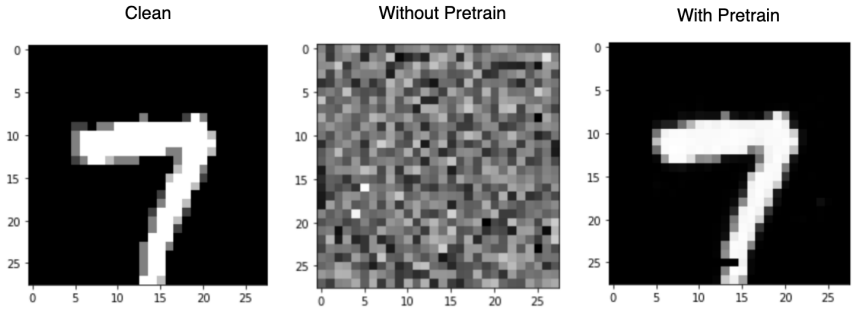


Figure 2.3: We visualize the poisoned data generated by the TGDA attack with/without pretraining the leader L on the MNIST dataset.

2.6.3 Ablation Studies

To better understand our TGDA attack, we perform ablation studies on the order in the Stackelberg game, the attack formulation, roles in our unified attack framework, and the choice of hyperparameters. For computational considerations, we run all ablation studies on the MNIST dataset unless specified. Furthermore, we include empirically comparison with *unlearnable examples* in Appendix A.2.

Who acts first. In Section 2.4, we assume that the attacker is the leader and the defender is the follower, i.e., that the attacker acts first. Here, we examine the outcome of reversing the order, where the defender acts first. Table 2.4 shows the comparison. We observe that across all models, reversing the order would result in a less effective attack. This result shows that even without any defense strategy, the target model would be more robust if the defender acts one step ahead of the attacker.

Attack formulation. In Section 2.4, we discuss a relaxed attack formulation, where $\ell = f$ and the game is zero-sum. We perform experiments on this setting and show results in Table 2.5. We observe that the non-zero-sum formulation is significantly more effective, and in some cases, the zero-sum setting actually *increases* the accuracy after poisoning. We also find that using target parameters would not work for neural networks as they are robust to label flip attacks even when ε_d is large. We ran a label flip attack with $\varepsilon_d = 100\%$ and observed only 0.1% and 0.07% accuracy drop on NN and CNN architectures, respectively.

Table 2.4: Comparing the TGDA attack with different orders: attacker as the leader and defender as the leader in terms of test accuracy/accuracy drop(%). Attacks are more effective when the attacker is the leader.

Target Model	Clean	Attacker as leader	Defender as leader
LR	92.35	89.56 / **2.79**	89.79 / 2.56
NN	98.04	96.54 / **1.50**	96.98 / 1.06
CNN	99.13	98.02 / **1.11**	98.66 / 0.47

Table 2.5: Comparing the TGDA attack with different formulations: non-zero-sum and zero-sum in terms of test accuracy/accuracy drop (%). Non-zero-sum is more effective at generating poisoning attacks.

Target Model	Clean	Non Zero-sum	Zero-sum
LR	92.35	89.56 / **2.79**	92.33 / 0.02
NN	98.04	96.54 / **1.50**	98.07 / -0.03
CNN	99.13	98.02 / **1.11**	99.55 / -0.42

This provides further evidence that neural networks are robust to massive label noise, as previously observed by Rolnick et al. (2017).

Role of pretraining. In Section 2.5, we propose two desired properties of L, among which L should generate $\mathcal{D}_p$ that is visually similar to $\mathcal{D}_{tr}$. Thus requires the pretraining of L for reconstructing images. We perform experiments without pretraining L to examine its role in effecting the attacker. Figure 2.3 confirms that without pretraining, the attacker will generate images that are visually different from the $\mathcal{D}_{tr}$ distribution, thus fragile to possible defenses. Moreover, Table 2.6 indicates that without pretraining L, the attack will also be ineffective. Thus we have demonstrated the necessity of the visual similarity between $\mathcal{D}_p$ and $\mathcal{D}_{tr}$.

Different ε_d. We have set $\varepsilon_d = 3\%$ in previous experiments. However, unlike prior

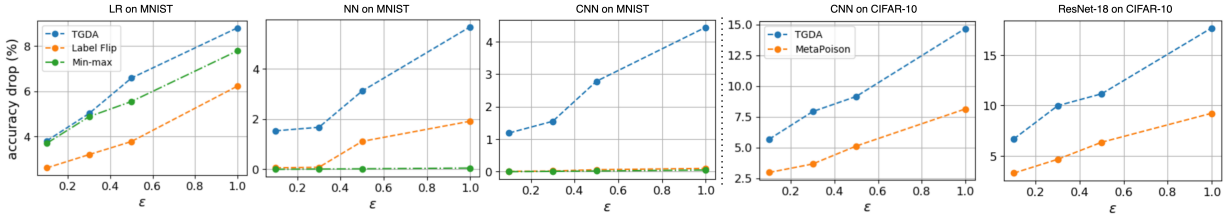


Figure 2.4: Accuracy drop induced by our TGDA poisoning attack and baseline methods versus ε_d (left three figures: MNIST; right two figures: CIFAR-10). Attack efficacy increases modestly with ε_d. Note that when $\varepsilon_d = 1$, only 50% of the training set is filled with poisoned data.

Table 2.6: Comparing the TGDA attack with/without pretraining the attacker L in terms of test accuracy/accuracy drop (%). Pretraining strongly improves attack efficacy.

Target Model	Clean	With Pretrain	Without Pretrain
LR	92.35	89.56 / **2.79**	92.09 / 0.26
NN	98.04	96.54 / **1.50**	97.47 / 0.57
CNN	99.13	98.02 / **1.11**	98.72 / 0.41

Table 2.7: Comparing different numbers of steps of the attacker (m) and defender (n) in terms of test accuracy/accuracy drop (%). Many attacker steps and a single defender step produce the most effective attacks.

Model	Clean	$m = 1, n = 0$	$m = 1, n = 1$	$m = 1, n = 20$	$m = 20, n = 1$	$m = n = 20$
LR	92.35	92.30 / 0.05	89.97 / 2.38	89.56 / 2.79	89.29 / **3.06**	89.77 / 2.57
NN	98.04	98.02 / 0.02	97.03 / 1.01	96.54 / 1.50	96.33 / **1.71**	96.85 / 1.19

methods which generate points one at a time, the running time of our attack does not scale with ε_d, and thus we can consider significantly larger ε_d and compare with other feasible methods. Figure 2.4 shows that attack efficacy increases with ε_d, but the accuracy drop is significantly less than ε_d when ε_d is very large. Moreover, TGDA outperforms baseline methods across any choice of ε_d.

Number of steps m and n. We discuss the choice of m and n, the number of steps of L and F, respectively. We perform three choices of m and n in Table 2.7. We observe that 20 steps of TGA and 1 step of GD results in the most effective attack. This indicates that when $m > n$, the outer maximization problem is better solved with more TGA updates. However, setting 4 ($m = 20, n = 1$) takes 10 times more computation than setting 3 ($m = 1, n = 20$), due to the fact that the TGA update is expensive. When $m = n = 1$, the attack is less effective as the defender might not be fully trained to respond to the attack. When $n = 0$, the attack is hardly effective at all as the target model is not retrained. We conclude that different choices of m and n would result in a trade-off between effectiveness and efficiency.

Cold-Start. The methods we compare in this work all belong to the partial warm-start category for bilevel optimization (Vicol et al. 2022). It is also possible to formulate the cold-start Stackelberg game for data poisoning. Specifically, we follow Vicol et al. 2022 such that in Algorithm 1, the follower update is modified to $\mathbf{w} \leftarrow \mathbf{w}_{pre} - \beta \nabla_{\mathbf{w}} f(\theta, \mathbf{w})$. We report the results in Table 2.8 on MNIST dataset. We observe that in indiscriminate data poisoning, partial warm-start is a better approach than cold-start overall, and our outer problem (autoencoder for generating poisoned points) does not appear to be highly over-parameterized.

Table 2.8: Comparing the TGDA attack with partial warm-start (our original setting) and cold-start in terms of test accuracy/accuracy drop (%). Cold-start training is less effective overall.

Model	Clean	Partial Warm-start	Cold-start
LR	92.35	89.56 / **2.79**	89.84 / 2.41
NN	98.04	96.54 / **1.50**	96.77 / 1.27
CNN	99.13	98.02 / **1.11**	98.33 / 0.80

Table 2.9: Transferability experiments on MNIST.

Surrogate	**LR**			**NN**			**CNN**		
Target	LR	NN	CNN	LR	NN	CNN	LR	NN	CNN
Accuracy Drop(%)	2.79	0.12	0.27	0.13	1.50	0.62	3.22	1.47	1.11

2.6.4 Visualization of attacks

Finally, we visualize some poisoned points $\mathcal{D}_p$ generated by the TGDA attack in Figure 2.5.

The poisoning samples against NN and CNN are visually very similar with $\mathcal{D}_{tr}$, as our attack is a clean label attack (see Section 2.5). Moreover, we evaluate the magnitude of perturbation by calculating the maximum of pixel-level difference. Both visual similarity and magnitude of perturbation provide heuristic evidence that the TGDA attack may be robust against data sanitization algorithms. Note that $\mathcal{D}_p$ against LR is visually distinguishable, and the reason behind this discrepancy between the convex model and the neural networks may be that the attacker L is not expressive enough to generate extremely strong poisoning attacks against neural networks.

2.6.5 Transferability of the TGDA attack

Even for training-only attacks, the assumption on the attacker's knowledge can be too strong. Thus we study the scenario when the attacker has limited knowledge regarding the defender's model F and training process, where the attacker has to use a surrogate model to simulate the defender. We report the transferability of the TGDA attack on different surrogate models in Table 2.9. We observe that poisoned points generated against LR and NN have a much lower impact against other models, while applying CNNs as the surrogate model is effective on all models.

Table 2.10: Comparison with pGAN on MNIST with loss defense.

Method	**TGDA (wo/w defense)**			**pGAN(wo/w defense)**		
Target Model	LR	NN	CNN	LR	NN	CNN
Accuracy Drop (%)	2.79/2.56	1.50/1.49	1.11/1.10	2.52/2.49	1.09/1.07	0.74/0.73

Table 2.11: TGDA attack on MNIST with Influence/Sever/MaxUp defense.

Model	Influence		Sever		MaxUp	
	wo defense	w defense	wo defense	w defense	wo defense	w defense
LR	2.79	2.45	2.79	2.13	2.79	2.77
NN	1.50	1.48	1.50	1.32	1.50	1.50
CNN	1.11	1.10	1.11	0.98	1.11	1.11

2.6.6 Against Defenses:

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

(a) We perform the loss defense (Koh, Steinhardt, et al. 2022) by removing 3% of training points with the largest loss. We compare with pGAN (Muñoz-González, Pfitzner, et al. 2019), which includes a constraint on the similarity between the clean and poisoned samples, and is thus inherently robust against defenses. In Table 2.10, we observe that although we do not add an explicit constraint on detectability in our loss function, our method still reaches comparable robustness against such defenses with pGAN.

(b) Other defenses remove suspicious points according to their effect on the learned parameters, e.g., through influence functions (influence defense in Koh and Liang 2017) or gradients (Sever in Diakonikolas, Kamath, Kane, J. Li, Steinhardt, et al. 2019). Specifically, influence defense removes 3% of training points with the highest influence, defined using their gradients and Hessian-vector products (Koh and Liang 2017); Sever removes 3% of training points with the highest outlier scores, defined using the top singular value in the matrix of gradients. Here we examine the robustness of TGDA against these two strong defenses. We observe in Table 2.11 that TGDA is robust against Influence defense, but its effectiveness is significantly reduced by Sever. Thus, we conclude Sever is potentially a good defense against the TGDA attack, and it might require special design (e.g., an explicit constraint on the singular value) to break Sever sanitation.

(c) We examine the robustness of our TGDA attack against strong data augmentations, e.g., the MaxUp defense[2] of Gong et al. 2021. 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. This training technique addresses overfitting and serves

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

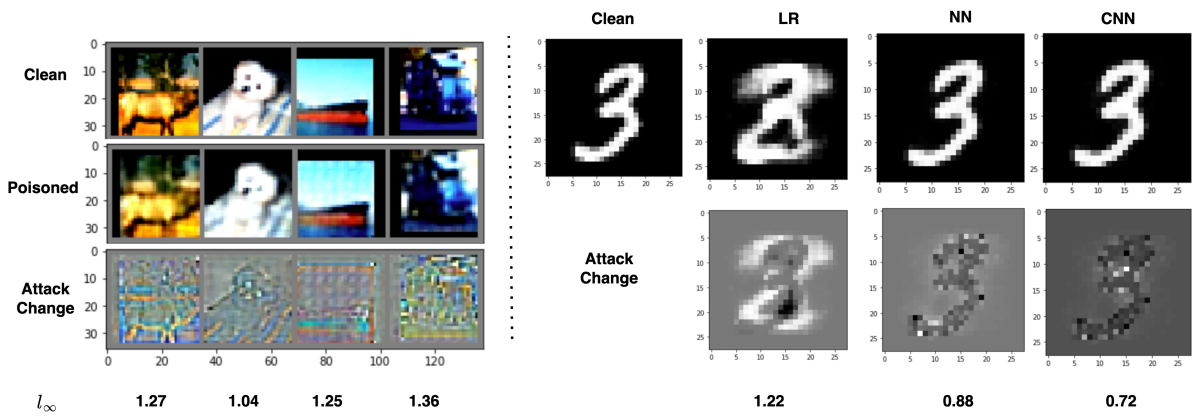


Figure 2.5: We visualize the poisoned data generated by the TGDA attack and report the magnitude of perturbation (left: CIFAR-10; right: MNIST).

as a possible 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 2.11, 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.

2.7 Conclusions

While indiscriminate data poisoning attacks have been well studied under various formulations and settings on convex models, non-convex models remain significantly underexplored. This chapter serves as a first exploration into poisoning neural networks under a unified architecture. While prior state-of-the-art attacks failed at this task due to either the attack formulation or a computationally prohibitive algorithm, we propose a novel Total Gradient Descent Ascent (TGDA) attack by exploiting second-order information, which enables generating thousands of poisoned points in only one pass. We perform experiments on (convolutional) neural networks and empirically demonstrate the feasibility of poisoning them. Moreover, the TGDA attack produces poisoned samples that are visually indistinguishable from unpoisoned data (i.e., it is a clean-label attack), which is desired in the presence of a curator who may attempt to sanitize the dataset.

Our work has some limitations. While our algorithm is significantly faster than prior methods, it remains computationally expensive to poison deeper models such as ResNet, or larger datasets such as ImageNet. Similarly, while our attacks are significantly more

effective than prior methods, we would ideally like a poison fraction of ε_d to induce an accuracy drop far larger than ε_d, as appears to be possible for simpler settings (Diakonikolas, Kamath, Kane, J. Li, Moitra, et al. 2016; Diakonikolas, Kamath, Kane, J. Li, Steinhardt, et al. 2019; Lai et al. 2016). This chapter sets an effective benchmark for later chapters on poisoning neural networks.

Chapter 3

Exploring the Limits of Model-Targeted Indiscriminate Data Poisoning Attacks

3.1 Preface

Abstract Indiscriminate data poisoning attacks aim to decrease a model's test accuracy by injecting a small amount of corrupted training data. Despite significant interest, existing attacks remain relatively ineffective against modern machine learning (ML) architectures. In this chapter, we introduce the notion of *model poisoning reachability* as a technical tool to explore the intrinsic limits of data poisoning attacks towards target parameters (i.e., model-targeted attacks). We derive an easily computable threshold to establish and quantify a surprising phase transition phenomenon among popular ML models: data poisoning attacks can achieve certain target parameters only when the poisoning ratio exceeds our threshold. Building on existing parameter corruption attacks and refining the Gradient Canceling attack, we perform extensive experiments to confirm our theoretical findings, test the predictability of our transition threshold, and significantly improve existing indiscriminate data poisoning baselines over a range of datasets and models. This chapter highlights the critical role played by the poisoning ratio, and sheds new insights on existing empirical results, attacks, and mitigation strategies in data poisoning.

The following sections are adapted from the main paper and appendix of Y. Lu, G. Kamath, and Y. Yu (2023). "Exploring the limits of model-targeted indiscriminate data poisoning attacks". In: *International Conference on Machine Learning*. PMLR.

3.2 Introduction

By now many data poisoning algorithms have been proposed; see Chapter 2 for some pointers. However, in the setting of indiscriminate data poisoning, where an attacker aims

to decrease the overall test accuracy by adding a small fraction of corrupted data, the effectiveness of existing attacks remains underwhelming. For example, the TGDA attack in Chapter 2 achieved 1.11% accuracy drop for a three-layer CNN on MNIST and a 5.54% accuracy drop for ResNet-18 on CIFAR-10, after adding $\varepsilon_d = 3\%$ poisoned data and retraining. Part of the difficulty lies in the computational challenge: the attacker has to anticipate what would happen after retraining the model on the mixed data (clean in-house data plus poisoned data). Other empirical works seem to suggest there might also be some intrinsic barrier to data poisoning; see Section 3.3 for a detailed discussion.

In this chapter we focus on model-targeted attacks (e.g., Koh, Steinhardt, et al. 2022; Suya et al. 2021) and introduce the notion of *model poisoning reachability*, i.e., given (arbitrary) clean training data, what model, represented by its parameter $\mathbf{w}$, can be achieved through data poisoning, and what is the minimum (relative) percentage ε_d of poisoned data that one has to introduce, with what algorithm? While model poisoning reachability intuitively depends on the clean training data, the loss and the target model we aim to achieve, we show that under mild conditions, it can be largely characterized by a simple threshold τ that is readily computable and involves no training at all. In particular, when the poisoning percentage ε_d falls under τ, no algorithm could achieve the target model by retraining on a mixed dataset (however crafted). On the flip side, if $\varepsilon_d > \tau$, we show that Gradient Canceling (GC), a refinement of the KKT attack of Koh, Steinhardt, et al. (2022), can achieve a given target model surprisingly efficiently. We further demonstrate that most ML classifiers exhibit a phase transition: they become poisoning reachable only when ε_d crosses the threshold τ. In contrast, regression methods can be poisoning reachable even when ε_d approaches 0. Thus, our results expose the critical role played by the poisoning percentage ε_d, and clarify the somewhat disparate empirical results in the literature (with varying ε_d).

Empirically, we apply the GC attack and verify the *model poisoning reachability* property across a wide range of ML models, from logistic regression to residual networks on various datasets. Moreover, our work can also be applied as a distillation device: given any target parameter (namely the teacher, however crafted or impractical) that is effective for certain purpose, we can use our threshold and GC attack to pinpoint the (minimum) amount of poisoning data that needs to be constructed in order to simulate the teacher through retraining the model (student) over the combination of clean and poisoned data. Indeed, using the target parameters generated by parameter corruption (Sun et al. 2020) as a teacher, GC is able to construct more practical and effective (student) data poisoning attacks than baseline methods.

We summarize our main contributions as follows:

- We formalize the notion of model poisoning reachability as a technical tool to study model-targeted data poisoning and we derive an easily computable threshold to characterize it.
- We quantify the critical role played by the poisoning ratio ε_d and we establish a surprising phase transition for ML classifiers, explaining seemingly disparate empirical results

Table 3.1: The attack accuracy/accuracy drop (%) on MNIST.

Model	Clean	**TGDA**	**GradPC**	
	Acc.	Accuracy/Drop	$\varepsilon_w = 0.5$	$\varepsilon_w = 1$
LR	92.35	89.56 / 2.79 ($\varepsilon_w = 2.45$)	69.80 / 22.55	**21.48** / **70.87**
NN	98.04	96.54 / 1.50 ($\varepsilon_w = 0.55$)	76.51 / 20.03	**31.14** / **66.90**
CNN	99.13	98.02 / 1.11 ($\varepsilon_w = 0.74$)	73.24 / 24.78	**12.98** / **86.15**

obtained with varying ε_d.

- We perform the Gradient Canceling attack on a number of models and datasets to extensively test our results. With carefully chosen target parameters, we are able to improve existing indiscriminate data poisoning baselines.

3.3 Background

While many algorithms have been proposed for data poisoning, their effectiveness remains largely underwhelming against neural networks, especially when ε_d, the relative proportion of poisoned data, is small. For example, Figure 2.4 in Chapter 2 and Table 2 of H. Huang et al. (2021) revealed that SOTA attacks can only decrease the test accuracy noticeably when ε_d is sufficiently (and sometimes exceedingly, e.g., $\varepsilon_d > 100\%$) large. These attacks, relying on sophisticated optimization tricks, are also rather expensive to run. On the other hand, any data poisoning attack amounts to an *indirect* way of rewiring an ML model (i.e., any change must be induced by retraining the model over clean and poisoned data). Direct approaches, such as the gradient-based parameter corruption (GradPC) attack of Sun et al. (2020) and Z. Zhang et al. (2021), seek to overwrite a target model *directly* (i.e., without constructing any poisoned data or retraining), under a perturbation constraint specified by ε_w, i.e., the relative change of the model parameter should be less than ε_w. While the applicability of direct approaches may seem limited, they are suitable for exploring the limits of more realistic data poisoning attacks.

In Table 3.1 we compare the performance of the direct approach GradPC (Sun et al. 2020) and the indirect approach TGDA in Chapter 2. The latter adds $\varepsilon_d = 3\%$ poisoned data while both attacks yield comparable perturbations of the (clean) model, as measured by ε_w. The difference is significant, and begs the obvious question: what caused this difference? Is it because existing data poisoning attacks are not sufficiently optimized yet, or is there some intrinsic barrier to produce certain target parameters through data poisoning? To what extent would increasing ε_d help, and how do we know without trying every ε_d? These questions will be formally and experimentally explored in the sequel, with the ultimate goal (if possible) to reduce the gap between data poisoning and parameter corruption attacks with comparable ε_w, as highlighted in Table 3.1.

Connection with Learning Theory: There has been significant work on training-time robustness in the learning theory literature, primarily focused on poisoning *worst-case*

distributions. Two models of robust PAC learning (Frénay and Verleysen 2013; Natarajan et al. 2013), slightly rephrased for the sake of comparison, include the malicious noise model, where the adversary adds points (e.g., Cesa-Bianchi et al. 1999; Kearns and M. Li 1988), and the nasty noise model, where the adversary may both add and remove points (e.g., Balcan et al. 2022; Bshouty et al. 2002). Many of these theoretical results show strong computational barriers against robust learning for even the most basic problems. Although our setting is similar to the malicious noise model (and we touch a bit on the nasty noise model in Appendix B.3.10), there are three major differences with the majority of the theory literature: (1) our attacks address distributions that arise in practice, which differ from worst-case distributions; (2) while other attacks flip labels, we consider "clean label" attacks which are not visibly mislabeled; (3) we focus on model-targeted attacks whose goal is to induce certain target parameters while the above-mentioned references focus directly on decreasing accuracy on the test sample.

3.4 Theoretical Results

In this section we formalize the notion of model poisoning reachability as a technical tool for studying model-targeted data poisoning. We further derive an easily computable threshold τ and reveal that model-targeted data poisoning attacks are effective only when ε_d, the (relative) percentage of poisoning data, crosses τ.

Notation and Preliminaries. Let $\ell(\mathbf{z}, \mathbf{w})$ be our loss that measures the fitness of our model $\mathbf{w}$ on data $\mathbf{z} \in \mathbb{Z}$, e.g., $\mathbf{z} = (\mathbf{x}, y)$ for supervised learning and $\mathbf{z} = \mathbf{x}$ for unsupervised learning. Let $\mathcal{P} = \mathcal{P}(\mathbb{Z})$ denote the set of all distributions on $\mathbb{Z}$, and we abstract the training data as an (empirical) distribution[1] $\mu \in \mathcal{P}$. For any given model $\mathbf{w}$ and training distribution μ, is it possible to construct a poisoning set, denoted by another (empirical) distribution ν, such that $\mathbf{w}$ minimizes ℓ over the mixed distribution $\chi = (1-\lambda)\mu + \lambda\nu$, where $\lambda = \frac{\varepsilon_d}{1+\varepsilon_d} \in [0, 1]$ is the proportion of poisoning data. To account for possible nonconvexity of the loss ℓ, we relax the optimality of a model $\mathbf{w}$ to simply have vanishing (sub)gradient. More formally, let

$$\mathbf{g}(\mathbf{z}) = \mathbf{g}(\mathbf{z}; \mathbf{w}) = \nabla_{\mathbf{w}} \ell(\mathbf{z}; \mathbf{w}) \tag{3.1}$$

be the gradient vector with respect to a fixed model $\mathbf{w}$ evaluated at the data $\mathbf{z}$. For practical reasons (e.g., to evade possible defenses or to account for the technical capabilities of an attacker) we also restrict the poisoning distribution ν into a convex subset $\Gamma \subseteq \mathcal{P}$ of admissible distributions. For instance, we may consider

$$\Gamma = \Gamma_{\mu,\delta} := \{\gamma : \|\gamma - \mu\| \leq \delta\},$$

[1]For convenience in this work we do not distinguish the (clean) training set from the training distribution, i.e., μ can be empirical.

where $\|\cdot\|$ denotes (say) the Wasserstein distance. By definition we always have $\mu \in \Gamma$. For each $\nu \in \Gamma$, define

$$\mathbf{g}(\nu) = \mathbf{g}(\nu; \mathbf{w}) := \mathbb{E}_{\mathbf{z}\sim\nu}\mathbf{g}(\mathbf{z}; \mathbf{w}),$$

i.e., the average gradient w.r.t. the distribution ν. Clearly,

$$\mathfrak{G} = \mathfrak{G}(\Gamma) := \{\mathbf{g}(\nu) : \nu \in \Gamma\} \tag{3.2}$$

is a subset of the closed convex hull of all gradient vectors. In fact, equality holds when $\Gamma = \mathcal{P}$ (e.g., $\delta = \infty$).

3.4.1 Model Poisoning Reachability

We can now state our fundamental problem of interest:

Definition 1 (Model Poisoning Reachability)**.** *We say a target parameter* $\mathbf{w}$ *is* $(\ell, \mu, \Gamma, \lambda)$*-poisoning reachable if there exists some poisoning distribution* $\nu \in \Gamma$ *such that*

$$\mathbf{g}(\chi; \mathbf{w}) = (1-\lambda)\mathbf{g}(\mu; \mathbf{w}) + \lambda\mathbf{g}(\nu; \mathbf{w}) = \mathbf{0},$$

i.e., the parameter $\mathbf{w}$ *has vanishing gradient (w.r.t. loss* ℓ*) over the mixed distribution* $\chi = (1-\lambda)\mu + \lambda\nu$*.*

When the loss ℓ, training distribution μ, and admissible poisoning distributions Γ are evident, we will simply say the parameter $\mathbf{w}$ is λ-poisoning reachable, or poisoning reachable if it is λ-poisoning reachable for some $\lambda \in [0, 1]$.

We make three further remarks regarding Definition 1: (a) If we are interested in more quantitative results about data poisoning, for example, is it possible to craft a poisoning set such that retraining on the mixed distribution would decrease test accuracy by a large margin, we need only specify a set of target models $\mathbf{w} \in \mathcal{W}$ that all decrease the test accuracy as required[2], and we say data poisoning is successful if any $\mathbf{w} \in \mathcal{W}$ is (λ-) poisoning reachable. (b) Definition 1 leaves out the computational aspects of data poisoning, i.e., how efficiently we can find such a poisoning distribution ν (whenever it exists). This will be studied in Section 3.5, using a gradient-based algorithm inspired directly by our definition. (c) We could also add other requirements, such as curvature or stability, to Definition 1.

Given the above formalization, the following characterization is immediate:

Theorem 1. *A target parameter* $\mathbf{w}$ *is* λ*-poisoning reachable iff* $\mathbf{0} \in \mathfrak{G}^\lambda = \mathfrak{G}^\lambda(\mathbf{g}(\mu)) := \{(1-\lambda)\mathbf{g}(\mu) + \lambda\mathbf{g} : \mathbf{g} \in \mathfrak{G}\}$.

[2] As pointed out by a reviewer, this may not be computationally feasible if one is too ambitious about the set $\mathcal{W}$.

Since $\mathfrak{G}$ (see equations equation 3.1-equation 3.2) is clearly convex, the subsets $\mathfrak{G}^\lambda$ are all convex and increasing with respect to λ, i.e.,

$$\mathbf{g}(\mu) = \mathfrak{G}^0 \subseteq \mathfrak{G}^\lambda \uparrow\subseteq \mathfrak{G}^1 = \mathfrak{G}.$$

Recall that $\lambda = \frac{\varepsilon_d}{1+\varepsilon_d}$ is the (absolute) proportion of the poisoned set. Thus, we conclude intuitively that the larger ε_d (equivalently λ) is, the easier it is to induce any target model $\mathbf{w}$ on any training distribution μ. In particular, the special case $\lambda = 1$ corresponds to the so-called "unlearnable examples" (Fowl, Chiang, et al. 2021; Fowl, Goldblum, et al. 2021; Fu et al. 2021; H. Huang et al. 2021; W. Liu and Chawla 2010; Sandoval-Segura et al. 2022; D. Yu et al. 2022), where an attacker is allowed to change the entire training set (i.e., empirical distribution μ).

Conversely, we can also conclude from Theorem 1 that if $\mathbf{0} \notin \mathfrak{G}(\Gamma)$, then data poisoning, with any budget ε_d, will not be successful in producing the target parameter $\mathbf{w}$. If $\mathbf{0} \notin \mathfrak{G}(\mathcal{P})$, then no training distribution can yield $\mathbf{w}$. In particular, data poisoning will not be successful in producing $\mathbf{w}$ even if $\varepsilon_d = \infty$.

Let us give some examples to illustrate our results so far.

Example 1 (Least-square regression)**.** *Consider*

$$\ell(\mathbf{z}; \mathbf{w}) = \tfrac{1}{2}(y - \mathbf{w}^\top \mathbf{x})^2, \text{ where } \mathbf{z} = (\mathbf{x}, y).$$

Clearly, we have

$$\mathbf{g}(\mathbf{x}, y) = (\mathbf{w}^\top \mathbf{x} - y)\mathbf{x} = (\mathbf{x}\mathbf{x}^\top)\mathbf{w} - y\mathbf{x},$$

and hence $\mathbf{g}(\mu) = \Sigma\mathbf{w} - \mathbf{m}$*, where* $\Sigma = \mathbb{E}_{\mathbf{x}\sim\mu}\mathbf{x}\mathbf{x}^\top$ *and* $\mathbf{m} = \mathbb{E}_{(\mathbf{x},y)\sim\mu} y\mathbf{x}$*. For simplicity let us assume* $\mathbb{Z} = \mathbb{R}^d \times \mathbb{R}$ *and* $\Gamma = \mathcal{P}$ *so that* $\mathfrak{G} = \mathbb{R}^d$ *(by considering product distributions where* $\mathbf{x}$ *concentrates on a single point). Therefore, we conclude from Theorem 1 that data poisoning with any* $\varepsilon_d > 0$ *is possible for least-square regression. The same conclusion holds even if we add regularization to* $\mathbf{w}$ *(which, we recall, is fixed).*

3.4.2 Scalar Output Linear Models

For linear models we can further simplify the iff condition in Theorem 1. We begin with the following result:

Theorem 2. *Suppose* $\Gamma = \mathcal{P}$ *contains all distributions,* $\ell((\mathbf{x}, y); \mathbf{w}) = l(\mathbf{w}^\top \mathbf{x}, y)$ *for some univariate loss* l*, and* $\langle \mathbf{w}, \mathbf{g}(\mu)\rangle \neq 0$*. Then,* $\mathbf{w}$ *is* λ*-poisoning reachable iff*

$$\lambda > \max\left\{\tfrac{\langle \mathbf{w},\mathbf{g}(\mu)\rangle}{\langle \mathbf{w},\mathbf{g}(\mu)\rangle - a}, \tfrac{-\langle \mathbf{w},\mathbf{g}(\mu)\rangle}{b - \langle \mathbf{w},\mathbf{g}(\mu)\rangle}\right\}, \quad \text{where} \tag{3.3}$$

$$a = \inf_{(\mathbf{x},y)\in\mathbb{Z}} (\mathbf{w}^\top \mathbf{x}) \cdot l'(\mathbf{w}^\top \mathbf{x}, y),$$

$$b = \sup_{(\mathbf{x},y)\in\mathbb{Z}} (\mathbf{w}^\top \mathbf{x}) \cdot l'(\mathbf{w}^\top \mathbf{x}, y),$$

with equality attained if the maximum is attained.

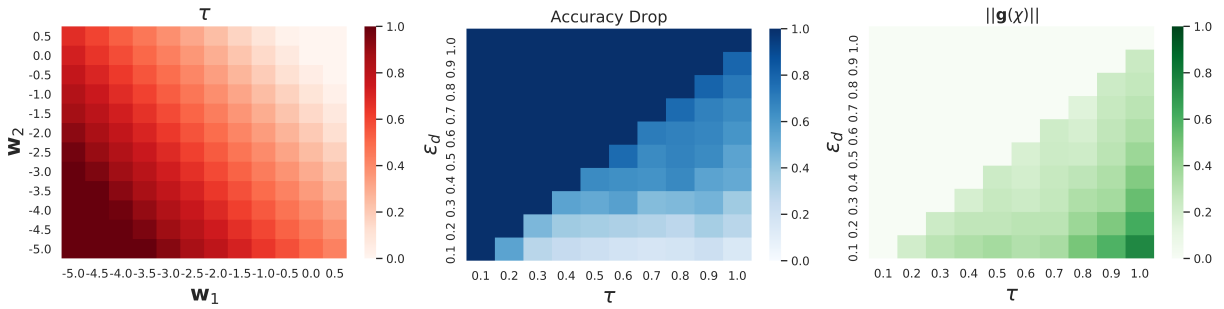


Figure 3.1: Logistic regression on the 2d OR dataset that verifies the transitioning threshold τ in Corollary 1. **Left**: τ w.r.t. target models $\mathbf{w} \in \mathbb{R}^2$, which all achieve 0 accuracy; **Middle**: accuracy drop due to the gradient canceling attack in Section 3.5. Indeed, poisoning successfully induces the target model $\mathbf{w}$ as long as $\varepsilon_d \geq \tau$; **Right**: norm of gradient w.r.t. model $\mathbf{w}$ over the mixed distribution χ, with ε_d the relative proportion of poisoned data. In general, the closer ε_d gets above τ, the smaller the gradient norm, which is an indication of the target model being more achievable through data poisoning.

Theorem 2 follows from the more general Theorem 5 in Appendix B.1, where we further remove the restriction $\Gamma = \mathcal{P}$. The condition $\langle \mathbf{w}, \mathbf{g}(\mu) \rangle \neq 0$ can be checked easily *a priori*; see Remark 1 (Appendix B.1) for discussions on when it fails. Remark 2 (Appendix B.1) draws further connection between our result and the breakdown point in robust statistics. Compared to the more general Theorem 1, Theorem 2 exploits the linear structure to simplify the set $\mathfrak{G}$ to basically an interval and hence the condition equation 3.3 is much easier to verify. Indeed, consider Example 1 again. It is clear that $l'(t, y) = t - y$, whence $a = -\infty$ and $b = \infty$. Thus, we verify more easily that data poisoning succeeds on least-square regression for any $\varepsilon_d > 0$.

The next example reveals a surprising phase transition in terms of the poisoning proportion λ (or equivalently ε_d):

Example 2 (Logistic regression)**.** *Consider now*

$$\ell(\mathbf{z}; \mathbf{w}) = \log(1 + \exp(-\mathbf{w}^\top \tilde{\mathbf{x}})),$$

where we have absorbed the binary label y into $\tilde{\mathbf{x}}$ (e.g., $\tilde{\mathbf{x}} \leftarrow y\mathbf{x}$). Clearly, we have $\mathbf{g}(\tilde{\mathbf{x}}) = -\frac{1}{1+\exp(\mathbf{w}^\top \tilde{\mathbf{x}})}\tilde{\mathbf{x}}$. On the direction $\mathbf{w}$, for any distribution μ we have

$$-\mathfrak{W}(\tfrac{1}{e}) = \inf_t \tfrac{-t}{1+\exp(t)} \leq \langle \mathbf{w}, \mathbf{g}(\mu) \rangle \leq \sup_t \tfrac{-t}{1+\exp(t)},$$

where the left-hand side is Lambert's W function and the right-hand side is clearly ∞. Therefore, suppose $\mathbb{X} = \mathbb{R}^d$ and $\Gamma = \mathcal{P}$, we have

$$\mathfrak{G} = \{\mathbf{g} : \mathbf{w}^\top \mathbf{g} \geq -\mathfrak{W}(1/e) \approx -0.28\},$$

which is not the entire space! Consequently, if

$$\lambda < \tfrac{\langle \mathbf{w}, \mathbf{g}(\mu) \rangle}{\langle \mathbf{w}, \mathbf{g}(\mu) \rangle + \mathfrak{W}(1/e)} \iff \varepsilon_d < \tau := \max\{\tfrac{\langle \mathbf{w}, \mathbf{g}(\mu) \rangle}{\mathfrak{W}(1/e)}, 0\}, \tag{3.4}$$

then any poisoning distribution ν (with any support) cannot produce $\mathbf{w}$ (along with training distribution μ)!

By simply changing $\tilde{\mathbf{x}} \leftarrow y\mathbf{x}$ and then dropping y we immediately obtain from Theorem 2 sufficient and necessary conditions for the poisoning reachability of binary margin classifiers. In particular, we record the following result:

Corollary 1 (Binary Margin Classifier)**.** *Consider linear models with loss $\ell(\tilde{\mathbf{x}}; \mathbf{w}) = l(\mathbf{w}^\top \tilde{\mathbf{x}})$.*

Suppose $\Gamma = \mathcal{P}$ consist of all distributions on $\tilde{\mathbb{X}}$ and $\langle \mathbf{w}, \mathbf{g}(\mu) \rangle \neq 0$. Define

$$a := \inf_{t \in \mathbf{w}^\top \tilde{\mathbb{X}}} t \cdot l'(t), \quad b := \sup_{t \in \mathbf{w}^\top \tilde{\mathbb{X}}} t \cdot l'(t).$$

Then, a target parameter $\mathbf{w}$ is λ-poisoning reachable iff equation 3.3 holds (with equality attained if the maximum there is attained).

The standard margin losses are decreasing, such as the logistic loss in Example 2, the exponential loss in Adaboost, and the hinge loss in SVM. When $\mathbb{X} = \mathbb{R}^d$ is unbounded, typically $b = \infty$ but $a > -\infty$, leading to a common phase transition phenomenon: data poisoning against these losses succeeds in producing a target parameter $\mathbf{w}$ iff λ crosses the threshold in equation 3.3. In particular, any target parameter $\mathbf{w}$ such that $\langle \mathbf{w}, \mathbf{g}(\mu) \rangle < 0$ is always poisoning reachable for any $\lambda > 0$. Interestingly, Koh, Steinhardt, et al. (2022, Proposition 3) showed that if a model is poisoning reachable, then it (often) can be poisoned to by a distribution ν supported on two distinct points (which however does not imply diminishing ε_d due to repetitions). Corollary 1 provides a definitive answer on when a model is poisoning reachable and hence complements the results of Koh, Steinhardt, et al. (2022).

We emphasize that with any further restrictions on the poisoning distribution (such that $\Gamma \subsetneq \mathcal{P}$), condition equation 3.3 remains to be necessary: data poisoning is apparently even harder in this case. For nonlinear models with a fixed feature map ϕ (such as kernel methods), our results extend immediately, after the obvious change-of-variable $\mathbf{x} \leftarrow \phi(\mathbf{x})$.

Figure 3.1 illustrates the transition threshold τ in equation 3.4 on the simple OR dataset (where each of the four points is repeated 50 times with small Gaussian perturbation, see Appendix B.3 for details). Logistic regression (LR), trained on the clean data, achieves perfect accuracy. In Figure 3.1 (left), each grid point represents a target parameter $\mathbf{w} = (w_1, w_2)$, all of which achieve 0 test accuracy (i.e., malicious models). The heat map indicates the threshold τ for each $\mathbf{w}$, which, as predicted by our theory, is the percentage of poisoning required to achieve $\mathbf{w}$ through retraining. In Figure 3.1 (middle) we run the gradient canceling attack (see Section 3.5) with varying percentage ε_d and verify that indeed we can reduce the 100% clean accuracy to 0% iff $\varepsilon_d \geq \tau$. In Figure 3.1 (right) we plot the magnitude of the gradient of the target parameter $\mathbf{w}$ over the mixed dataset (clean training data plus poisoned data), as an approximate measure of how close $\mathbf{w}$ can

be achieved by retraining on the mixed dataset. Overall, the larger ε_d is, the larger the accuracy drop is (not surprisingly) and the smaller the gradient norm is, with a clear transition once ε_d crosses τ (perhaps surprisingly).

3.4.3 Multiple Output Linear Models

Next, we extend our results to multiple outputs (classes):

Theorem 3 (Multiclass). *Consider $\ell(\mathbf{x}, \mathbf{y}; W) = l(W^\top \mathbf{x}, \mathbf{y})$ for some loss l. Then*[3]*,*

$$G(\mathbf{x}, \mathbf{y}) := \nabla_W \ell(\mathbf{x}, \mathbf{y}; W) = \mathbf{x} \otimes \nabla l(W^\top \mathbf{x}, \mathbf{y}).$$

Suppose $W^\top G(\mu)$ is non-degenerate and $\Gamma = \mathcal{P}$ contains all distributions. Then, W is λ-poisoning reachable iff

$$\mathbf{0} \in (1-\lambda) W^\top G(\mu) + \lambda \{W^\top G(\nu) : \nu \in \Gamma\}. \tag{3.5}$$

Compared to Theorem 5, condition equation 3.5 is no longer univariate but a square matrix of dimensions the same as $\mathbf{y}$ (the output). Nevertheless, we may simply take the trace on both sides to arrive at an easier albeit only necessary condition. We illustrate the last point through a familiar example:

Example 3 (Cross-entropy). *Let $\mathbf{h} = W^\top \mathbf{x}$. The cross-entropy loss corresponds to*

$$l(\mathbf{h}, \mathbf{y}) = -\langle \mathbf{h}, \mathbf{y} \rangle + \log \sum\nolimits_k \exp h_k,$$

where $\mathbf{y}$ is one-hot. Taking trace on equation 3.5 we obtain

$$\begin{aligned} 0 &= (1-\lambda) g(\mu) + \lambda g(\nu), \ \text{where} \\ g(\nu) &= \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\nu} \langle \mathbf{h}, \mathbf{p} - \mathbf{y} \rangle, \end{aligned}$$

and $\mathbf{p} := \texttt{softmax}(\mathbf{h}) = \exp(\mathbf{h}) / \sum_k \exp(h_k)$. In Appendix B.1 we prove the tight bound $-\mathfrak{W}(\frac{c-1}{e}) \le g(\nu) \le \infty$, leading to the necessary condition for inducing W:

$$\varepsilon_d \ge \tau = \tau(c) := \max\{\langle W, G(\mu) \rangle / \mathfrak{W}(\tfrac{c-1}{e}), 0\}, \tag{3.6}$$

where c is the number of classes. When $c = 2$, we recover the sufficient and necessary condition in equation 3.4.

We remark that all of our results continue to hold as necessary (but may not be sufficient) conditions for neural networks where the input $\mathbf{x}$ goes through a *learned* feature transformation $\varphi(\mathbf{x}; \mathbf{u})$, parameterized by $\mathbf{u}$:

[3] We use the notation $\mathbf{a} \otimes \mathbf{b} := \mathbf{a}\mathbf{b}^\top$ for two column vectors.

Theorem 4 (Neural Networks)**.** *Consider* $\ell(\mathbf{x}, \mathbf{y}; W, \mathbf{u}) = l(\mathbf{h}, \mathbf{y})$ *for some loss* l*, where* $\mathbf{h} := W^\top \varphi(\mathbf{x}; \mathbf{u})$*. Then,*

$$\nabla_W \ell(\mathbf{x}, \mathbf{y}; W, \mathbf{u}) = \varphi(\mathbf{x}; \mathbf{u}) \otimes \nabla_{\mathbf{h}} l(\mathbf{h}, \mathbf{y}) \tag{3.7}$$

$$\nabla_{\mathbf{u}} \ell(\mathbf{x}, \mathbf{y}; W, \mathbf{u}) = \nabla_{\mathbf{u}} \varphi(\mathbf{x}; \mathbf{u}) W \nabla_{\mathbf{h}} l(\mathbf{h}, \mathbf{y}), \tag{3.8}$$

and $(W, \mathbf{u})$ *is* λ*-poisoning reachable iff there exists* $\nu \in \Gamma$ *such that*

$$\mathbf{0} \in (1 - \lambda) G(\mu) + \lambda G(\nu), \tag{3.9}$$

where $G(\nu) := \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\nu} (\nabla_W \ell, \nabla_{\mathbf{u}} \ell)$*. In particular,* $(W, \mathbf{u})$ *is* λ*-poisoning reachable only if there exists some* $\nu \in \Gamma$ *such that*

$$\mathbf{0} \in (1 - \lambda) G_1(\mu) + \lambda G_1(\nu), \tag{3.10}$$

where $G_1(\nu) := \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\nu} \varphi(\mathbf{x}; \mathbf{u}) \otimes \nabla_{\mathbf{h}} l(\mathbf{h}, \mathbf{y})$*.*

Proof. It is straightforward to compute the gradients in equation 3.7 and equation 3.8. The iff condition in equation 3.9 then follows from Theorem 1. The necessary condition in equation 3.10 is obtained by simply ignoring the second part of $G(\mu)$ (that corresponds to $\nabla_{\mathbf{u}} \ell$). □

From equation 3.10 we conclude that the poisoning distribution ν must be supported at least on $s = \operatorname{rank}(G_1(\mu))$ points, as long as $\lambda \in (0, 1)$. Taking inner product w.r.t. WA on both sides of equation 3.10 we obtain

$$0 = (1 - \lambda) g_A(\mu) + \lambda g_A(\nu), \tag{3.11}$$

where $g_A(\nu) = \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\nu} \langle A \nabla_{\mathbf{h}} l(\mathbf{h}, \mathbf{y}), \mathbf{h} \rangle$ and A is arbitrary. The condition equation 3.11 is univariate and easy to check, albeit being necessary but not sufficient. We remark that the free choice of the matrix A may be exploited to tighten this necessary condition.

3.5 Gradient Canceling Attack

In this section we discuss how to find a poisoning distribution $\nu \in \Gamma$ so that upon retraining on the mixed distribution $\chi = (1 - \lambda)\mu + \lambda\nu$, the target parameter $\mathbf{w}$ will be favored. We recall that μ is the (clean) training distribution and λ is the (absolute) poisoning proportion.

The algorithm we propose is very intuitive and directly inspired by our Definition 1: we simply find a poisoning distribution $\nu \in \Gamma$ so that

$$\mathbf{g}(\chi) = (1 - \lambda)\mathbf{g}(\mu) + \lambda \mathbf{g}(\nu) \approx \mathbf{0},$$

where recall that $\lambda = \frac{\varepsilon_d}{1+\varepsilon_d}$ and $\mathbf{g}(\nu) := \mathbb{E}_{\mathbf{z}\sim\nu}\nabla_{\mathbf{w}}\ell(\mathbf{z};\mathbf{w})$ is the model gradient computed over a distribution ν. Thus, we arrive at the following Gradient Canceling problem[4]:

$$\min_{\nu\in\Gamma} \ \tfrac{1}{2}\|\mathbf{g}(\mu) + \varepsilon_d \mathbf{g}(\nu)\|_2^2, \tag{3.12}$$

which is always convex (since $\mathbf{g}(\nu)$ is linear in ν while Γ is a convex subset of admissible distributions). In Appendix B.2 we discuss a measure optimization approach for solving equation 3.12, while below we focus on a Lagrangian approach that directly constructs a poisoning dataset and eliminates the need of resampling from ν.

In more details, we constrain the poisoning distribution to be uniform over $n\varepsilon_d$ data points $\{\mathbf{z}_j\}$:

$$\hat{\nu} = \frac{1}{n\varepsilon_d}\sum_{j=1}^{n\varepsilon_d} \delta_{\mathbf{z}_j}, \tag{3.13}$$

where n is the size of the (clean) training set and $\delta_{\mathbf{z}}$ denotes the point mass concentrated on $\mathbf{z}$. We only optimize the locations of the points $\mathbf{z}_j$ but keep their mass uniform throughout.

Thus, we arrive at the following problem:

$$\min_{\hat{\nu}\in\Gamma} \ \frac{1}{2}\left\|\mathbf{g}(\mu) + \varepsilon_d \cdot \frac{1}{n\varepsilon_d}\sum_{j=1}^{n\varepsilon_d} \nabla_{\mathbf{w}}\ell(\mathbf{z}_j;\mathbf{w})\right\|_2^2, \tag{3.14}$$

where we remind that $\mathbf{g}(\mu) = \mathbb{E}_{\mathbf{z}\sim\mu}\nabla_{\mathbf{w}}\ell(\mathbf{z};\mathbf{w})$ as well as the target parameter $\mathbf{w}$ are fixed during optimization. For supervised tasks where $\mathbf{z} = (\mathbf{x}, \mathbf{y})$, we may choose to optimize both the feature $\mathbf{x}$ and label $\mathbf{y}$, or simply optimize the feature $\mathbf{x}$ only (as in our experiments).

We apply (projected) gradient descent to solve equation 3.14, where the gradient with respect to the j-th poisoning data $\mathbf{z}_j$ is:

$$\frac{\partial}{\partial \mathbf{z}_j} = \frac{1}{n}\nabla_{\mathbf{z}}\nabla_{\mathbf{w}}\ell(\mathbf{z}_j;\mathbf{w}) \cdot [\mathbf{g}(\mu) + \varepsilon_d \mathbf{g}(\hat{\nu})]. \tag{3.15}$$

We note that using auto-differentiation, the above matrix vector product can be computed very efficiently, costing essentially as much as gradient calculation. The constraint for $\hat{\nu}$ to lie in Γ can be handled by projection. For instance, the constraint $\mathbf{z} \in \mathbb{Z}$ (e.g. pixels must lie in $\mathbb{Z} = [0, 1]$) can be enforced by projecting the gradient update onto $\mathbb{Z}$.

We summarize the Gradient Canceling(GC) attack in Algorithm 2, and we emphasize that it can take *any* target parameter $\mathbf{w}$ as "teacher" and construct a poisoning dataset such that retraining will arrive (approximately) at $\mathbf{w}$. We note that Gradient Canceling is a refinement of the KKT attack of Koh, Steinhardt, et al. (2022): our refinement lies in the

[4]Other merit functions than the ℓ_2-norm here can also be used.

Algorithm 2: Gradient Canceling(GC) Attack

Input: training distribution μ,
step size η, poisoning fraction ε_d, and target parameter $\mathbf{w}$.

initialize poisoned dataset $\hat{\nu}$ in equation 3.13, e.g., randomly subsample clean training data

calculate $\mathbf{g}(\mu) = \mathbb{E}_{\mathbf{z}\sim\mu}\nabla_{\mathbf{w}}\ell(\mathbf{z};\mathbf{w})$

for $t = 1, 2, \ldots$ **do**

  calculate $\mathbf{g}(\hat{\nu}) \leftarrow \frac{1}{n\varepsilon_d}\sum_{j=1}^{n\varepsilon_d}\nabla_{\mathbf{w}}\ell(\mathbf{z}_j;\mathbf{w})$

  calculate loss $\mathcal{L} = \frac{1}{2}\|\mathbf{g}(\mu) + \varepsilon_d\mathbf{g}(\hat{\nu})\|_2^2$

  update poisoned data using equation 3.15: $\mathbf{z}_j \leftarrow \mathbf{z}_j - \eta\frac{\partial\mathcal{L}}{\partial\mathbf{z}_j}$

  project to admissible set: $\hat{\nu} \leftarrow \mathrm{Proj}_{\Gamma}(\hat{\nu})$

return the final poisoned dataset $\hat{\nu}$

generalization to any loss ℓ, different optimization strategy, exploring target parameters generated by the much stronger GradPC attack (Sun et al. 2020), experimenting on a variety of different models, and studying the effect of the poisoning proportion. Other authors such as Suya et al. (2021) also explored (rather costly) attacks based on a target parameter in the online setting (that require retraining in each round), whereas their lower bound on the amount of poisoned points may not be easily computed even for logistic regression.

Comparison with Gradient Matching. Geiping et al. (2021) proposed a gradient matching algorithm for crafting *targeted* poisoning attacks, which can be easily adapted to our setting. Suppose that a defender aims at minimizing a loss ℓ to achieve model $\mathbf{w}$ on (clean) training distribution μ. Let $\eth$ be a reversed version of ℓ. For example, if l is the cross-entropy loss in Example 3, then

$$\eth(\mathbf{h},\mathbf{y}) = -\log[1-\exp(-l(\mathbf{h},\mathbf{y}))], \text{ where } \mathbf{h} = W^\top\mathbf{x},$$

is the reversed cross-entropy loss (Fowl, Chiang, et al. 2021). As $\eth$ discourages the model from classifying clean data $\mathbf{x}$ as $\mathbf{y}$, Geiping et al. (2021) proposed to match its gradient $\nabla_{\mathbf{w}}\eth(\mu,\mathbf{w})$ over a poisoned distribution $\hat{\nu}$ (within some proximity of μ), based on some dissimilarity function $\mathcal{S}$ (e.g., cosine dissimilarity):

$$\min_{\hat{\nu}\in\Gamma}\ \mathcal{S}\left(\nabla_{\mathbf{w}}\eth(\mu;\mathbf{w}),\ \nabla_{\mathbf{w}}\ell(\hat{\nu};\mathbf{w})\right).$$

We point out some key differences between gradient matching (Fowl, Chiang, et al. 2021) and our work: (1) Gradient matching focuses on $\lambda = 1$, i.e., an attacker is able to modify the entire training set. While this is useful in certain settings (e.g., crafting "unlearnable examples"), it masks the effect of the poisoning proportion, which, as we showed in Section 3.4, can determine if a target parameter is poisoning reachable at all. (2) Gradient matching requires the construction of a reversed loss, whose gradient may not be at the

Table 3.2: The attack accuracy/accuracy drop (%) on MNIST, CIFAR-10 and TinyImageNet. We perform GC based on the target parameters generated by GradPC. Our attack significantly outperforms TGDA and Gradient Matching.

Dataset	Target Model	Clean Acc	GradPC	**Gradient Canceling**				**TGDA**			**Gradient Matching**		
	ε_d	0	0	3%	10%	100%	$\varepsilon = \tau$	0.03	0.1	1	0.03	0.1	1
MNIST	LR	92.35	-70.87 (τ=1.15)	**-22.97**	**-63.83**	**-67.01**	-69.66	-2.79	-4.01	-8.97	-3.33	-8.14	-12.13
	NN	98.04	-20.03 (τ=2.48)	**-6.10**	**-9.77**	**-12.05**	-19.05	-1.50	-1.72	-5.49	-2.82	-3.71	- 4.03
	CNN	99.13	-24.78 (τ=0.98)	**-9.55**	**-20.10**	**-23.80**	-23.77	-1.11	-1.31	-4.76	-2.01	-3.80	-6.94
CIFAR-10	ResNet-18	94.95	-21.69 (τ=129%)	**-13.73**	**-16.40**	**-18.33**	-19.98	-5.54	-6.28	-17.21	-6.01	-7.62	-9.80
TinyImageNet	ResNet-34	66.65	-24.77 (τ=108%)	**-13.22**	**-16.11**	**-20.15**	-22.79	-4.42	-6.52	-14.33	-5.53	-7.72	-10.85

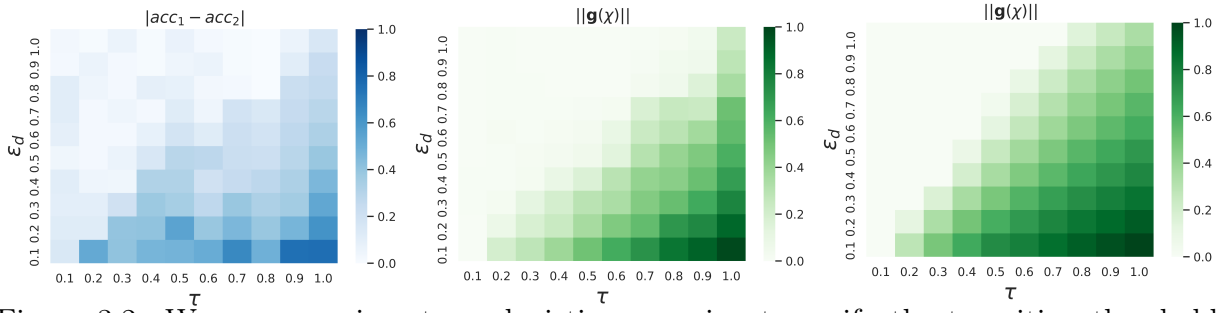


Figure 3.2: We run experiments on logistic regression to verify the transition threshold τ in Corollary 1. **Left**: accuracy difference between GC and GradPC on 10-d Gaussian dataset; **Middle**: norm of the gradient over the mixed dataset χ on 10-d Gaussian dataset; **Right**: norm of the gradient over the mixed dataset χ on MNIST-17.

same scale as that of the loss we are interested in. Thus, one typically can only hope to align the direction of gradients, which does not necessarily imply the desired matching in performance. In contrast, Algorithm 2 only requires the original loss and our theory gives guidance on when it succeeds. (3) There is no guarantee that after retraining over $\hat{\nu}$, gradient matching will arrive at the target parameter while Algorithm 2 explicitly aims to achieve this goal. Further experimental comparisons against gradient matching will be presented in Section 3.6 and Appendix B.3.

3.6 Experiments

We perform extensive experiments to verify our main results: (a) how competitive the gradient canceling attack (Algorithm 2) is compared to SOTA baselines in indiscriminate data poisoning? (b) to what extent our threshold τ (see equation 3.6) can predict model poisoning reachability?(c) how effective gradient canceling remains against certain existing defense mechanisms?

3.6.1 Experimental Settings

Dataset: We consider image classification on MNIST (L. Deng 2012) (60k training and 10k test images), CIFAR-10 (Krizhevsky 2009) (50k training and 10k test images), and TinyImageNet (Chrabaszcz et al. 2017) (100k training, 10k validation and 10k test images). For the first two datasets, we further split the training data into 70% training set and 30% validation set, respectively.

Target Models: We examine the following ML models. On MNIST: Logistic Regression (LR), a fully connected neural network (NN) with three layers and a convolutional neural network (CNN) with two convolutional layers, max-pooling and one fully connected layer; On CIFAR-10: ResNet-18 (K. He, X. Zhang, et al. 2016); and on TinyImageNet: ResNet-34.

Baselines: We compare to TGDA (Lu, Kamath, et al. 2022) and Gradient Matching (Geiping et al. 2021) attacks. To our knowledge, the TGDA attack is one of the most effective data poisoning attacks against neural networks. Gradient Matching was originally proposed for targeted attacks and unlearnable examples, and we also compare against it due to its similarity with the Gradient Canceling (GC) attack.

Implementation: For GC implementation, we follow Algorithm 2 and we discuss the effect of the projection step in Section 3.6.4. Most of our target parameters are generated using GradPC [5] except LR on MNIST, where we use $\varepsilon_w = 1$ to allow meaningful accuracy drop and transition threshold τ [6]. We initialized the poisoned points with a random $n\varepsilon_d$ sample from the clean training set and we only optimized the feature vectors but not the labels. Accuracy drops are obtained after retraining.

Evaluation Protocol: To evaluate the effectiveness of different attacks, we first apply each attack to acquire its poisoned set and then retrain the model from scratch (initialized with the same random seed across all attacks) on both clean and poisoned data until convergence. The (test) accuracy drop, compared with clean accuracy (obtained by training on clean data only), is reported across all experiments.

3.6.2 How Competitive Is Gradient Canceling (GC)?

Table 3.2 reports the accuracy drop of LR, NN, CNN and ResNet due to GC on the aforementioned datasets. We note the trade-off of ε_w in GradPC when generating a target parameter $\mathbf{w}$: the larger ε_w is, the more effective GradPC is but also the larger the resulting transition threshold τ is, meaning that GC (or any other data poisoning attack) can succeed (in reproducing $\mathbf{w}$) only with a larger proportion ε_d of poisoned points. We used $\tau = \tau(2)$ in Table 3.2 as we find it is much more indicative than the more conservative $\tau(c)$ (which is roughly 11 times smaller on TinyImageNet and 4 times smaller otherwise).

[5] We follow the implementation in `https://github.com/TobiasLee/ParamCorruption`.

[6] We discuss the selection of target parameters in Appendix B.4.

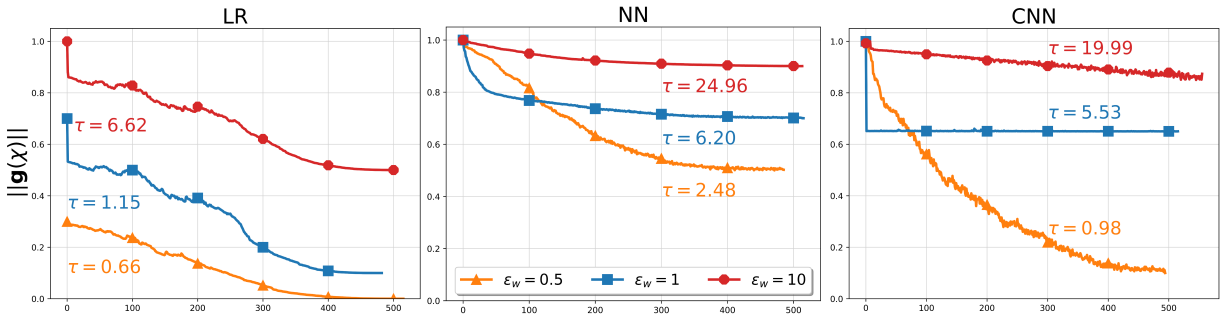


Figure 3.3: The learning curve for running GC on MNIST with different target models $\mathbf{w}$ and ε_w. We fix $\varepsilon_d = 1$, and the curves indicate the decrease of the gradient $\|\mathbf{g}(\chi)\|$ w.r.t. GC epoch. We confirm that GC fails to achieve $\mathbf{w}$ when $\varepsilon_d < \tau$.

We observe that GC is much more effective than TGDA and Gradient Matching, across all datasets, models, and choices of ε_d. This confirms that existing data poisoning attacks are under-optimized and there is room for future improvements. Moreover, when ε_d approaches the transition threshold τ, GC, a *bona fide* data poisoning attack, indeed achieves a comparable accuracy drop as GradPC (which directly overwrites the model). While Table 3.1 still has room to improve, both in terms of the tightness of τ and the effectiveness of GC, we believe our results yield significant insights on indiscriminate data poisoning, in particular the theoretical and experimental quantification of the detrimental effect of a large proportion ε_d of poisoned points.

3.6.3 Predicting Poisoning Reachability Using τ

Next, we further examine the predictability of the transition threshold $\tau \approx \max\{3.6 \langle W, G(\mu) \rangle, 0\}$, whose main term is simply proportional to the inner product between a target parameter and its gradient on the clean training data.

Binary Logistic Regression: We have already shown the predictability of τ on the OR dataset in Figure 3.1. In Figure 3.2 we show additional results on a 10-dimensional Gaussian dataset (see Appendix B.3.1) and MNIST-17 (consisting only of digits 1 and 7). The observations are similar: GC could achieve similar accuracy drops as GradPC (which directly overwrites the model), as long as ε_d crosses the threshold τ. We note that the threshold τ tends to be more conservative as the dimension of the problem increases, which we believe is largely because the optimization cost of GC becomes accordingly higher, making convergence harder to attain.

Multi-class with Cross-Entropy: We also perform experiments on multi-class problems with the cross-entropy loss in Example 3. In Table 3.2 we have confirmed that when $\varepsilon_d > \tau$, GC largely achieves the target parameters generated by GradPC. We now further examine the opposite case where $\varepsilon_d < \tau$. We fix $\varepsilon_d = 1$ and vary ε_w in GradPC, consequently generating target parameters with varying τ on MNIST. Figure 3.3 shows how much the gradient $\|\mathbf{g}(\chi)\|$ of the target parameters decreases w.r.t. each epoch of

Table 3.3: Accuracy drop (%) of Gradient Canceling (w/wo clipping) on MNIST against Sever defense (+ indicates the accuracy increased by the defense). GC-c: GC with clipped output; GC-d: GC after defense; GC-cd: GC-c after defense.

Model	Clean	ε_d	GC	GC-c	Sever	
					GC-d	GC-cd
LR	92.35	0.03	-22.79	-11.28	-12.81 / +**9.98**	-9.66 / +**1.62**
		0.1	-63.83	-26.77	-59.79 / +**4.04**	-25.53 / +**1.24**
		1	-67.01	-28.99	-65.01 / +**2.00**	-27.89 / +**1.10**
NN	98.04	0.03	-6.10	-3.25	-3.22 / +**2.88**	-2.26 / +**0.90**
		0.1	-9.77	-5.10	-7.66 / +**2.11**	-4.46 / +**0.56**
		1	-12.05	-6.53	-10.02 / +**2.03**	-6.11 / +**0.42**
CNN	99.13	0.03	-9.55	-5.87	-5.55 / +**4.00**	-4.36 / +**1.51**
		0.1	-20.10	-12.50	-16.55 / +**3.55**	-11.32 / +**1.18**
		1	-23.80	-13.32	-21.05 / +**2.75**	-12.51 / +**0.81**

Table 3.4: The Certified Accuracy (CA) (%) of DPA and Accuracy drop (%) of Gradient Canceling ($\varepsilon_d = 0.008$) on MNIST against the DPA defense (+ indicates the accuracy increased by the defense).

Model	Clean	k	Clean (DPA)	GC	CA	DPA
LR	92.35	1200	91.33	-8.25	47.12	-4.68/+**3.57**
		3000	89.97	-8.25	49.23	-4.21/+**4.04**
NN	98.04	1200	94.65	-2.25	46.11	-1.29/+**0.96**
		3000	92.37	-2.25	48.52	-1.17/+**1.08**
CNN	99.13	1200	95.53	-2.77	47.22	-1.66/+**1.11**
		3000	93.15	-2.77	50.01	-1.52/+**1.25**

GC (when χ, the mixed dataset, gets updated). We observe that the gradients do not converge to 0, indicating that GC failed to produce the target parameters. The failure of GC indicates that a larger poisoning proportion ε_d may be necessary to produce the target parameters, as confirmed by our theory.

3.6.4 Does GC Remain Effective Against Defenses?

Lastly, we choose several defenses from (Angel et al. 2022) and examine the effectiveness of GC against (1) a distribution-wise certified defense Sever (Diakonikolas, Kamath, Kane, J. Li, Steinhardt, et al. 2019), which removes ε_d training points with the highest outlier scores, defined using the top singular value of the gradient matrix, and (2) one of the SOTA pointwise certified defenses (A. Levine and Feizi 2021; W. Wang, A. Levine, et al. 2022; W. Wang, A. J. Levine, et al. 2022) called Deep Partition Aggregation (DPA) (A. Levine and Feizi 2021), which provides certified robustness for individual test samples. More results w.r.t. other defenses (e.g., influence defense and max-up defense) can be found in Appendix B.3.8.

Results on Sever: Table 3.3 reports the accuracy drops on MNIST. We observe that (1) Sever indeed reduces the effectiveness of GC, consistently across all models. (2) Clipping poisoned data to the range of the clean training set makes GC more robust against all defenses, at the cost of less effectiveness in terms of accuracy drop. (3) Even with clipping and against defenses, GC still largely outperforms TGDA and Gradient Matching. (4) Larger ε_d generally makes GC both more effective and more robust, which matches our observation in least-squares regression (see Appendix B.3.3).

Results on DPA: although DPA is originally proposed for pointwise robustness, it can be easily applied to the indiscriminate data poisoning setting. Here we choose $k = 1200/3000$ for DPA and fix $\varepsilon_d = 0.008$ to roughly preserve median certified robustness on MNIST. Note that we choose the base classifiers to be the same as the target models. We report the certified accuracy (CA), which is the percentage of certified robust examples among the test set, and (relative) accuracy increase due to deploying DPA in Table 3.4. We observe that DPA is generally effective against GC, where the (relative) accuracy increased by DPA roughly approaches its certified accuracy. For example, on LR with $k = 3000$, GC was able to decrease test accuracy by 8.25%, whereas with the DPA defense, 4.04% (relative) accuracy drop of GC are rectified, leading to an effectiveness that is roughly proportional to its certified accuracy, i.e., $4.04 \approx 8.25 * 0.4923$.

3.7 Conclusion and Future Work

In this work, we introduce the notion of *model poisoning reachability* as a technical tool to study the intrinsic limits in model-targeted data poisoning. We give complete characterizations on the poisoning ratio that any data poisoning attack has to satisfy (in order to induce a given target parameter), and we derive an easily computable threshold that is readily applicable and gives guidance on crafting effective model-targeted attacks. Using the gradient canceling attack, we perform extensive experiments on a number of datasets and models to quantify the critical role played by the poisoning ratio, confirm the precision of our transition threshold, and achieve better performance against existing baselines (w/wo several existing defenses). Our empirical results also reveal further room to sharpen the transition threshold and develop more effective data poisoning attacks, and we mention the exciting possibility of designing (clean) in-house data to mitigate and regulate the risk of future poisoning attacks.

One limitation of this work is its focus on achieving specific target parameters, which may not always be available or necessary. Indeed, data poisoning attacks that are not based on any target parameter abound. However, we point out that our work may still be valuable for the latter class of attacks, for instance, as a distillation device: a data poisoning attack can use our threshold to evaluate the potential "wastefulness" of its constructed poisoning set (along with the model parameter obtained by retraining) and then use GC to further distill and improve it. Another limitation is that most existing data poisoning attacks, including GC, assume a lot of knowledge of the victim model (e.g., fixed architecture,

access to clean training data, etc.) and hence may not always be realistic. Advanced and adaptive defense mechanisms may also thwart the effectiveness of many attacks (including GC). Further investigations of these issues form another important direction for future research.

Chapter 4

Indiscriminate Data Poisoning Attacks on Pre-trained Feature Extractors

4.1 Preface

Abstract Machine learning models have achieved great success in supervised learning tasks for end-to-end training, which requires a large amount of labeled data that is not always feasible. Recently, many practitioners have shifted to self-supervised learning (SSL) methods (e.g., contrastive learning) that utilize cheap unlabeled data to learn a general feature extractor via pre-training, which can be further applied to personalized downstream tasks by simply training an additional linear layer with limited labeled data. However, such a process may also raise concerns regarding data poisoning attacks. For instance, indiscriminate data poisoning attacks, which aim to decrease model utility by injecting a small number of poisoned data into the training set, pose a security risk to machine learning models, but have only been studied for end-to-end supervised learning. In this chapter, we extend the exploration of the threat of indiscriminate attacks on downstream tasks that apply pre-trained feature extractors. Specifically, we propose two types of attacks: (1) *the input space attacks*, where we modify existing attacks (e.g., TGDA attack and GC attack) to directly craft poisoned data in the input space. However, due to the difficulty of optimization under constraints, we further propose (2) *the feature targeted attacks*, where we mitigate the challenge with three stages, firstly acquiring target parameters for the linear head; secondly finding poisoned features by treating the learned feature representations as a dataset; and thirdly inverting the poisoned features back to the input space. Our experiments examine such attacks in popular downstream tasks of fine-tuning on the same dataset and transfer learning that considers domain adaptation. Empirical results reveal that transfer learning is more vulnerable to our attacks. Additionally, input space attacks are a strong threat if no countermeasures are posed, but are otherwise weaker than feature targeted attacks.

The following sections are adapted from the main paper and appendix of Y. Lu, M. Y.

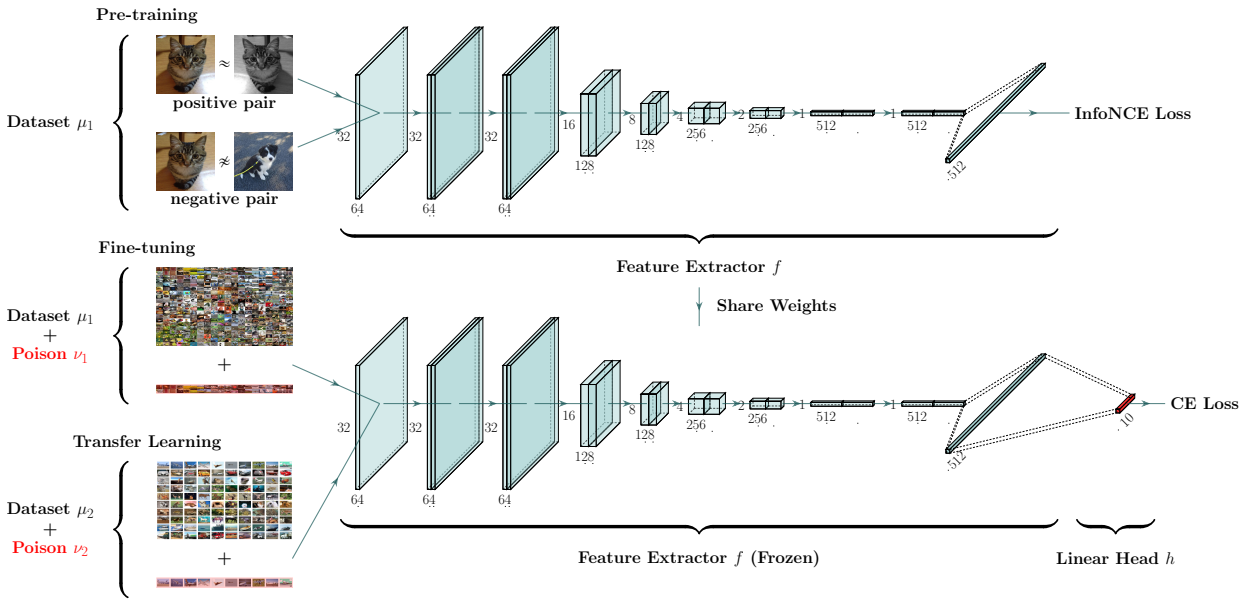


Figure 4.1: An illustration of our threat model:(top row) we acquire the weights of a feature extractor f with contrastive learning methods and optimizing w.r.t. the InfoNCE loss; (bottom row) we inject poisoned samples to the training dataset on downstream applications (image classification in this example) to perturb the linear head only. We examine two scenarios in this chapter: (1) fine-tuning where pre-training and downstream tasks share the same training set; and (2) transfer learning where the downstream task is performed on a different dataset.

Yang, G. Kamath, and Y. Yu (2024). "Indiscriminate Data Poisoning Attacks on Pre-trained Feature Extractors". In: *Secure and Trustworthy Machine Learning*. IEEE.

4.2 Introduction

From the previous chapters, we know indiscriminate attacks have been well-studied under end-to-end training schemes (Biggio et al. 2012; Koh, Steinhardt, et al. 2022; Lu, Kamath, et al. 2022, 2023; Muñoz-González, Biggio, et al. 2017; Suya et al. 2021, and the references therein), e.g., supervised learning for image classification tasks. However, as end-to-end training urges the need for large labeled datasets, practitioners with limited budgets are gradually adopting pre-trained feature extractors returned by self-supervised learning (SSL, e.g., contrastive learning methods (T. Chen, Kornblith, Norouzi, et al. 2020; K. He, Fan, et al. 2020)) for personalized downstream tasks. Specifically, SSL pre-trains a general feature extractor f with potentially unlimited unlabeled data (e.g., in the wild) such that one only needs to train an additional linear head h on top of f. While several works have studied how to poison the pre-training process on obtaining f (Balcan et al. 2022; Carlini and Terzis 2021; H. He et al. 2022), there lacks an exploration of the vulnerability

of performing downstream tasks given a certified f under data poisoning attacks, which could pose a serious threat to SSL beneficiaries.

Concretely, we aim to answer the following question:

> *Can we poison downstream tasks with a fixed (and trustworthy) pre-trained feature extractor?*

Specifically, we consider two popular downstream tasks in this chapter: (1) use the same (but labeled) dataset as pre-training for *fine-tuning*[1] the linear head h (where the parameters of the feature extractor f is frozen); (2) adapt to a new dataset for *transfer learning*.

To address the above question, we first adopt existing data poisoning attacks in a straightforward manner (which we refer to as *input space attack*) to directly craft poisoned points using the GC or TGDA attack by optimizing towards a fixed encoder. For example, considering the classification problem and the GC attack (a model-targeted approach), we perform parameter corruption only on the linear head instead of the entire model to acquire target parameters.

In practice, without imposing any constraints on the poisoned samples, we observe that GC input space attack is indeed effective (e.g., induce 29.54% accuracy drop for fine-tuning and 39.35% for transfer learning with only $\varepsilon_d = 3\%$ poisoning budget). However, such an attack can also be easily defended by data sanitization methods as the poisoned points have a much bigger magnitude and are visually malicious (see Figure 4.2). To generate "benign" poisoned images, we further pose a fidelity constraint during optimization and observe that the effectiveness of the attacks is largely restricted (e.g., only 2.50% accuracy drop and 4.40% accuracy drop under the same experimental settings).

To improve the GC input space attack, we break down the problem into three stages to mitigate the optimization difficulties: (1) acquiring target parameters of the linear head; (2) obtaining poisoned features with GC feature-space attack by treating the learned feature embeddings $f(\mu)$ as the clean dataset, and directly constructing the poisoned feature set ζ, such that the linear evaluation is poor after retraining on $f(\mu) + \zeta$; (3) inverting the poisoned feature ζ back to the input space to construct ν that look visually similar to clean data, where we propose two viable approaches: decoder inversion that learns an inversion f^{-1} of the feature extractor, and feature matching that optimizes ν directly.

Finally, we examine another indiscriminate data poisoning attack of a different flavor, the Unlearnable Example (UE) attack. UE operates under the strong premise that an adversary is capable of perturbing the entire training set (i.e., the poisoning fraction $\varepsilon_d = \infty$), albeit the perturbation of each point is relatively small.

[1]The term "fine-tuning" can be used in different contexts for training with a pre-trained network as initialization. In this chapter, we restrict "fine-tuning" to performing contrastive learning and downstream tasks on the same dataset.

In our empirical evaluation, we observe that downstream tasks with a fixed feature extractor are surprisingly robust to the popular UE approach of H. Huang et al. (2021).

In summary, we make the following contributions:

- We expose the threat of indiscriminate data poisoning attacks on pre-trained feature extractors and set baselines on fine-tuning and transfer learning downstream tasks, where transfer learning is generally more vulnerable to the considered attacks;
- We tailor existing attacks (e.g., TGDA, Gradient Canceling attack, and Unlearnable Examples) to poisoning fixed future extractors and empirically identify the scenarios where they succeed (e.g., without constraints) and fail (with constraints);
- We propose new attacks called the *feature targeted* (FT) attack that involves three stages to alleviate the challenge of optimizing input space attacks with constraints and observe empirical improvement.
- Finally, we examine unlearnable examples in the context of poisoning fixed feature extractors and identify that a popular UE attack becomes much less effective on the downstream tasks.

4.3 Background

In this section, we introduce the background on self-supervised learning (e.g., contrastive learning) methods.

Contrastive Learning Recently, self-supervised learning methods, in particular, contrastive learning methods have been widely applied to learn general representations without any label information for various downstream tasks. Existing approaches aim to learn a good feature extractor f (parameterized by a neural network) by minimizing the distance between representations of positive samples $f(\mathbf{x}^1)$ and $f(\mathbf{x}^2)$ (where $\mathbf{x}^1$ and $\mathbf{x}^2$ are different data augmentations of a training sample $\mathbf{x}$), while maximizing that of negative samples $f(\mathbf{x}^1)$ and $f(\mathbf{x}_n^1)$ (where $\mathbf{x}_n$ is another sample from the training set) at the same time. Existing approaches including Contrastive Predictive Coding (CPC) (Oord et al. 2018), SimCLR (T. Chen, Kornblith, Norouzi, et al. 2020; T. Chen, Kornblith, Swersky, et al. 2020), MoCo (K. He, Fan, et al. 2020; Huynh et al. 2020) and f-MICL (Lu, G. Zhang, et al. 2023) apply InfoNCE-based losses to enforce the contrast between positive and negative pairs. We call this process *contrastive learning pre-training*. Upon deployment, given a fixed feature extractor f, we consider two popular schemes of performing downstream tasks in this chapter: (1) fine-tuning, where we apply the same (or slightly augmented) dataset used for pre-training and fine-tune an additional linear head on top of f (also known as linear evaluation). Such a linear head can be chosen to suit specific downstream



Figure 4.2: We visualize some clean training samples of CIFAR-10 (which serve as initialization to the attacks) in the first row, and poisoned samples generated by GC input space attacks (which induce an accuracy drop of 29.54%) for $\varepsilon_d = 0.03$ in the second row. The poisoned images show that GC input space attack generates images with no semantic meaning if no explicit constraints are posed. Clean images and their corresponding poisoned ones are chosen randomly.

tasks, e.g., classification, object detection, and semantic segmentation. In this chapter, we consider classification as an example; (2) transfer learning, where we aim at training on a personalized dataset, which is usually smaller than the dataset used in pre-training (e.g., transfer from ImageNet to CIFAR-10). Again we consider linear evaluation where a linear head is learned.

4.4 Input Space Attacks

Existing indiscriminate attacks on neural networks, i.e., Total Gradient Descent Ascent (TGDA) attack in Chapter 2, Gradient Canceling (GC) attack in Chapter 3 and Unlearnable Examples (H. Huang et al. 2021) generate (or optimize) poisoned points directly in the input space, which is suitable for end-to-end training. In this section, we discuss how to apply them to our problem setting, where part of the model parameters are fixed. Notably, we call this generic class *input space* attacks.

4.4.1 Problem Setting

We first specify our **threat model**:

- Knowledge of the attacker: the attacker has access to the training data, the target model (including its pre-trained weights), and the training procedure;
- Capability: the attacker can (actively or passively) inject a poisoned set to the training data;

- Objective: given a pre-trained feature extractor, the objective of a downstream task and the corresponding training dataset, an adversary aims to construct a poisoned set to augment the clean training set, such that by training on the mixed dataset, the performance of the downstream task on the test set is reduced.

We then formally introduce the problem of interest. Given a clean training distribution μ and target model architecture that consists of a (fixed) feature extractor f and a linear head h with learnable parameters $\boldsymbol{\omega}$, a standard ML pipeline aims to learn a set of clean parameters $\boldsymbol{\omega}$ to minimize its cost on μ while keeping f untouched:

$$\underset{\boldsymbol{\omega}}{\operatorname{argmin}} \; l(f(\mu); \boldsymbol{\omega}).$$

An adversary thus considers designing a poisoning distribution ν to maximize the cost on a clean validation set $\tilde{\mu}$ (which comes from the same distribution as the training set μ):

$$\underset{\nu \in \Gamma}{\operatorname{argmax}} \; l(f(\tilde{\mu}); \boldsymbol{\omega}). \tag{4.1}$$

As $\boldsymbol{\omega}$ is dynamically changing during optimization, solving equation 4.1 is essentially a nested optimization problem (i.e., a non-zero-sum Stackelberg game):

$$\begin{aligned} &\max_{\nu \in \Gamma} \; l(f(\tilde{\mu}); \boldsymbol{\omega}_*), \\ &\text{s.t. } \boldsymbol{\omega}_* \in \underset{\boldsymbol{\omega}}{\operatorname{argmin}} \; l(f(\mu) + f(\nu); \boldsymbol{\omega}), \end{aligned} \tag{4.2}$$

where the poisoning effectiveness is measured by retraining the linear head $\boldsymbol{\omega}$ on the mixed feature distribution $f(\mu) + f(\nu)$. Note that the attack budget is measured by the poisoning fraction $\varepsilon_d = |\nu|/|\mu|$. Firstly, we introduce two viable approaches for solving the problem when ε_d is small (e.g., $< 100\%$), namely the TGDA attack and GC attack. Next, we discuss UE (H. Huang et al. 2021), where $\varepsilon_d = \infty$, which amounts to substituting the entire training distribution μ as ν. Notably, we call these adaptations TGDA input space attack, GC input space attack and UE input space attack, respectively.

4.4.2 TGDA input space attack

Solving equation 4.2 is non-trivial, especially for the outer maximization problem as the dependence of $l(f(\tilde{\mu}); \boldsymbol{\omega}_*)$ on ν is indirectly through retraining the head model on the mixed distribution. As a result, one can not directly apply simple algorithms such as gradient ascent as it would lead to zero gradients. Lu, Kamath, et al. (2022) propose to measure the influence of ν through the total derivative $\mathsf{D}_\nu l(f(\tilde{\mu}); \boldsymbol{\omega}) = -\nabla_{\boldsymbol{\omega}\nu} l_2 \cdot \nabla^{-1}_{\boldsymbol{\omega}\boldsymbol{\omega}} l_2 \cdot \nabla_{\boldsymbol{\omega}} l_1$ (where we simplify $l_1 = l(f(\tilde{\mu}), \boldsymbol{\omega}), l_2 = l(f(\mu) + f(\nu); \boldsymbol{\omega})$), which implicitly appraises the change of $\boldsymbol{\omega}$ with respect to ν. Specifically, TGDA considers the following total gradient ascent step for the attacker and gradient descent step for the defender:

$$\nu = \nu + \eta \cdot \mathsf{D}_\nu l(f(\tilde{\mu}); \boldsymbol{\omega}),$$

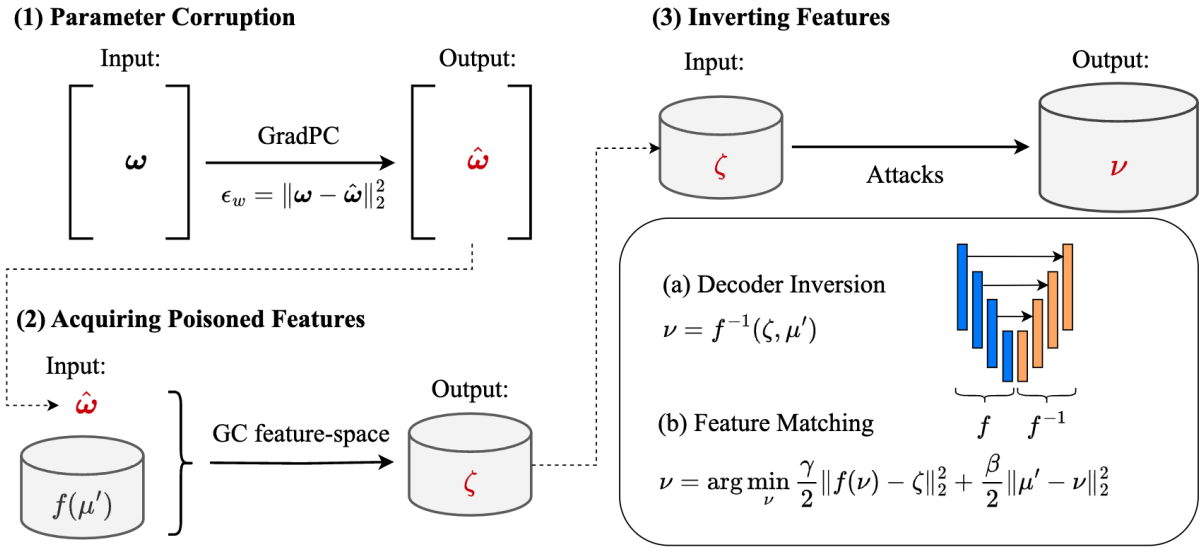


Figure 4.3: An illustration of the three stages of feature-targeted attacks: (1) obtaining the target linear head parameter $\boldsymbol{\omega}$ with GradPC; (2) acquiring poisoned features ζ with GC feature-space attack; (3) invert ζ back to the input space using feature matching or decoder inversion (decoder inversion requires training an autoencoder with a fixed encoder f).

$$\boldsymbol{\omega} = \boldsymbol{\omega} - \eta \cdot \nabla_{\boldsymbol{\omega}} l(f(\mu) + f(\nu); \boldsymbol{\omega}).$$

For implementation, we still perform the attack end-to-end, namely that we directly feed the data into the model architecture, while keeping f untouched (e.g., `requires_grad=False` in PyTorch).

4.4.3 GC input space attack

Next, we generalize the GC attack (discussed in Section 4.3) to our problem setting. Recall that to avoid solving the difficult equation 4.2, GC aims to reach a specific target parameter $\hat{\boldsymbol{\omega}}$ that the model will converge to by retraining on the mixed feature distribution. Here reachability is defined by having a vanishing (sub)gradient over the mixed feature distribution:

$$\nabla_{\boldsymbol{\omega}} l(f(\chi); \hat{\boldsymbol{\omega}}) \propto \nabla_{\boldsymbol{\omega}} l(f(\mu); \hat{\boldsymbol{\omega}}) + \nabla_{\boldsymbol{\omega}} l(f(\nu); \hat{\boldsymbol{\omega}}) = \mathbf{0}.$$

Thus, GC input space attack amounts to solving the following problem:

$$\underset{\nu \in \Gamma}{\operatorname{argmin}} \ \tfrac{1}{2}\|\nabla_{\boldsymbol{\omega}} l(f(\mu); \hat{\boldsymbol{\omega}}) + \nabla_{\boldsymbol{\omega}} l(f(\nu); \hat{\boldsymbol{\omega}})\|_2^2. \tag{4.3}$$

Note that GC requires a target parameter $\hat{\boldsymbol{\omega}}$ that performs poorly on the test set, which Lu, Kamath, et al. (2023) acquired by another attack that perturbs the clean model parameters

$\boldsymbol{\omega}$ directly, namely the Gradient Parameter Corruption (GradPC) attack of Sun et al. (2020). In a nutshell, GradPC overwrites $\boldsymbol{\omega}$ with $\hat{\boldsymbol{\omega}}$ directly in our problem setting using a gradient approach. Note that the perturbation introduced by GradPC is measured by $\varepsilon_w = \|\boldsymbol{\omega} - \hat{\boldsymbol{\omega}}\|_2^2$ and can be specified as a constraint by the algorithm.

In practice, we observe that GC input space attack is very effective without considering any constraints on ν, e.g., it approximately reaches the target parameter $\hat{\boldsymbol{\omega}}$ for $\varepsilon_d = 0.1$ in Table 4.1 and Table 4.2, Section 4.6. However, the poisoned samples generated by GC has a significantly larger magnitude than the clean distribution μ and does not retain semantic information (see Figure 4.2 for a visualization). To further generate "benign" poisoned points to throttle defenses, we add a penalty term $\frac{1}{2}\|\mu' - \nu\|_2^2$ in equation 4.3 on constraining ν to be similar with μ', which is part of μ that initializes ν ($|\mu'| = |\nu|$).

However, we find the optimization procedure to be much harder with such a constraint, and the attack effectiveness is largely restricted. For example, we observe an accuracy drop of only 2.5% compared to 29.54% (without constraints) in Table 4.1. To mitigate the difficulty of optimizations, we further propose feature targeted attacks in Section 4.5 by breaking down the problem into several stages.

4.4.4 UE (EMN) input space attack

Before diving into Section 4.5, we introduce the last input space attack we consider, the Unlearnable Examples (UE), where we pose a strong assumption on $\varepsilon_d = \infty$. We apply the error-minimizing noise (EMN) approach in H. Huang et al. (2021). Here the task is to perturb the entire training set with a noise factor δ such that the model learns a strong correlation between δ and the label information. Note that we consider δ to be a sample-wise noise, namely that $\mu + \delta = \{\mathbf{x}_i + \delta_i\}_{i=1}^{|\mu|}$. EMN input space attack optimizes the following objective:

$$\min_{\boldsymbol{\omega}} \min_{\delta \in \Gamma} l(f(\mu + \delta); \boldsymbol{\omega}). \tag{4.4}$$

Note that optimizing equation 4.4 is substantially easier than equation 4.2, as it is a min-min problem and can be easily solved by (alternating projected) gradient descent algorithms.

4.5 Feature Targeted Attack

Motivated by the optimization challenge of GC input space attack, we propose a staged strategy to mitigate the difficulty of the constrained problem. Assuming we ① acquired the target parameter $\hat{\boldsymbol{\omega}}$ with parameter corruption, let us rewrite equation 4.3 by introducing a feature distribution ζ:

$$\min_{\nu \in \Gamma} \min_{\zeta = f(\nu)} \frac{1}{2} \| \underbrace{\nabla_{\boldsymbol{\omega}} l(f(\mu); \hat{\boldsymbol{\omega}})}_{:=\mathbf{g}} + \nabla_{\boldsymbol{\omega}} l(\zeta; \hat{\boldsymbol{\omega}}) \|_2^2.$$

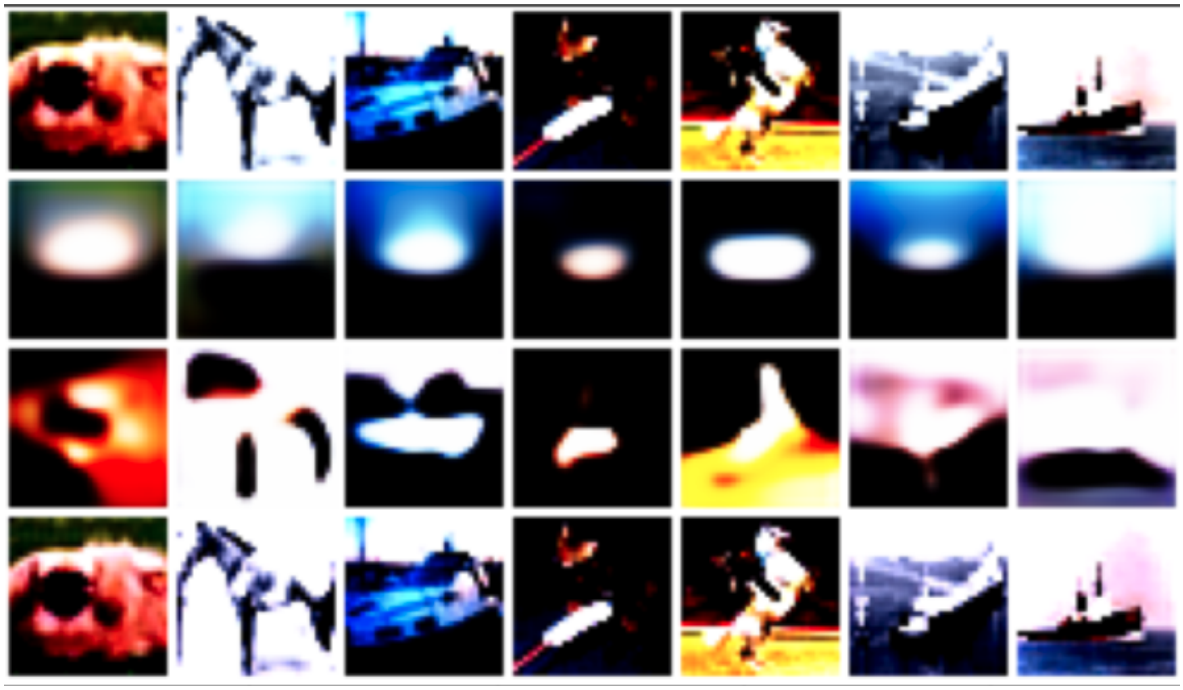


Figure 4.4: Here we visualize original test images (first row), and images reconstructed by an autoencoder with fixed ResNet-18 feature extractor learned by MoCo (second row), the same autoencoder trained end-to-end (third row), and the autoencoder with skip connections, i.e., a U-Net (fourth row).

Next, we relax both constraints by introducing penalty terms[2]:

$$\min_{\nu,\zeta} \underbrace{\tfrac{1}{2}\|\mathbf{g}+\nabla_{\boldsymbol{\omega}} l(\zeta;\hat{\boldsymbol{\omega}})\|_2^2}_{(2)} + \underbrace{\tfrac{\gamma}{2}\|f(\nu)-\zeta\|_2^2 + \tfrac{\beta}{2}\|\mu'-\nu\|_2^2}_{(3)}. \tag{4.5}$$

The variables ν and ζ are now separated, which immediately suggests an alternating algorithm that we elaborate on below.

We follow the above three steps and break down the optimization into corresponding three stages: (1): acquiring the target parameter $\hat{\boldsymbol{\omega}}$ with GradPC; solving (2) with GC feature space attack, where we treat the learned representation as the clean dataset and inject poisoned features directly; solving (3) where we invert the poisoned features back to the input space considering constraints in the input space, where we introduce two possible approaches: decoder inversion (without optimization) and feature matching (with optimization). Overall, we call this approach *Feature Targeted* (FT) attack and we visualize the attack procedures in Figure 4.3.

[2]Other penalty terms other than the (squared) L_2 norm could also be used.

4.5.1 GC feature space attack

Firstly, we generate poisoned features ζ using the GC attack. We chose GC as it is the best-performing algorithm in the feature space. We consider $f(\mu)$ as the clean feature training set and enable an attacker to inject poisoned features ζ directly. Following the notations in Section 4.4, GC considers reachability in the feature space:

$$\nabla_{\boldsymbol{\omega}} l(f(\mu); \hat{\boldsymbol{\omega}}) + \nabla_{\boldsymbol{\omega}} l(\zeta; \hat{\boldsymbol{\omega}}) = \mathbf{0},$$

and optimizes the following problem:

$$\underset{\zeta}{\operatorname{argmin}} \ \tfrac{1}{2} \|\nabla_{\boldsymbol{\omega}} l(f(\mu); \hat{\boldsymbol{\omega}}) + \nabla_{\boldsymbol{\omega}} l(\zeta; \hat{\boldsymbol{\omega}})\|_2^2,$$

which corresponds to the term (2) in equation 4.5. We denote ζ as the output of the GC feature space attack and apply it as input to the next stage.

Here the target parameter $\hat{\boldsymbol{\omega}}$ is generated by (1) GradPC from the feature space, where we treat $f(\mu)$ as the training set and perturb the linear head h. Unlike poisoning attacks, such a change does not make a significant difference for GradPC as it perturbs the linear head parameters directly.

4.5.2 Inverting features

Next, we show two viable approaches to solve (3) on inverting the poisoned feature back to the input space.

4.5.2.1 Decoder Inversion

First, we show how to solve (3) without optimizing ν directly by training a decoder network. Specifically, given a pre-trained feature encoder f that maps μ to $f(\mu)$, we aim to learn its inversion f^{-1} such that $f^{-1}(f(\mu)) \approx \mu$. For image reconstruction and related tasks (e.g., image inpainting, image-to-image translation, semantic segmentation), it is common to utilize an autoencoder architecture, where the encoder is f and the decoder is naturally f^{-1}. However, such an autoencoder is usually trained end-to-end, while we require training only the decoder f^{-1} with a fixed encoder f. In Figure 4.4 (rows 1-3), we apply a ResNet-18 encoder learned by MoCo on CIFAR-10 and train the corresponding decoder on reconstructing CIFAR-10. We show such a constraint indeed brings a challenge: the autoencoder with a fixed f does not learn the detailed structure of images compared to that trained end-to-end, which may imply that an encoder trained by contrastive learning is not necessarily a good encoder for image reconstruction. Additionally, we observe that a plain autoencoder does not reconstruct the images well, even when training end-to-end.

Previous works on image reconstruction tasks show that a key success of the architectural design is to add skip connections between encoders and decoders, also known as U-Net

(Ronneberger et al. 2015). In Figure 4.4 (row 4), we show that adding skip connections leads to superb reconstruction quality, even with a fixed encoder architecture.

Upon acquiring a good decoder f^{-1}, we can then invert the poisoned feature ζ back to the input space. Note that with a U-Net architecture, f^{-1} does not only take ζ as input but also the intermediate features of μ' [3], which we denote as $f_1^n(\mu')$, where n is the number of skip connections. In summary, the output of the decoder inversion is $\nu = f^{-1}(\zeta, f_1^n(\mu'))$, where the first term in (3) is minimized if the decoder is well-trained, and the second term is automatically handled by the skip connections.

4.5.2.2 Feature Matching

In decoder inversion, we train the decoder f^{-1} and hoping the acquired ν is a successful inversion, i.e., $f(\nu) = \zeta$. However, in practice, we find the distance between the two is still large, e.g., $\|f(\nu) - \zeta\|_2^2 \approx 150$, which indicates the gap between the decoder inversion and the implicit ground truth is still large (another indication is the performance gap between GC feature space attack and decoder inversion attack in Section 4.6).

To further close the gap between $f(\nu)$ and ζ, we introduce an alternative method that directly optimizes the poisoned dataset ν by matching its feature $f(\nu)$ to the target poisoned feature ζ while enforcing the distance between μ and ν to be small. We call this algorithm *Feature Matching*.

Our feature matching algorithm is inspired by the feature collision (FC) algorithm (Shafahi et al. 2018) in targeted attack. In targeted attacks, FC considers a target test sample t in the test set and a different base instance b. FC aims to construct a single poisoned data sample $\mathbf{x}_p$ such that the distance between $f(t)$ and $f(\mathbf{x}_p)$ in the feature space is small while $\mathbf{x}_p$ is close to b in the input space.

When adapting to our setting, instead of matching the feature of a target sample, we match the target feature ζ that we already acquired. Thus, we need only optimize the term (3) in equation 4.5:

$$\underset{\nu}{\operatorname{argmin}} \ \tfrac{\gamma}{2}\|f(\nu) - \zeta\|_2^2 + \tfrac{\beta}{2}\|\mu' - \nu\|_2^2,$$

where recall that the second term is a relaxation of the constraint $\nu \in \Gamma$. We present our algorithm in Algorithm 3, where we follow Shafahi et al. (2018) and apply a forward gradient step to minimize the first term on feature matching and a backward proximal step to enforce the constraint on the input space. Note that β is a hyperparameter for controlling the strength of the constraint.

Again, μ' is part of the training distribution where $|\mu'| = \varepsilon_d|\mu| = |\zeta|$. Instead of choosing μ' from the initialization of ζ, as in decoder inversion, we follow the following

[3] Here μ' refers to the clean samples corresponding to $f(\mu')$, which are the initialization of the GC feature space attack.

Algorithm 3: Feature Matching

Input: base instances as part of the training distribution $\mu' = \varepsilon_d \mu$, step size η, a pre-trained encoder f

Output: poisoned dataset ν

$\nu \leftarrow \mu'$ // initialize poisoned data

$\zeta \leftarrow f(\nu)$ // initialize poisoned feature

for $k = 1, ...$ **do**

  for $s = 1, 2, ...$ **do**

    $\zeta \leftarrow \zeta - \eta \cdot \nabla_\zeta [\frac{1}{2} \|\mathbf{g} + \nabla_{\boldsymbol{\omega}} l(\zeta; \hat{\boldsymbol{\omega}})\|_2^2]$

    $\zeta \leftarrow [\zeta + \eta\gamma f(\nu)]/(1 + \eta\gamma)$ // optional

  for $t = 1, 2, ...$ **do**

    $\nu \leftarrow \nu - \eta \cdot \nabla_\nu [\frac{\gamma}{2} \|f(\nu) - \zeta\|_2^2]$ // forward

    $\nu \leftarrow (\nu + \eta\beta\mu')/(1 + \eta\beta)$ // backward

better strategy. The key observation is that in practice, ζ is not necessarily close to the initial corresponding $\mu' = \nu_0$, thus making the initial optimization hard. In other words, we find it necessary to explore whether the choice of μ' can be optimized in order to find a better matching in the input space. A simple solution is to perform a grid search for individual poisoned features: for every $\mathbf{z} \in \zeta$, we calculate the point-wise distance $\|f(\mathbf{x}) - \mathbf{z}\|_2^2$ for every $\mathbf{x} \in \mu$. By ranking the distances, we simply choose the one with the smallest distance to form a pair $\{\mathbf{x}, \mathbf{z}\}$. Such a grid search provides a much better initialization and significantly reduces the running time upon convergence. Moreover, we find that such a process reduces the difficulty of optimization and helps the algorithm converge at a lower value.

4.6 Experiments

In this section, we present our experimental results on the proposed indiscriminate attacks. For a fixed feature extractor, we consider two approaches to performing supervised downstream tasks: (1) fine-tuning, where the contrastive learning pre-training and fine-tuning share the same training distribution (the difference lies on without/with label information); (2) transfer learning, where pre-training is performed on a (relatively) large-scale dataset (without label information), and the linear head is trained on a smaller or customized dataset with labels.

Our experiments will thus be structured as these two major parts, where we consider input space attacks and feature space attacks for each class.

Table 4.1: Fine-tuning results: comparison between input space attacks (TGDA and GC input space attacks with/without constraints on poisoned samples) and feature targeted attacks (Decoder inversion and Feature Matching with different β) w.r.t. different ε_d. The white background indicates methods that generate target parameters or features; blue indicates input space attacks and pink indicates feature targeted attacks.

Attack method	Clean	$\varepsilon_d = 0.03$ (1500)	$\varepsilon_d = 0.1$ (5000)	$\varepsilon_d = 0.5$ (25000)	$\varepsilon_d = 1$ (50000)
GradPC (input space)	89.71%	53.15% / -36.56%	53.15% / -36.56%	53.15% / -36.56%	53.15% / -36.56%
TGDA-input (no constraints)	89.71%	85.43% / - 4.28%	84.26% / - 5.45%	76.55% / -13.16%	70.38% / -19.33%
TGDA-input (with constraints)	89.71%	87.52% / - 2.19%	87.45% / - 2.26%	83.67% / - 6.04%	78.66% / -11.05%
GC-input (no constraints)	89.71%	60.17% / -29.54%	53.65% / -36.06%	53.15% / -36.56%	53.15% / -36.56%
GC-input (with constraints)	89.71%	87.21% / - 2.50%	87.05% / - 2.66%	80.30% / - 9.41%	75.33% / -14.38%
GradPC (feature space)	89.71%	53.00% / -36.71%	53.00% / -36.71%	53.00% / -36.71%	53.00% / -36.71%
GC-feature space	89.71%	53.10% / -36.61%	53.03% / -36.68%	54.09% / -35.62%	52.15% / -37.56%
Decoder inversion	89.71%	85.49% / - 4.22%	84.61% / - 5.10%	78.65% / -11.06%	72.25% / -17.46%
Feature matching ($\beta = 0.25$)	89.71%	83.41% / - 6.30%	82.33% / - 7.38%	76.15% / -13.56%	69.17% / -20.54%
Feature matching ($\beta = 0.1$)	89.71%	77.24% / -12.44%	76.56% / -13.15%	73.24% / -15.47%	65.14% / -24.57%
Feature matching ($\beta = 0.05$)	89.71%	75.34% / -14.37%	74.29% / -15.42%	71.99% / -17.72%	63.19% / -26.52%

4.6.1 Experimental setting

Hardware and Package: Fine-tuning experiments are run on a cluster with NIVIDIA T4 and P100 GPUs, while transfer learning experiments are run on another machine with 2 NIVIDIA 4090 GPUs. The platform we use is `PyTorch`.

Datasets: We consider image classification as the downstream task on the CIFAR-10 dataset (Krizhevsky 2009) (50k training and 10k testing images). For contrastive learning pre-training, we use CIFAR-10 and ImageNet 1K (J. Deng et al. 2009) for fine-tuning and transfer learning respectively.

Feature Extractors: Following the standard contrastive learning pre-training architectures, we examine the ResNet-18 (K. He, X. Zhang, et al. 2016) model with a feature dimension of size 512 for pre-training on CIFAR-10, following the common recipe of changing the convolution kernel size from 7×7 to 3×3. The model weights are obtained by training on SimCLR (T. Chen, Kornblith, Norouzi, et al. 2020) for 800 epochs with a batch size of 512. For clean linear evaluation, we acquire an accuracy of 89.71% with 100 epochs. For ImageNet pre-training we apply the ResNet-50 architecture using MoCo V3 (X. Chen, Xie, et al. 2021) with a feature dimension of size 2048 and a batch size of 4096, trained for 1000 epochs. We directly utilize the pre-trained weights provided in the MoCo V3 Github Repo[4]. Upon transfer learning, we follow the general recipe to resize the CIFAR-10 training sample from 32×32 to 224×224 and acquire an accuracy of 93.03% by training the linear head for 100 epochs.

Attacks implementations: We specify the implementations of all attacks we apply in this chapter:
(1) TGDA input space attack: we follow and modify the implementation of TGDA in

[4] `https://dl.fbaipublicfiles.com/moco-v3/r-50-1000ep/r-50-1000ep.pth.tar`

Chapter 2[5] while fixing the feature extractor of the follower (defender) and train for 200 epochs for all tasks;
(2) GC input space/feature space attack: we follow and modify the official code of GC in Chapter 3[6] and perform 2000 epochs or early stop upon reaching stopping criteria (loss smaller than 1). Note that for acquiring target parameters, we run GradPC[7] for one single attack step and set $\varepsilon_w = 1$;
(3) UE input space attacks: we follow and modify the implementation of H. Huang et al. (2021)[8] for sample-wise error minimizing noise generation;
(4) Decoder inversion: for the architectural design on the decoder, we use the implementation of this Github repo[9] as a key reference. For image reconstruction, we apply a combination of the multi-scale structural similarity index (MS-SSIM) loss (Z. Wang et al. 2003) and L_1 loss following the analysis of Khare et al. 2021;
(5) Feature matching: finally, we follow Algorithm 3 and set $k = 1$, $t = 2000$. We set the learning rate as 0.1 with a cosine scheduler. Unless specified, $\gamma = 1, \beta = 0.25$ across all tasks.

Label Information: During optimizations of indiscriminate attacks, we only modify the input $\mathbf{x}$ and generate clean-label poisoned samples. Specifically, we assign the label corresponding to the clean images during the initialization stage of every attack. Note that for feature matching, as we perform ranking after the GC feature space attack, we reassign the labels as that of clean base samples to every ζ, thus the output ν.

Evaluation Protocol: To evaluate the performance of each attack, we first acquire the poisoning dataset ν and retrain the model (initialized with the same random seed across all attacks) on both clean and poisoned data $\mu + \nu$ until convergence (we train for 100 epochs across all tasks). Our evaluation metric is the test accuracy drop, compared with clean accuracy (obtained by training on clean data only).

4.6.2 Fine-tuning

We report our main results for poisoning the fine-tuning downstream task in Table 4.1. We expand discussions and our key observations in the following paragraphs for input space attacks and feature-targeted attacks, respectively.

4.6.2.1 Input space attacks

Recall in Section 4.4, we introduce the TGDA input space attack and the GC input space attack. We first examine the two attacks without considering any constraints, such that

[5] https://github.com/watml/TGDA-Attack
[6] https://github.com/watml/plim
[7] https://github.com/TobiasLee/ParamCorruption
[8] https://github.com/HanxunH/Unlearnable-Examples
[9] https://github.com/mkisantal/backboned-unet

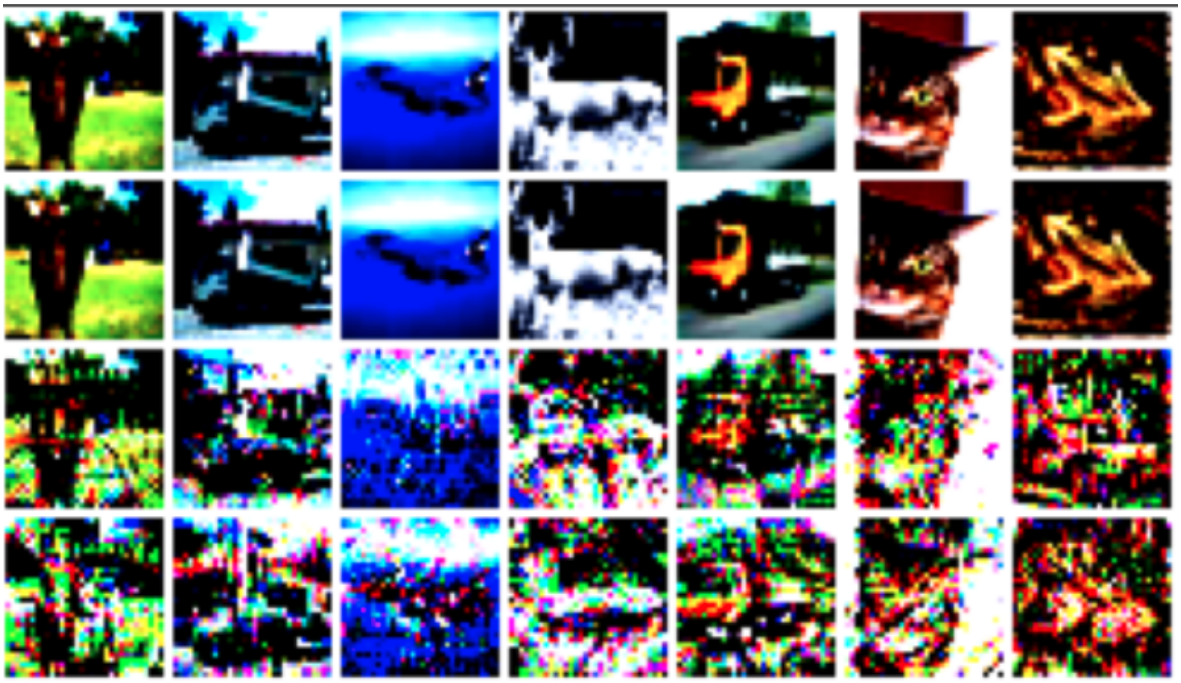


Figure 4.5: Here we visualize clean images (first row), and poisoned samples returned by the feature matching algorithm with $\beta = 0.25, 0.1, 0.05$ respectively from the second to the fourth row.

the poisoned points ν could be randomly large and do not need to resemble the real distribution μ. Although under this setting the poisoned samples can be easily detected, it is still valuable to study as it reveals the security threats without any data sanitization process, which is not uncommon in practice. From Table 4.1 (upper part, row 1,2,4), we observe that compared to TGDA, GC is very effective, where it can almost reach the target parameter generated by GradPC when $\varepsilon_d = 0.1$.

However, GC generates poisoned samples that have a much larger magnitude than the clean samples, especially when ε_d is small (e.g., $100\times$ larger for $\varepsilon_d = 0.03$ and decreases to $5\times$ larger for $\varepsilon_d = 1$). This also matches the observations in Chapter 3: a higher ε_d is desired for an attacker as it not only increases the attack effectiveness but also brings the poisoned points closer to the clean distribution, at least magnitude-wise.

Additionally, Chapter 3 shows that the GC attack does not change the semantics of the clean samples much, even without explicit constraints for end-to-end training. However, we show it is not the case for a fixed feature extractor in Figure 4.2. Instead, GC input space attack returns noisy samples that lose all information of the input sample.

Based on the above two observations, we further add two constraints to the two attacks to make them less vulnerable to possible defenses: (1) clipping the poisoned samples to the range of clean samples; (2) adding a term on measuring the (squared) L_2 distance between input (clean sample as initialization) and output (poisoned points) specified in Section 4.4. In Table 4.1 (upper part, row 3,5), we find the attacks are much less effective, especially

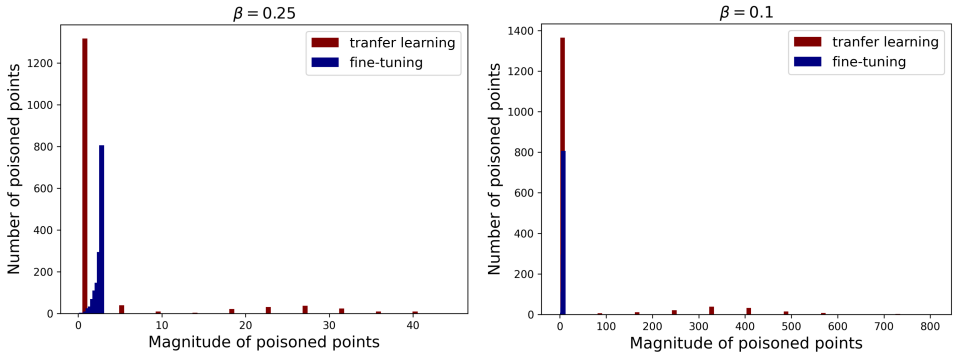


Figure 4.6: The histogram of the (L_∞) magnitude of 1500 ($\varepsilon_d = 0.03$) poisoned points generated by feature matching feature matching on transfer learning (red) and fine-tuning (blue) with $\varepsilon_d = 0.03$ for $\beta = 0.25$ (left), $\beta = 0.1$ (right). The plot shows that while for fine-tuning the poisoned points are roughly in the same range, for transfer learning, $\approx 10\%$ of the poisoned samples are anomalies.

the GC attack, and may not be considered a strong threat. Compared with poisoning end-to-end training in Chapter 3, our poisoned samples on fixed feature extractors are much weaker with constraints, e.g., Chapter 3 reports the test accuracy drop decreases $\approx 10\%$ for $\varepsilon_d = 0.03$, which is significantly less than ours (29.54% (no constraints) - 2.50% (with constraints) = 27.04%). This confirms that with a fixed feature extractor, it is much more difficult to construct feasible poisoning samples than end-to-end training for input space attacks, which motivates us to design and examine stronger attacks.

Table 4.2: Transfer learning results: comparison between input space attacks (TGDA and GC input space attacks with/without constraints on poisoned samples) and feature targeted attacks (Decoder inversion and Feature Matching with different β) w.r.t. different ε_d. The white background indicates methods that generate target parameters or features; blue indicates input space attacks and pink indicates feature targeted attacks.

Attack method	Clean	$\varepsilon_d = 0.03$ (1500)	$\varepsilon_d = 0.1$ (5000)	$\varepsilon_d = 0.5$ (25000)	$\varepsilon_d = 1$ (50000)
GradPC (input space)	93.03%	49.39% / -43.64%	49.39% / -43.64%	49.39% / -43.64%	49.39% / -43.64%
TGDA-input (no constraints)	93.03%	87.84% / - 5.19%	86.97% / - 6.06%	78.74% / -14.29%	72.93% / -20.10%
TGDA-input (with constraints)	93.03%	89.71% / - 3.32%	89.44% / - 3.59%	85.25% / - 7.78%	79.97% / -13.06%
GC-input (no constraints)	93.03%	53.68% / -39.35%	52.84% / -40.19%	50.04% / -42.99%	49.48% / -43.55%
GC-input (with constraints)	93.03%	88.63% / - 4.40%	87.44% / - 5.59%	79.52% / - 13.51%	76.05% / -16.98%
GradPC (feature space)	93.03%	49.53% / -43.50%	49.53% / -43.50%	49.53% / -43.50%	49.53% / -43.50%
GC-feature space	93.03%	50.01% / -43.02%	50.01% / -43.02%	50.01% / -43.02%	50.01% / -43.02%
Decoder inversion	93.03%	87.92% / - 5.11%	86.96% / - 6.07%	79.27% / -13.76%	74.06% / -18.97%
Feature matching ($\beta = 0.25$)	93.03%	71.50% / -21.53%	65.04% / -27.99%	57.93% / -35.10%	50.69% / -42.34%
Feature matching ($\beta = 0.1$)	93.03%	59.58% / -33.45%	57.74% / -35.29%	50.74% / -42.29%	49.63% / -43.40%

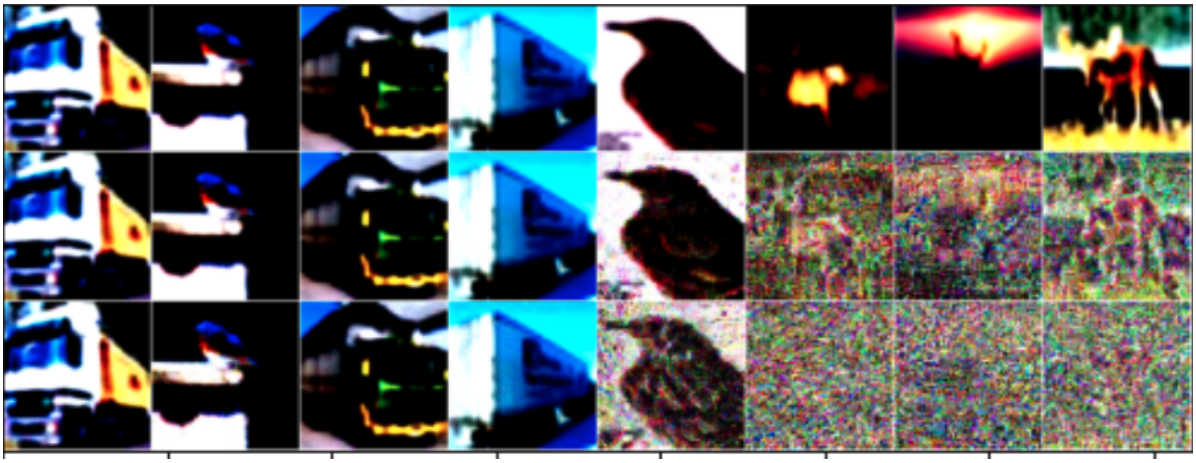


Figure 4.7: We plot poisoned images generated by feature matching on transfer learning for $\varepsilon_d = 0.03$. We show clean images and poisoned images for $\beta = 0.25$ and $\beta = 0.1$ on rows 1,2,3 respectively. Columns 1-4 are images within the normal range, and columns 5-8 are abnormal poisoned images with a bigger magnitude.

4.6.2.2 Feature targeted attacks

Next, we discuss two feature targeted (FT) attacks introduced in Section 4.5, namely the decoder inversion and the feature matching attack. Both of the FT attacks involve three stages: (1) GradPC feature space attack for acquiring the target parameters $\hat{\boldsymbol{\omega}}$, where we achieve roughly the same performance as that in input space; (2) GC feature space attack for obtaining the target feature ζ; (3) taking ζ as the input, we either invert the feature back directly using a decoder or perform feature matching.

Note that we do not treat GC feature space attack as a viable attack (as injecting poisoned features may not be realistic), but as an essential building block and intermediate step for decoder inversion and feature matching. Additionally, our following results for GC feature space attack provide important information on how good a poisoned feature ζ can be (in terms of attack performance) and the difficulty of inverting them back to the input space.

In Table 4.1 (lower part, rows 1-2), we report GradPC (feature space) for stage 1 and GC-feature space for stage 2. We observe that the GC feature space attack is very successful in reproducing the target parameters $\boldsymbol{\omega}$, even when $\varepsilon_d = 0.03$. Additionally, comparing with the GC input space attack, we observe that GC feature space induces a much smaller perturbation in terms of magnitude, e.g., the poisoned features are $4\times$ larger than the normal ones for $\varepsilon_d = 0.03$ and lie within the same range for $\varepsilon_d \geq 0.2$.

Surprisingly, although GC feature space attack obtains the target feature ζ with relatively small perturbations, it is difficult to invert it back to the input space. For decoder inversion (row 3), although we observe that the images are in the normal range and visually benign (see Figure 4.4), it is much less effective than the target poisoned features. In

practice, we find the poisoned samples are largely dominated by the skip connections of the encoder architecture, and thus the constraint on the input space could be too strong and cannot be easily controlled.

In comparison, the feature matching algorithm (rows 4-6) is more effective than decoder inversion, i.e., for $\beta = 0.25$, we observe a further accuracy drop $\approx 2\%$ for every choice of ε_d. Moreover, by controlling the hyperparameter β, we can easily tune the constraints on the input similarity, where a smaller β indicates less power of constraints. In Table 4.1, we observe that a smaller β generally returns a much stronger attack. However, tuning β also involves a trade-off: in Figure 4.5, we visualize the poisoned images generated by feature matching attacks with different β and observe that a smaller β generally introduce more noise, and poisoned images generated by $\beta = 0.1, 0.05$ can be easily discriminated by human eyes. Additionally, smaller β induce poisoned images with a larger magnitude, i.e., for $\beta = 0.25, 0.1, 0.05$, we observe the magnitude of $1\times, 3\times, 5.5\times$ the normal range, respectively.

In summary, we find both of the feature targeted attacks consistently outperform input space attacks across all choices of ε_d. However, for indiscriminate attacks in general, inverting poisoned features back to the input space attack is rather difficult, and cannot be achieved exactly with our existing attacks.

4.6.3 Transfer Learning

Next, we present our experimental results for the task of transfer learning. Recall that for transfer learning, we take the feature extractor (ResNet-50) pre-trained on ImageNet without labels and perform a linear evaluation on the CIFAR-10 dataset with labels. We report our main results in Table 4.2 and expand our discussions as follows.

4.6.3.1 Input space attacks

Firstly we perform the same input space attacks on the transfer learning task. In Table 4.2 (upper part), we observe: (1) In general, input space attacks are more effective (an increased accuracy drop $\approx$ 2%-10%) on transfer learning than on fine-tuning tasks. A large factor could be that transfer learning involves a distribution shift, such that the linear evaluation process is more vulnerable to poisoning attacks; (2) without any constraints, GC input space still outperforms TGDA input space. Meanwhile, GC generates poisoned samples that have a larger magnitude than that of fine-tuning, i.e., about $290\times$ larger for $\varepsilon_d = 0.03$ (fine-tuning: $100\times$ larger) than clean samples, and reduces to $14\times$ larger for $\varepsilon_d = 1$ (fine-tuning: $5\times$ larger). As for poisoned samples, TGDA and GC input space attacks still generate noisy images that have no semantic meanings (analogous to that in Figure 4.2); (3) again we add the same constraints on magnitude and visual similarity with the input images and observe that the attack performance significantly drops.

Table 4.3: Comparison between feature matching and feature matching after removing anomaly points (feature matching-r).

Attack method	Clean	$\varepsilon_d = 0.03$
Feature matching ($\beta = 0.25$)	93.03%	71.50% / -21.53%
Feature matching-r ($\beta = 0.25$)	93.03%	85.82% / - 7.21%
Feature matching ($\beta = 0.10$)	93.03%	59.58% / -33.45%
Feature matching-r ($\beta = 0.10$)	93.03%	86.38% / - 6.65%

4.6.3.2 Feature targeted attacks

We perform the same feature targeted attacks and report the results in Table 4.2 (lower part). For the feature matching attacks, we omit the results for $\beta = 0.05$ as it does not show a substantial improvement over $\beta = 0.1$. We observe that in general, similar to input space attacks, the attacks are more effective on transfer learning tasks than on fine-tuning. Additionally, decoder inversion and feature matching still outperform all input space attacks considering constraints across all choices of ε_d.

Notably, upon further exploration, we observe that feature matching behaves differently in transfer learning. Recall that for fine-tuning when we choose $\beta = 0.25$, the magnitude of poisoned samples is within the normal range, and for $\beta = 0.1$, the magnitude is generally $3\times$. However, for transfer learning, the algorithm returns several abnormal images with a larger magnitude and more noise. In Figure 4.6, we randomly select 100 poisoned samples (out of 1500 for $\varepsilon_d = 0.03$) and plot their L_∞ magnitude. For the entire poisoned set, we observe that for $\beta = 0.25$ and $\beta = 0.1$, there are 181 (12.1%) and 274 (18.2%) anomalies, respectively. Additionally, when β is smaller, while most of the poisoned points are in the normal range, the anomalies have a much bigger magnitude, due to the power of the constraint being weakened. In Figure 4.7, we compare poisoned images within the normal range (columns 1-4) and abnormal ones (columns 5-9). We observe that abnormal poisoned images also contain more noise, especially when β is smaller. To examine the real effect of feature matching attacks, we perform a simple data sanitization process by removing all of the abnormal points before evaluation. We report the results in Table 4.3 and observe the attack becomes much less effective after the defense. Nevertheless, feature matching-r ($\beta = 0.25$) still outperforms other baseline attacks.

Overall, compared to fine-tuning, we find transfer learning easier to poison for every attack, with or without constraints in the input space. However, inverting poisoned features back to input space, either explicitly through feature targeted attacks or implicitly through input space attacks, is still difficult in the condition of considering poisoned points in a feasible set. Specifically, we find that no existing attack can exactly reproduce the target parameters (features) with a limited number of visually benign poisoned samples.

Table 4.4: Comparison between UE (EMN) in the context of poisoning a fixed feature extractor with linear head and training end-to-end (with the same initialization). We report the results on both fine-tuning and transfer learning.

Task	**Attack**	Clean	$\varepsilon_d = \infty$
Fine-tuning	Original	94.77%	19.93% / -74.84%
	End-to-end	95.22%	68.79% / -26.43%
	Input space	89.71%	82.16% / - 7.55%
Transfer Learning	Original	94.42%	18.89% / -75.53%
	End-to-end	95.09%	66.05% / -29.04%
	Input space	93.03%	85.15% / - 7.88%

4.6.4 Unlearnable Examples

Finally, we discuss the unlearnable examples of poisoning fixed feature extractors (the EMN input space attack). We apply the attack of H. Huang et al. (2021) and report the results in Table 4.4. We compare three variants of EMN: (1) original: we consider the original setting in H. Huang et al. (2021), where during the attack and evaluation, the entire model is updated with random initialization on the unlearnable dataset[10]; (2) end-to-end: the model is initialized with the pre-trained feature extractor, but all parameters can be updated; (3) input space: we consider our setting where the feature extractor is fixed during training and evaluation. We visualize the poisoned images generated by EMN in Figure 4.8.

From Table 4.4 we obtain several interesting observations: (1) EMN is highly sensitive to weights initialization: by comparing original and end-to-end, we show that random initialization is much more vulnerable to EMN than a good initialization (i.e., pre-trained weights by contrastive learning); (2) EMN input space is not very effective: with a significantly larger poisoning budget $\varepsilon_d = \infty$, EMN only performs similar to feature matching when $\varepsilon_d = 0.03$. Compared with original and end-to-end, EMN input space is also less than a magnitude effective for both fine-tuning and transfer learning.

In summary, our observations reveal that training a neural network architecture with a good feature extractor as initialization, especially when it is frozen, is much more robust to the EMN attack.

4.7 Conclusions and Limitations

In this work, we study indiscriminate data poisoning attacks on pre-trained feature extractors with different budgets, from injecting a small amount of poisoned data to perturbing the entire training set. Specifically, we consider the model parameters acquired by

[10]The results are directly obtained from Table 1 in H. Huang et al. (2021), CIFAR-10 ResNet-18/50.



Figure 4.8: We plot poisoned images generated by EMN input space attack. We show clean images and poisoned images for fine-tuning (which incurs an accuracy drop of 7.55%) and transfer learning (incurring an accuracy drop of 7.88%) on rows 1,2,3, respectively.

contrastive learning methods and poisoning the downstream task with a trainable linear head and the fixed feature extractor. Based on the attack scheme, we propose two types of indiscriminate attacks: (1) input space attacks, including TGDA, GC, and UE input space attacks, where the attacker perturbs the feature representations indirectly through the input space, and (2) feature-targeted attacks, including decoder inversion and feature matching attacks, where an adversary acquires poisoned features and inverts them back to the input space.

We further examine our attacks on popular downstream tasks of fine-tuning and transfer learning. Our experimental results reveal that for injecting limited numbers of poisoned samples, input space attacks generate out-of-distribution poisoned samples without constraints and become less effective after projection. On the other hand, feature targeted attacks, especially feature matching attacks exhibit an improvement with a tunable trade-off between the legitimacy of the poisoned samples and their effectiveness. Moreover, considering a more challenging domain adaptation scenario, transfer learning is more vulnerable than fine-tuning on these attacks in general. Additionally, empirical results on unlearnable examples demonstrate that EMN is much less effective than it performs on end-to-end training with random initialization.

Limitations and Future directions: In this chapter, we set baselines on indiscriminate attacks on pre-trained feature extractors and we believe it opens several interesting future directions: (1) Understanding the difficulty of feature inversion: prior works on adversarial examples show that small perturbation in the input space could lead to dramatic changes in the feature space for a neural network with large Lipschitz constant. In this chapter,

we show that a relatively small perturbation in the feature space cannot be easily inverted to the input space with constraints. Such observations might be counterintuitive and surprising, and deserve more study; (2) Possible defenses against Unlearnable Examples: although UE, e.g., EMN poses a strong indiscriminate attack, its countermeasure has not been explored. Notably, data sanitization cannot be easily performed as the entire training distribution is altered. In this chapter, we provide evidence that UE is fragile against fixed feature extractors and a well-trained initialization. Based on this, there might exist a natural defense mechanism against UE, by carefully designing the architecture or the training schemes, e.g., adding a fixed encoder for any task of interest.

Chapter 5

Disguised Copyright Infringement of Latent Diffusion Models

5.1 Preface

Throughout the preceding chapters, we have conducted an in-depth analysis of indiscriminate data poisoning attacks and their detrimental effects on overall model integrity for machine learning developers. In this final chapter, we pivot our focus to examine implications for individual data owners. Specifically, we demonstrate how strategically crafted data can transcend conventional poisoning effects and potentially introduce significant copyright infringement concerns.

Abstract Copyright infringement may occur when a generative model produces samples substantially similar to some copyrighted data that it had access to during the training phase. The notion of access usually refers to including copyrighted samples *directly* in the training dataset, which one may inspect to identify an infringement. We argue that such visual auditing largely overlooks a concealed copyright infringement, where one constructs a disguise that looks drastically different from the copyrighted sample yet still induces the effect of training Latent Diffusion Models on it. Such disguises only require *indirect access* to the copyrighted material and cannot be visually distinguished, thus easily circumventing the current auditing tools. In this chapter, we provide a better understanding of such disguised copyright infringement by uncovering the disguises generation algorithm, the revelation of the disguises, and importantly, how to detect them to augment the existing toolbox. Additionally, we introduce a broader notion of *acknowledgment* for comprehending such *indirect access*.

The following sections are adapted from the main paper and appendix of Y. Lu, M. Y. Yang, Z. Liu, G. Kamath, and Y. Yu (2024). "Disguised Copyright Infringement of Latent Diffusion Model". In: *International Conference on Machine Learning*. PMLR.

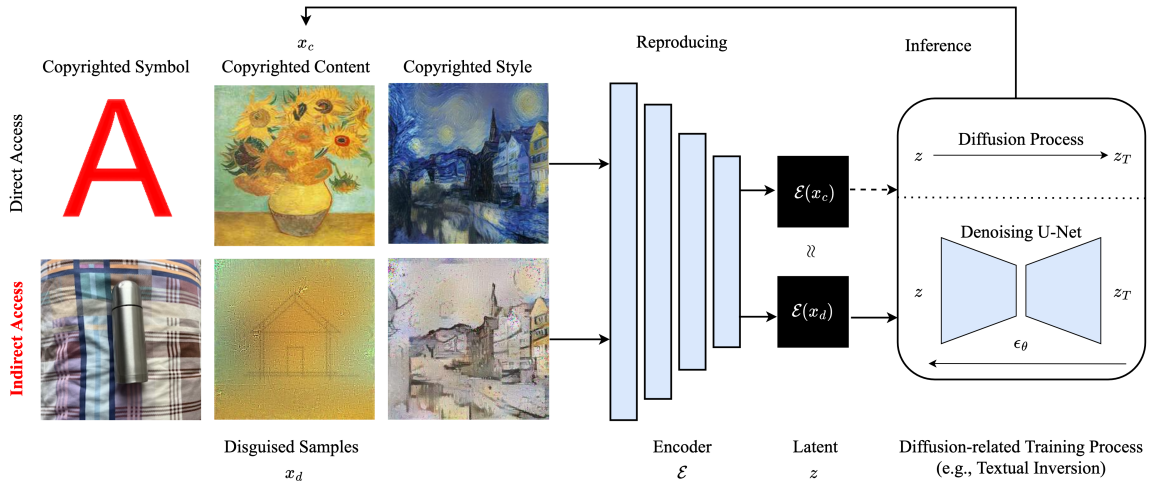


Figure 5.1: An overview of conventional (with *direct access* to copyrighted material) and disguised (with *indirect access*) copyright infringement for latent diffusion models. For *direct access*, training an LDM-based model on copyrighted material x_c and reproducing x_c is subject to copyright infringement. For *indirect access*, one trains the same model on disguised samples x_d, which are drastically different from x_c, but are still able to reproduce x_c during inference.

5.2 Introduction

Generative models, especially the recent advanced Latent Diffusion Models (LDM) (Rombach et al. 2022), have shown tremendous ability to generate new images, even of creative or artistic form according to text prompts. Such models are trained on a large corpus of data, which may consist of copyrighted material. Additionally, prior works have established that such generative models are prone to regurgitating content from their training data (Carlini, Ippolito, et al. 2022; Ippolito et al. 2023; Somepalli et al. 2023a,b; Vyas et al. 2023; C. Zhang et al. 2021), which may also be copyrighted.

In this chapter, we will focus on copyright law within the jurisdiction of the United States. To establish a copyright violation, two factors *must* be present. First, the accused must have had access to the copyrighted material. Second, the accused must produce content that bears "substantial similarity" to the copyrighted material (reproducing). Note that the definition of substantial similarity can be ambiguous. Within the context of images, its definition appears to be relatively broad (*Steinberg v. Columbia Pictures Industries, Inc.* 1987), and in particular encompasses near-exact copies.

Turning our attention to the former "access" criterion: the natural way to establish that a model had access to a particular piece of copyrighted material is to inspect its training data. For example, in the case of Andersen v. Stability AI Ltd. (Dist. Court 2023), the case was allowed to proceed based on the fact that copyrighted images were found in LAION-5B (Schuhmann, Beaumont, et al. 2022) (the training data used for Stable Diffusion) using

haveibeentrained.com.

We challenge the perspective that such visual auditing is *sufficient* to establish access to copyrighted material. Our results show that it is possible to *conceal* copyrighted images within the training dataset for LDMs. Specifically, LDMs are equipped with a fixed encoder for dimension reduction such that the diffusion learning process occurs in the latent space. This structure can be maliciously exploited to generate disguised copyrighted samples: given a copyrighted image, we show how to generate a disguise such that it is visually different from the copyrighted sample but shares similar latent information. The closeness of the two samples in the latent space can be quantitatively measured by a distance function, or qualitatively revealed by a concept extraction tool called textual inversion (Gal et al. 2022), both of which we will demonstrate in our empirical study.

Our study reveals the possibility of creating a new training dataset that does not appear to *directly* or blatantly contain any copyrighted data. Nonetheless, if a model is trained on this derivative training dataset, it would behave similarly as if the copyrighted data were present. Such disguises may still exhibit copyright infringement, although only accessing proprietary data *indirectly*. In Figure 5.1, we display a comparison between the previous copyright infringement phenomenon (*direct access*) with the disguised copyright infringement (*indirect access*). Clearly, there was still access to the copyrighted material in the latter training pipeline, which raises the following question:

What constitutes access? How to quantify it?

We answer the first question by introducing a notion of *acknowledgment*, which refers to a criterion that any sample that contains similar latent information as that of a copyrighted sample should be considered *acknowledging* it, despite possible visual dissimilarity. To quantify *acknowledgment* in practice, a deeper inspection than visually auditing the training set is required. Thus we further propose a two-step detection method: (1) a feature similarity search for screening suspects; (2) an encoder-decoder examination to confirm disguises, which augments the existing criterion. In summary, we make the following contributions:

- We challenge the current "access" criterion and point out its insufficiency in more delicate cases of copyright infringement;
- We propose an algorithm that demonstrably crafts disguised copyrighted data to conceal the content (or concepts) of copyrighted images in the training set;
- We show disguised data contain copyrighted information in the latent space, such that by finetuning them on textual inversion or DreamBooth, or training on LDM, the model reproduces copyrighted data during inference;
- We propose methods to detect such disguises, which further encourage the expansion and quantification of "access" in the context of copyright infringement.

5.3 Background

5.3.1 Diffusion Models

We focus on latent diffusion models (Rombach et al. 2022) as they are ideal for text-to-image generation. We first recall the objective of regular diffusion models (Sohl-Dickstein et al. 2015):

$$\mathcal{L}_{\texttt{DM}} = \mathbb{E}_{x,\varepsilon,t}\Big[\|\varepsilon - \varepsilon_\theta(x_t, t)\|_2^2\Big],$$

where $\varepsilon \sim \mathcal{N}(0, \mathbb{I})$, t is the timestep uniformly sampled from $\{1, ..., T\}$, and x_t is a noisy version of the input sample x at timestep t . Briefly, diffusion models are probabilistic models designed to learn a data distribution $p(x)$ by gradually denoising a normally distributed variable, which corresponds to learning the reverse process of a fixed Markov chain of length T.

Note that the intermediate x_t are all in pixel space, which makes the training and inference of diffusion models expensive. To address this problem, Rombach et al. (2022) propose to perform the diffusion process in the latent space, namely latent diffusion models (LDM). Specifically, LDMs utilize a pre-trained (and fixed) autoencoder architecture which consists of an encoder $\mathcal{E}$ and a decoder $\mathcal{D}$, where $\mathcal{E}$ is only used during training and $\mathcal{D}$ is only used during inference. The objective of LDM can be expressed as:

$$\mathcal{L}_{\texttt{LDM}} = \mathbb{E}_{x,\varepsilon,t}\Big[\|\varepsilon - \varepsilon_\theta(\mathcal{E}(x_t), t)\|_2^2\Big].$$

When we perform text-to-image tasks, a text condition (or prompt) y needs to be considered. Such a text condition is compressed into the latent space with a pre-trained text-embedding model $c_\theta(\cdot)$ (typically a BERT (Devlin et al. 2018) text encoder) and fed into LDM training as a conditioning vector:

$$\mathcal{L}_{\texttt{LDM-c}} = \mathbb{E}_{x,y,\varepsilon,t}\Big[\|\varepsilon - \varepsilon_\theta(\mathcal{E}(x_t), t, c_\theta(y))\|_2^2\Big]. \tag{5.1}$$

Intuitively, LDMs perform the diffusion training process on a compressed version of the training data, while the compression procedure is deterministic. As a result, it is possible that two training samples share similar latent representations, but differ significantly visually. Based on this intuition, we propose a method to generate disguised data, demonstrating that the use of pre-trained autoencoders may *conceal* the inclusion of copyrighted data in the training set.

5.3.2 Data Poisoning attacks

Copyright infringement can be regarded as a special case of data poisoning, where one includes some "poisoned data" as a subset of a training set χ, such that after training an

LDM on χ, the model reproduces a copyrighted sample x_c. For *direct access*, "poisoned data" simply refers to direct copies of x_c. In this chapter, we also examine *indirect access*, where one can construct the "poisoned data" so they are visually different from x_c using an adaptation of the data poisoning attack in Shafahi et al. (2018).

5.3.3 Textual Inversion

Art generation and related copyrighted material reproduction frequently happen in LDM-based generative tools. For example, textual inversion (Gal et al. 2022) is a popular tool for modifying personalized images with language-guided LDMs. Textual inversion uses a small set of images (typically three to five), which depict the target concept S_* (which is a placeholder in a text prompt, e.g., "A photo of S_*") across multiple settings such as varied backgrounds or poses. The concept S_* is tokenized, converted to an embedding ν_*, and then encoded as part of the text embedding $c_\theta(y)$. The embedding ν_* is acquired by optimizing the LDM loss in equation 5.1 over the training images:

$$\nu_* = \underset{\nu}{\operatorname{argmin}} \mathcal{L}_{\texttt{LDM-c}} \tag{5.2}$$

Optimization is performed using the same training scheme as the original LDM while keeping ε_θ, c_θ and $\mathcal{E}$ fixed. After obtaining ν_*, it can be combined with various text prompts (typically derived from the CLIP ImageNet templates (Radford et al. 2021)) for generation. Overall, textual inversion is a great tool for capturing particular concepts and may also be used to reproduce copyrighted content.

5.4 Generating Disguises

We describe how to *conceal* copyrighted images within the training dataset. Specifically, we demonstrate that one can construct a disguised sample x_d that is visually distinct from a copyrighted sample x_c, but contains essentially similar information in the latent space (measured via the distance between feature representations). Specifically, we first recall the feature matching attack (Shafahi et al. 2018) designed for targeted data poisoning attacks, and then discuss how to adapt the attack to generate disguises.

5.4.1 Feature Matching Attack

We recall that targeted attacks aim to change the prediction (e.g., causing misclassification) of a model (typically a classifier) on a targeted test sample x_t (with label y_t) by training on a mixture of the existing clean data and poisoned data. Specifically, Shafahi et al. (2018) propose a feature matching attack, where a poisoned sample x_p is acquired by making imperceptible changes to a base sample x_b with label y_b (different than y_t) such that the

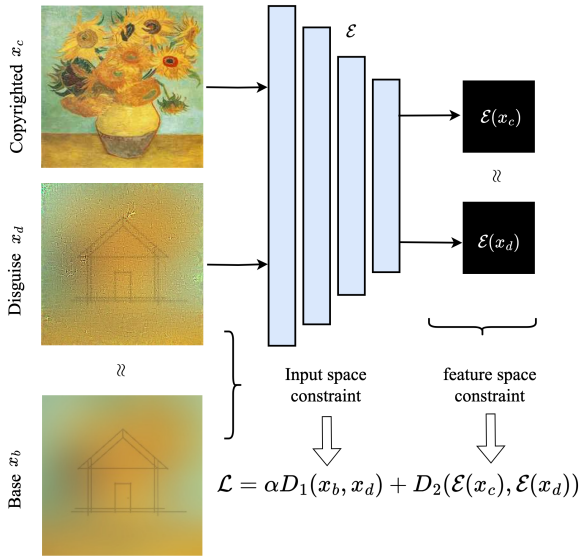


Figure 5.2: An illustration on the algorithm to generate disguises. We aim to optimize the loss $\mathcal{L}$ consisting of an input space constraint that measures the distance between the base image x_b and the disguise x_d in the input space, and a feature space constraint that measures the distance between the copyrighted x_c and x_d in the feature space.

feature representation of the poisoned sample $f(x_p)$ matches that of the target sample $f(x_t)$, where the model f is a (usually fixed) pre-trained feature extractor. By training the model (e.g., with a softmax layer on top of f) on x_p, it is likely to misclassify the target sample x_t as the wrong (base) label y_b.

Note that the generality of the feature matching attack is restricted in the context of classification tasks because it is only applicable when the feature extractor f is fixed, which may only be applicable for fine-tuning or transfer learning. However, we will show how it can be used to craft disguised copyrighted images in the next paragraph.

Adapting to copyright infringement: Training LDMs on a dataset χ amounts to performing a regular diffusion learning process on a latent dataset ζ, where $\zeta = \mathcal{E}(\chi)$. Suppose there exists a set of N copyrighted data $\{x_c^i\}_{i=1}^N \subseteq \chi$, which we aim to substitute with the same amount of disguised data $\{x_d^i\}_{i=1}^N$ such that the latent dataset ζ stays intact. Compared with data poisoning, the target sample x_t is set to x_c, the feature extractor f is set to $\mathcal{E}$, the base sample can be any uncopyrighted image different from x_c, the sample we aim to optimize is x_d, and the feature matching algorithm can be adapted. Conveniently, the encoder model $\mathcal{E}$ is pre-trained and its weights are fixed during the training of diffusion, thus making the above attack realistic.

Algorithm 4: Disguise Generation

Input: copyrighted image x_c, base image x_b, pre-trained encoder $\mathcal{E}$, input threshold γ_1, feature threshold γ_2, distance measure on input space $D_1(\cdot)$, distance measure on feature space $D_2(\cdot)$, hyperparameter on input space constraint α, learning rate η.

Initialize disguise x_d with base image x_b
repeat
  $D_1 \leftarrow D_1(x_b, x_d)$ `// image distance`
  $D_2 \leftarrow D_2(\mathcal{E}(x_c), \mathcal{E}(x_d))$ `// feature distance`
  $\mathcal{L} \leftarrow \alpha D_1 + D_2$ `// calculate loss`
  $x_d \leftarrow x_d - \eta \frac{\partial \mathcal{L}}{\partial x_d}$ `// update disguise`
  $x_d \leftarrow \text{Proj}_\Gamma(x_d)$ `// project to admissible set`
until $D_1 \leq \gamma_1$ *and* $D_2 \leq \gamma_2$
return disguise x_d

5.4.2 Disguised Copyrighted Image Generation

Now we introduce our disguise generation algorithm for individual images. Formally, given a target copyrighted image x_c, a base image x_b which is chosen to be visually different from x_c, distance measures $D_1(\cdot)$ and $D_2(\cdot)$ for input space and latent space respectively, and a fixed pre-trained encoder $\mathcal{E}$, we aim to construct a disguised image x_d such that it satisfies:

$$D_1 = D_1(x_b, x_d) \leq \gamma_1, D_2 = D_2(\mathcal{E}(x_c), \mathcal{E}(x_d)) \leq \gamma_2,$$

where γ_1 is the input threshold that measures the visual distance of x_d compared with x_b, γ_2 is the threshold that measures the feauture distance between x_c and x_d , which can be both determined empirically. We can then express the objective function as follows:

$$\underset{x_d}{\text{argmin}}\, \alpha D_1 + D_2,$$

where α is a tunable hyperparameter for controlling the tradeoff. To illustrate more, the extreme case $\alpha = 0$ refers to the scenario where there is no input space constraint on x_d while $\alpha = \infty$ refers to no feature space constraint. Our algorithm is summarized in Algorithm 4 and Figure 5.2.

Connection to Nightshade: Shafahi et al. (2018) introduced the idea of attacking a model by manipulating training datapoints so that their feature representation resembles that of a different point. They applied this principle to targeted data poisoning attacks against classifiers, whereas Shan et al. (2024) (in their work "Nightshade") and we apply it to LDMs. To highlight some of the differences between our work and that of Shan et al. (2024), there are crucial technical differences in how we instantiate this algorithm. Specifically, Nightshade intends to *stop* LDM from producing the image containing the



(a) x_c (b) Disguises x_d



(c) Inference on textual inversion by learning on x_d only

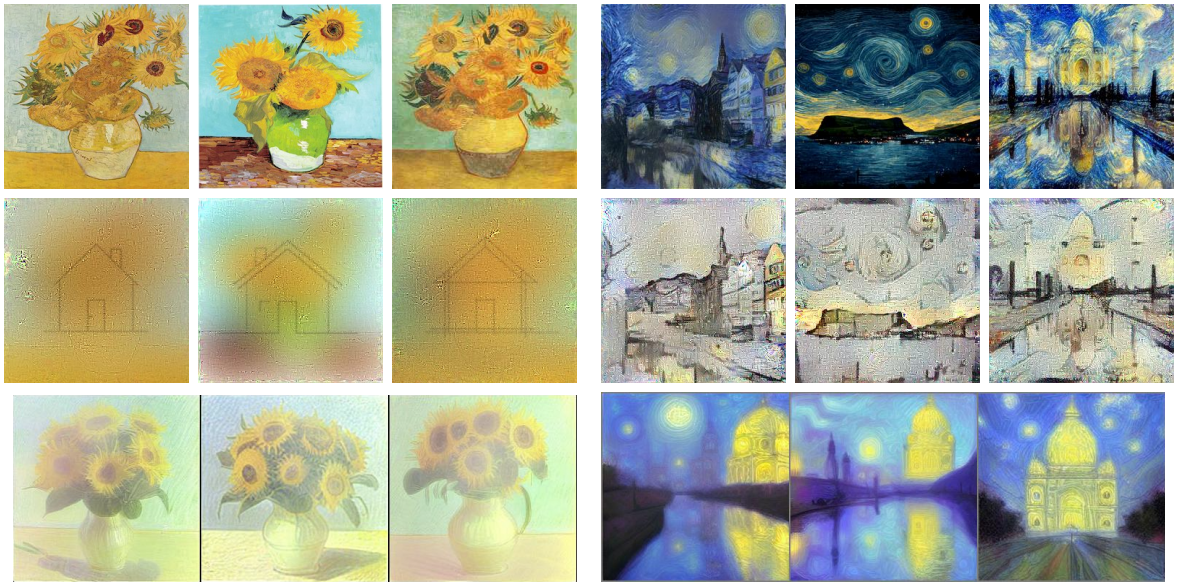
Figure 5.3: The disguised symbol on textual inversion. The first row from left to right: Column (1) the designated copyrighted symbol; Columns (2)-(5) four disguises x_d generated with different x_b. The second row: images generated by textual inversion after training on the above x_d.

correlated concept. In contrast, we aim to *urge* LDM to reproduce the target image (copyrighted image) through indirect access. Furthermore, the two works employ this algorithm towards very different ends. Nightshade aims to sabotage the connection between an image and its corresponding text prompt, thus requiring a pair of (image, text) as input. In contrast, our method only requires the form of images as input.

5.5 Revealing Disguises

Recall that the set of disguised data $\{x_d^i\}_{i=1}^N$ is constructed to substitute their copyrighted counterpart $\{x_c^i\}_{i=1}^N$ such that the latent dataset ζ is not changed much. To evaluate whether this goal is accomplished, we extract the latent information contained in the acquired samples x_d using textual inversion to qualitatively *reveal* the disguises. We also report the feature threshold γ_2 for quantitative analysis.

Moreover, examination of textual inversion also demonstrates how disguised copyright infringement may occur in the wild, i.e., reproducing copyrighted materials using products that utilize pre-trained LDMs to generate arts, which bring major concerns for human artists. Such LDM-based methods usually require only a few images as the training set χ, and can be easily substituted by disguises entirely.



(a) (b)

Figure 5.4: **(a)** Disguised copyrighted content on textual inversion. The first row: the designated copyrighted image x_c (*The Sunflowers* by Vincent van Gogh); the second row: three disguises x_d generated with different x_b; the third row: images generated by textual inversion after training on the above x_d. **(b)** Disguised copyrighted style on textual inversion. The first row: the designated copyrighted style x_c (in the style of *The Starry Night* by Vincent van Gogh); the second row: disguises x_d generated with different x_b; the third row: images generated by textual inversion after training on the above x_d.

5.5.1 Experimental settings

Note that the "copyrighted" images x_c used in our experiments may not be actually copyrighted, but used as a substitute to demonstrate disguised copyright infringement.

LDM: We adopt the official PyTorch implementation[1] of conditional LDM Rombach et al. 2022 and acquire the pre-trained weights[2](including that of the encoder $\mathcal{E}$, the denoising U-Net ε_θ and the text embedding $c_\theta(\cdot)$) of a 1.45B parameter *KL*-regularized LDM-8 (8 denotes downsampling factor) model conditioned on language prompts on LAION-400M Schuhmann, Kaczmarczyk, et al. 2021. We apply the pre-trained encoder $\mathcal{E}$ in Algorithm 4 to generate disguises, and the entire pre-trained LDM for textual inversion. Note that the LDM is not re-trained in our experiments.

[1] `https://github.com/CompVis/latent-diffusion`

[2] `https://ommer-lab.com/files/latent-diffusion/nitro/txt2img-f8-large/model.ckpt`

Generating Disguises: Throughout our experiments, we apply the pre-trained *KL*-regularized encoder $\mathcal{E}$, and set the input distance measure $D_1(\cdot)$ as a sum of the multi-scale structural similarity index (MS-SSIM) loss (Z. Wang et al. 2003) and L_1 loss following the analysis of Khare et al. 2021, and the feature distance measure to be the L_2 loss: $D_2(\mathcal{E}(x_c), \mathcal{E}(x_d)) = \|\mathcal{E}(x_c) - \mathcal{E}(x_d)\|_2$. The choice of the copyrighted image x_c, base image x_b, the input threshold γ_1, the feature threshold γ_2 and the hyperparameter on the input constraint α are task-dependent (for different copyrighted material), and we specify them in their corresponding paragraphs below. We set the admissible set to be in the range of $[0, 1]$ as the legitimate (normalized) image pixel value and run the algorithm for 100000 epochs (early stop if the stopping criteria are reached) for all experiments.

Textual inversion: After acquiring the disguises x_d, we feed them (we generate $N = \{3, 4\}$ disguises for extracting the same concept, following the normal recipe) into textual inversion (Gal et al. 2022) and optimize the embedding ν_* with equation 5.2. We adopt the official PyTorch implementation[3], which utilizes the pre-trained *KL*-regularized LDM-8 with the same encoder $\mathcal{E}$. Note that training textual inversion requires an initial word as input, which is an initialization for the concept S_*. Throughout our experiments, we choose the initial word to match the visual appearance of the disguises (thus the concept of the base image) and we observe that changing the initial word does not lead to a significant difference for the algorithm. For generating new images, we select from the pool of prompts included in the existing implementation of textual inversion[4] which we specify for each task below.

5.5.2 Concepts learned on disguises

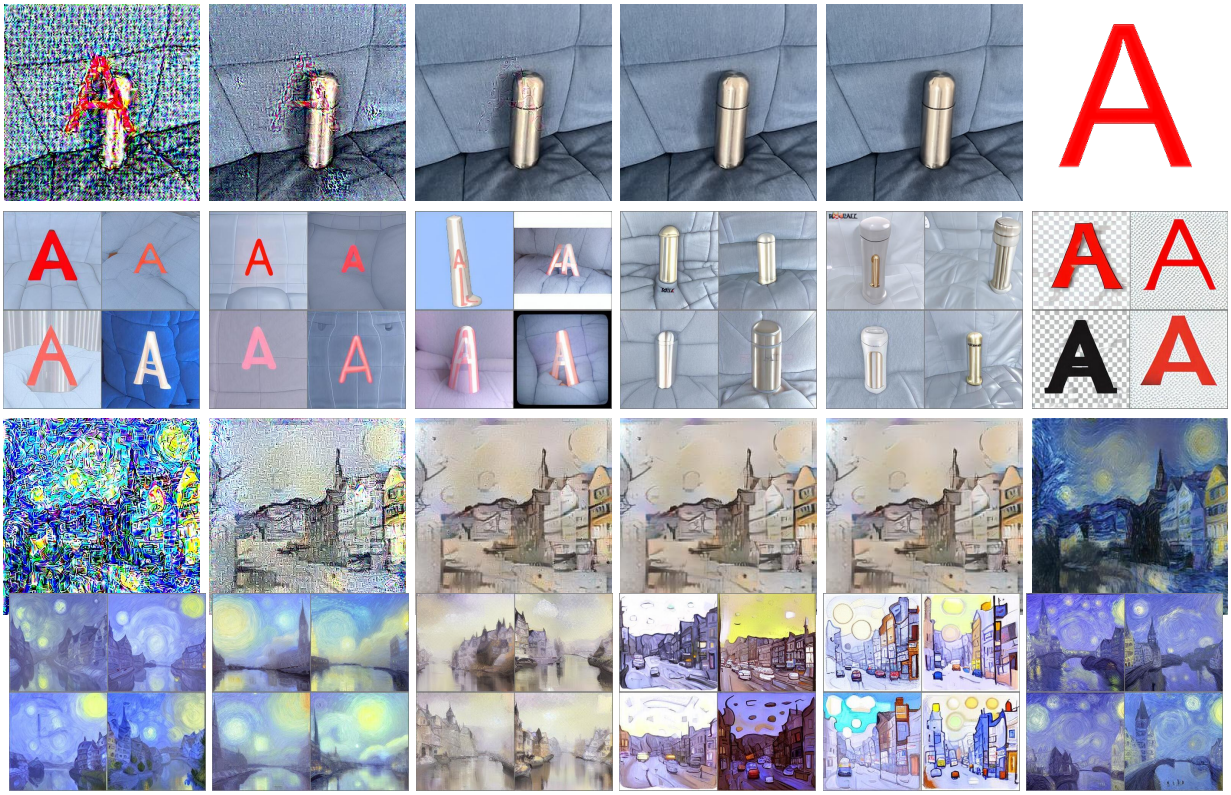
We demonstrate the concepts learned by textual inversion on the disguises with respect to different copyrighted patterns. (1) a copyrighted symbol; (2) copyrighted content; and (3) style. We note that although the style is generally *not* copyrightable, there are still ongoing debates about the *ethics* of using copyrighted artwork to train a model that can imitate an artist's style[5]. For this reason, we also examine the case of hiding an artist's style in disguised images.

Disguised symbol: We first show the easiest scenario to generate disguises, namely copyrighted symbol. In Figure 5.3, we pick the symbol "A" as x_c (note that here we abstract x_c as the symbol "A", in our experiment, it is combined with each base image to create the corresponding copyrighted image x_c) and choose four images of a water bottle as base images x_b. The base images do not need to be carefully constructed and are photos

[3] https://github.com/rinongal/textual_inversion

[4] https://github.com/rinongal/textual_inversion/blob/main/ldm/data/personalized.py

[5] https://news.bloomberglaw.com/ip-law/ai-imitating-artist-style-drives-call-to-rethink-copyright-law



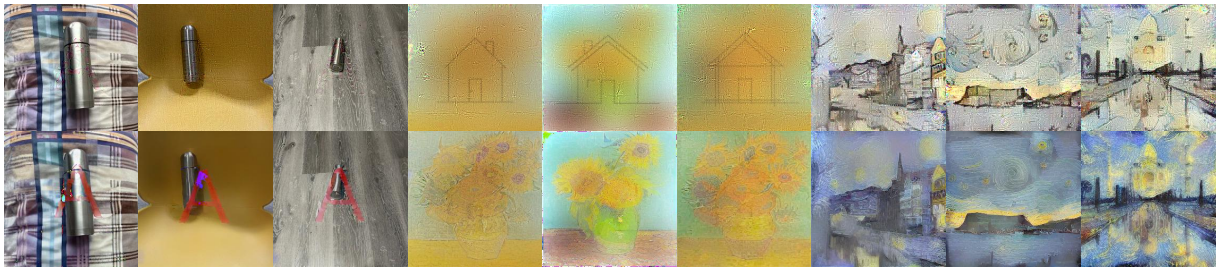
(a) $\alpha = 0$ (b) $\alpha = 1000$ (c) $\alpha = 8000$ (d) $\alpha = 64000$ (e) $\alpha = \infty$ (f) target

Figure 5.5: The effect of tuning the hyperparameter α on disguises (Rows 1-2: symbol; Rows 3-4: style). Rows 1 and 3 show disguises generated with different α; Rows 2 and 4 show the images generated by textual inversion by training on the disguises above. Note that the criterion on the input and feature threshold may not be satisfied in this series of experiments and we only train the disguise generation algorithm until convergence.

taken by ourselves. Each disguise x_d is created by taking a pair of x_c and x_b as input to Algorithm 4. We set $\gamma_1 = 0.05, \gamma_2 = 0.35$ (normalized to the range of $[0, 1]$) as the threshold of input and feature distance, respectively, and $\alpha = 8000$. By feeding x_d into textual inversion with the text prompt "a photo of a *", we reproduce the target symbol "A" without being exposed to the semantic information of the copyrighted content.

Disguised content We then show a more challenging task to generate disguised content. In Figure 5.4 (a), we pick three drawings of *The Sunflowers*[6] by Vincent van Gogh as x_c (first row). Due to the difficulty of the task, we cannot choose any image as x_b. Thus we first blur x_c such that they lose their semantic information and retain the color pattern,

[6]From left to right: F456 (*1888*), F453 (*1888*), F455 (*1889*).



(j) (k) (l) (m) (n) (o) (p) (q) (r)

Figure 5.6: Results for the encoder-decoder examination. The first row shows the disguises x_d and the second row includes the output of the autoencoder: $\mathcal{D}(\mathcal{E}(x_d))$. Columns j-l, m-o, and p-r show results for symbol, content, and style, respectively. The autoencoder reveals the copyrighted information contained in the disguises.

then we add simple sketches of houses[7] as base images x_b (second row). The background is essential for our purpose, where the disguises are ineffective with a white background. We set $\gamma_1 = 0.08, \gamma_2 = 0.26, \alpha = 4000$. By feeding x_d into textual inversion with the text prompt "a photo of a *", we recover the content of the sunflowers (third row) without containing the semantic information of x_d.

Disguised style Finally, we show the results for style scraping. In Figure 5.4(b), we pick three drawings[8] in the style of *The Starry Night* by Vincent van Gogh as x_c (first row). The base images x_b (second row) are the target images with another style (watercolor), generated with AdaIN-based (X. Huang and Belongie 2017) style transfer[9]. We set $\gamma_1 = 0.03, \gamma_2 = 0.33$, $\alpha = 2000$. By feeding x_d into textual inversion with the text prompt "a painting in the style of *", we reproduce the style of x_c (*The Starry Night*) while textual inversion only learns from images that visually resemble watercolor style.

In summary, for disguised symbols, generating the disguises x_d with any base image x_b is easy. However, for generating disguised content and style, one needs to choose the base image carefully for successful optimization.

5.5.3 Visual appearance of disguises

Next, we control the visual appearance of x_d by tuning α, which is the weighting parameter of the input space constraint. In Figure 5.5, we show the tradeoff of tuning the hyper-parameter α: (1) a smaller α indicates weaker input space constraint and it could lead

[7] These simple sketches of houses were generated by ChatGPT with the text prompt "create a very very simple line art of a very simple house in the middle with a white background"

[8] From left to right: The Neckarfront in Tubingen, Germany (photo by Andreas Praefcke); The Faroe Islands (courtesy Listasavn Føroya); Taj Mahal (adamkaz/Getty Images) in the style of *The Starry Night*, generated with Neural Style Transfer (Gatys et al. 2015).

[9] `https://github.com/tyui592/AdaIN_Pytorch`

to an ineffective disguise, where x_d still visually contains the copyrighted material; (2) in contrast, a bigger α shifts the optimization focus away from feature matching to generate latent embeddings distinct from the copyrighted content's. In the latter case, the tradeoff is that despite having a strong disguise that visually hides the copyrighted content, the textual inversion process may not successfully learn the copyrighted material, thus rendering our attack pointless. Note that there is no input constraint for the extreme case $\alpha = 0$, and the visual appearance of disguises largely depends on the initialization (we initialize with x_b above).

5.5.4 Detection

Next, we introduce a two-step detection method specifically for disguised samples that go beyond browsing through the training set, e.g., haveibeentrained.com.

(1) **Feature similarity search:** Quantitatively, a disguised sample has a similar feature representation with that of a copyrighted sample, i.e., $D_2(\mathcal{E}(x_c), \mathcal{E}(x_d)) \leq \gamma_2$. As we have provided the feature threshold γ_2 which is sufficient for replicating the copyrighted content for different tasks, one can use this as a reference threshold to detect possible disguises. Specifically, given an encoder $\mathcal{E}$ (which is usually easily accessible), a copyrighted image x_c which needs examination for infringement, one simply goes through the training set and computes $\mathcal{E}(x)$ for every single sample and compare with that of x_c. However, such a search alone only extracts suspects, which may not be true disguises. To rule out the sheer chance of collisions in the feature space, we perform another examination below.

(2) **Encoder-decoder examination:** The encoder architecture in LDMs is part of an autoencoder architecture (e.g., the KL-based VAE), where the encoder and decoder are used separately for encoding and inference. Consequently, the decoder $\mathcal{D}$ can be naturally used to detect disguises qualitatively. Specifically, for a well-trained autoencoder, $\mathcal{D}(\mathcal{E}(x_c)) \approx x_c$, while for disguises $\mathcal{E}(x_d) \approx \mathcal{E}(x_c)$, thus we have $\mathcal{D}(\mathcal{E}(x_d)) \approx x_c$. In Figure 5.6, we show that the encoder-decoder architecture is a great detection tool for disguises, where the output of the autoencoder reveals the copyrighted content hidden in x_d.

5.5.5 A broader definition of access

Our experiments have demonstrated that by generating disguises, one can indirectly acquire access to copyrighted material as part of the training data to fuel LDM-based models, which could reproduce the copyrighted samples during inference or deployment. We expand the definition of *direct access* to *acknowledgment* to cover such scenarios as possible copyright infringement: any training data x that contains similar latent information as that of copyrighted image x_c, measured by their similarity in the latent feature representation, even visually different from x_c, shall be considered to have an *acknowledgment* of x_c.

The notion of *acknowledgment* may also augment existing regulatory frameworks (e.g., the White House Executive Order on AI Biden 2023) on AI governance in terms of data quality evaluation. Additionally, our quantification of *acknowledgment* (the detection method) provides a timely tool for auditing beyond black-box access, which was pointed out to be insufficient in Casper et al. 2024.

5.6 Conclusion

In this chapter, we review the current access criterion of containing copyrighted material in the training set (*direct access*) in copyright infringement of generative models and point out its insufficiency by introducing disguised copyright infringement (*indirect access*). Specifically, such an infringement is realized by injecting disguised samples into the training set, which urges LDMs to produce copyrighted content. Such disguises are generated with a simple algorithm and demonstrated to share the same concept with their target copyrighted images using LDM-based applications. To alleviate the concern on the disguises, we expand the current visual auditing (browsing the training set) with additional tools, i.e., feature similarity search and encoder-decoder examination to better identify these disguises. Furthermore, we propose a broader definition of *acknowledgment* to cover this new type of copyright violation.

Limitations and future work: Although our algorithms can generate descent disguises, we believe there is still room for improvement for optimization. Moreover, one extension we didn't touch is the possibility of "chopping" copyrighted data and hiding it in several images. It is intriguing to explore whether it is possible to generate such a smuggler's dataset and detection towards it.

Impact Statements In this chapter, we adopt a simple algorithm to disguise images, which we realize may be applied as a concealed copyright infringement tool. We are surprised at how easy it is for Generative AI to circumvent the copyright law, which was originally designed for regulating human scrapers. This seemingly undetectable infringing threat may bring disadvantages for human artists, who are already losing the battle with GenAI. Furthermore, disguised images may not only be a concern to copyright owners but also a threat to Generative AI companies: if these disguises are accidentally collected as part of the training data, copyright infringement could be triggered unconsciously and unwillingly. As computer scientists, we are obligated to reveal the existence of such possible unlawful acts and provide technical tools to identify such behaviors. We expect this chapter to be a prudent tool such that disguised copyright infringement and its corresponding detection methods are recognized by law experts.

Acknowledgment and fair use Finally we discuss the anticipated impact of our definition of *acknowledgment* on judicial decisions. Specifically, we believe *acknowledgment* will not directly affect the current fair use doctrine or change its decision. To further illustrate the above claim, we first propose two principles: (1) copyright infringement is not fair use: although generative AI law is not rigorously defined, we generally assume that "reproducing the copyrighted content" is not fair use when there's direct access; (2) disguised infringement does not (appear to) use the copyrighted data: there is no decision of fair use doctrine here as there is no apparent "usage" of the copyrighted data, not to mention "fair use". Thus the decision is on the "originality" of the output (reproduction of the copyrighted content) but not fair use. As a result, the notion of *acknowledgment* aims to identify disguised copyright infringement in the second principle and challenge the decision of "originality". Based on the acquired outcome (reproduction), there would be no scenarios that *acknowledgment* can change the decision of fair use according to the first principle.

Learning from noisy data A simultaneous and independent work (Daras et al. 2024) considers learning LDMs from noisy data. Although the techniques are very different, our works share a similar implication: the training dataset may not immediately resemble the generations produced, thus allowing copyright issues to be disguised from an auditor who manually inspects the dataset. In the work of Daras et al. (2024), the training data is the original data with Gaussian noise applied to it. In our case, the training data has the original data hidden in the latent space. In Section 5.5.5, we argue that if training data contains the same latent information as some copyrighted data, then it should be acknowledged. Daras et al. 2024 shows that even when the latent space is corrupted the copyrighted information could still be contained, which could raise a stronger disguised copyright infringement attack.

Chapter 6

Conclusion

This thesis has investigated the critical challenges of untrusted training data in machine learning, focusing on data poisoning attacks from both theoretical and empirical perspectives. Through four interconnected studies, we have examined how maliciously crafted data can influence model behavior across various ML paradigms, while providing novel insights and techniques for both understanding and mitigating these threats.

Our work began by formulating data poisoning as a bi-level optimization problem, specifically a non-zero-sum Stackelberg game. By developing efficient algorithms and architectures for this formulation, we enabled the generation of thousands of poisoned samples simultaneously, a significant advancement over prior methods that operated on individual data points. This approach provided the first comprehensive analysis of indiscriminate data poisoning attacks against modern neural networks.

Our investigations revealed a surprising phenomenon: neural networks demonstrate considerable robustness against indiscriminate data poisoning attacks compared to linear models. To understand this distinction, we introduced a novel theoretical framework centered on the concept of "reachability," which rigorously defines attack success and facilitates exploration of attack budgets across different model architectures. Through this framework, we discovered that reaching a target parameter is only possible when the attack budget exceeds a calculable threshold, providing a complete characterization of poisoning attack difficulty.

Extending beyond standard supervised learning, we examined vulnerabilities in modern ML pipelines that utilize pre-trained feature extractors. Our research demonstrated that transfer learning scenarios with significant domain shifts, such as adapting models to medical imaging, exhibit heightened vulnerability to data poisoning. For critical applications like tumor detection in histopathologic scans, we showed that poisoning can induce accuracy drops of up to 50% without appropriate countermeasures.

Finally, we explored the implications for generative models, particularly text-to-image latent diffusion models. Beyond the well-documented concerns of direct copyright infringement, we uncovered a more insidious threat: disguised copyright infringement enabled by

targeted data poisoning. This attack vector allows adversaries to trigger the reproduction of copyrighted content through seemingly unrelated inputs—a vulnerability currently overlooked by legal frameworks, offering important insights for legal experts and policymakers addressing the emerging challenges of AI-generated content.

Collectively, this thesis advances our understanding of untrusted data impacts from dual perspectives: the ML developer concerned with model integrity and the data owner concerned with privacy and copyright. The methods, theoretical frameworks, and empirical findings presented here provide valuable tools for creating more robust ML systems in an era where data sources cannot always be trusted. As machine learning continues to permeate critical domains, addressing the vulnerabilities introduced by untrusted training data becomes increasingly important. Future work should focus on developing more robust training methodologies, extending theoretical understanding to emergent model architectures, and creating practical detection systems for identifying poisoned data across diverse applications. This thesis lays the groundwork for such efforts, contributing to the development of trustworthy machine learning systems in an increasingly complex data landscape.

References

Achiam, J. et al. (2023). "Gpt-4 technical report". *arXiv preprint arXiv:2303.08774*.

Aghakhani, H., D. Meng, Y.-X. Wang, C. Kruegel, and G. Vigna (2021). "Bullseye polytope: A scalable clean-label poisoning attack with improved transferability". In: *IEEE European Symposium on Security and Privacy (EuroS&P)*, pp. 159–178.

Angel, N. B. et al. (2022). "Benchmarking the Effect of Poisoning Defenses on the Security and Bias of the Final Model". In: *Annual Conference on Neural Information Processing Systems*.

Balcan, M.-F., A. Blum, S. Hanneke, and D. Sharma (2022). "Robustly-reliable learners under poisoning attacks". In: *Proceedings of Thirty Fifth Conference on Learning Theory*, pp. 4498–4534.

Barreno, M., B. Nelson, R. Sears, A. D. Joseph, and J. D. Tygar (2006). "Can machine learning be secure?" In: *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, pp. 16–25.

Bartlett, P. L., M. I. Jordan, and J. D. McAuliffe (2006). "Convexity, classification, and risk bounds". *Journal of the American Statistical Association*, vol. 101, no. 473, pp. 138–156.

Basu, S., P. Pope, and S. Feizi (2021). "Influence functions in deep learning are fragile". In: *International Conference on Learning Representations (ICLR)*.

Bejnordi, B. E. et al. (2017). "Diagnostic assessment of deep learning algorithms for detection of lymph node metastases in women with breast cancer". *Jama*, vol. 318, no. 22, pp. 2199–2210.

Biden, J. R. (2023). "Executive order on the safe, secure, and trustworthy development and use of artificial intelligence".

Biggio, B., B. Nelson, and P. Laskov (2011). "Support vector machines under adversarial label noise". In: *Proceedings of the Asian Conference on Machine Learning (ACML)*, pp. 97–112.

— (2012). "Poisoning attacks against support vector machines". In: *Proceedings of the 29th International Conference on Machine Learning (ICML)*, pp. 1467–1474.

Bshouty, N. H., N. Eiron, and E. Kushilevitz (2002). "PAC learning with nasty noise". *Theoretical Computer Science*, vol. 288, no. 2, pp. 255–275.

Carlini, N., D. Ippolito, M. Jagielski, K. Lee, F. Tramer, and C. Zhang (2022). "Quantifying Memorization Across Neural Language Models". In: *The Eleventh International Conference on Learning Representations*.

Carlini, N., M. Jagielski, C. A. Choquette-Choo, D. Paleka, W. Pearce, H. Anderson, A. Terzis, K. Thomas, and F. Tramèr (2024). "Poisoning web-scale training datasets is practical". In: *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, pp. 407–425.

Carlini, N. and A. Terzis (2021). "Poisoning and Backdooring Contrastive Learning". In: *International Conference on Learning Representations*.

Casper, S. et al. (2024). "Black-box access is insufficient for rigorous AI audits". arXiv preprint arXiv:2401.14446.

Cesa-Bianchi, N., E. Dichterman, P. Fischer, E. Shamir, and H. U. Simon (1999). "Sample-efficient strategies for learning in the presence of noise". *Journal of the ACM (JACM)*, vol. 46, no. 5, pp. 684–719.

Chen, T., S. Kornblith, M. Norouzi, and G. Hinton (2020). "A Simple Framework for Contrastive Learning of Visual Representations". In: *ICML*.

Chen, T., S. Kornblith, K. Swersky, M. Norouzi, and G. E. Hinton (2020). "Big Self-Supervised Models are Strong Semi-Supervised Learners". *NeurIPS*.

Chen, X., S. Xie, and K. He (2021). "An empirical study of training self-supervised vision transformers". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9640–9649.

Chen, X., C. Liu, B. Li, K. Lu, and D. Song (2017). "Targeted backdoor attacks on deep learning systems using data poisoning". arXiv:1712.05526.

Chrabaszcz, P., I. Loshchilov, and F. Hutter (2017). "A downsampled variant of ImageNet as an alternative to the CIFAR datasets". arXiv preprint arXiv:1707.08819.

Cinà, A. E. et al. (2022). "Wild Patterns Reloaded: A Survey of Machine Learning Security against Training Data Poisoning".

Dalvi, N., P. Domingos, S. Sanghai, and D. Verma (2004). "Adversarial classification". In: *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 99–108.

Daras, G., A. G. Dimakis, and C. Daskalakis (2024). "Consistent Diffusion Meets Tweedie: Training Exact Ambient Diffusion Models with Noisy Data". arXiv preprint arXiv:2404.10177.

Deng, J., W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei (2009). "ImageNet: A large-scale hierarchical image database". In: *CVPR*. IEEE.

Deng, L. (2012). "The MNIST database of handwritten digit images for machine learning research [best of the web]". *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142.

Devlin, J., M.-W. Chang, K. Lee, and K. Toutanova (2018). "Bert: Pre-training of deep bidirectional transformers for language understanding". *arXiv preprint arXiv:1810.04805*.

Diakonikolas, I., G. Kamath, D. M. Kane, J. Li, A. Moitra, and A. Stewart (2016). "Robust Estimators in High Dimensions without the Computational Intractability". In: *Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science*, pp. 655–664.

Diakonikolas, I., G. Kamath, D. M. Kane, J. Li, J. Steinhardt, and A. Stewart (2019). "Sever: A Robust Meta-Algorithm for Stochastic Optimization". In: *Proceedings of the 36th International Conference on Machine Learning*, pp. 1596–1606.

Dist. Court, N. C. (2023). "Andersen v. STABILITY AI LTD." Case No. 23-cv-00201-WHO.

Dogan, Ü., T. Glasmachers, and C. Igel (2016). "A Unified View on Multi-class Support Vector Classification". *Journal of Machine Learning Research*, vol. 17, no. 45, pp. 1–32.

Dosovitskiy, A. et al. (2020). "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *International Conference on Learning Representations*.

Evtushenko, Y. G. (1974). "Iterative methods for solving minimax problems". *USSR Computational Mathematics and Mathematical Physics*, vol. 14, no. 5, pp. 52–63.

Farhadkhani, S., R. Guerraoui, O. Villemaud, et al. (2022). "An equivalence between data poisoning and Byzantine gradient attacks". In: *International Conference on Machine Learning*, pp. 6284–6323.

Fiez, T., B. Chasnov, and L. J. Ratliff (2020). "Implicit Learning Dynamics in Stackelberg Games: Equilibria Characterization, Convergence Analysis, and Empirical Study". In: *Proceedings of the International Conference on Machine Learning (ICML)*.

Fowl, L., P.-y. Chiang, M. Goldblum, J. Geiping, A. Bansal, W. Czaja, and T. Goldstein (2021). "Preventing unauthorized use of proprietary data: Poisoning for secure dataset release". arXiv preprint arXiv:2103.02683.

Fowl, L., M. Goldblum, P.-y. Chiang, J. Geiping, W. Czaja, and T. Goldstein (2021). "Adversarial Examples Make Strong Poisons". In: *Advances in Neural Information Processing Systems*, pp. 30339–30351.

Frénay, B. and M. Verleysen (2013). "Classification in the presence of label noise: a survey". *IEEE transactions on neural networks and learning systems*, vol. 25, no. 5, pp. 845–869.

Fu, S., F. He, Y. Liu, L. Shen, and D. Tao (2021). "Robust unlearnable examples: Protecting data privacy against adversarial learning". In: *International Conference on Learning Representations*.

Gal, R., Y. Alaluf, Y. Atzmon, O. Patashnik, A. H. Bermano, G. Chechik, and D. Cohen-Or (2022). "An image is worth one word: Personalizing text-to-image generation using textual inversion". arXiv preprint arXiv:2208.01618.

Gao, L. et al. (2020). "The Pile: An 800GB Dataset of Diverse Text for Language Modeling". arXiv preprint arXiv:2101.00027.

Gatys, L. A., A. S. Ecker, and M. Bethge (2015). "A neural algorithm of artistic style". arXiv preprint arXiv:1508.06576.

Geiping, J., L. H. Fowl, W. R. Huang, W. Czaja, G. Taylor, M. Moeller, and T. Goldstein (2021). "Witches' Brew: ial Scale Data Poisoning via Gradient Matching". In: *International Conference on Learning Representations*.

Goldblum, M., D. Tsipras, C. Xie, X. Chen, A. Schwarzschild, D. Song, A. Madry, B. Li, and T. Goldstein (2023). "Dataset Security for Machine Learning: Data Poisoning, Backdoor Attacks, and Defenses". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 2, pp. 1563–1580.

Gong, C., T. Ren, M. Ye, and Q. Liu (2021). "MaxUp: Lightweight Adversarial Training with Data Augmentation Improves Neural Network Training", pp. 2474–2483.

Gu, T., B. Dolan-Gavitt, and S. Garg (2017). "Badnets: Identifying vulnerabilities in the machine learning model supply chain". arXiv:1708.06733.

Guo, J. and C. Liu (2020). "Practical Poisoning Attacks on Neural Networks". In: *European Conference on Computer Vision*, pp. 142–158.

Hampel, F. R. (1974). "The influence curve and its role in robust estimation". *Journal of the american statistical association*, vol. 69, no. 346, pp. 383–393.

He, H., K. Zha, and D. Katabi (2022). "Indiscriminate Poisoning Attacks on Unsupervised Contrastive Learning". In: *The Eleventh International Conference on Learning Representations*.

He, K., H. Fan, Y. Wu, S. Xie, and R. Girshick (2020). "Momentum contrast for unsupervised visual representation learning". In: *CVPR*.

He, K., X. Zhang, S. Ren, and J. Sun (2016). "Deep Residual Learning for Image Recognition". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 770–778.

Huang, H., X. Ma, S. M. Erfani, J. Bailey, and Y. Wang (2021). "Unlearnable Examples: Making Personal Data Unexploitable". In: *International Conference on Learning Representations*.

Huang, W. R., J. Geiping, L. Fowl, G. Taylor, and T. Goldstein (2020). "Metapoison: Practical general-purpose clean-label data poisoning". In: *Advances in Neural Information Processing Systems*. Vol. 33, pp. 12080–12091.

Huang, X. and S. Belongie (2017). "Arbitrary style transfer in real-time with adaptive instance normalization". In: *Proceedings of the IEEE international conference on computer vision*, pp. 1501–1510.

Huynh, T., S. Kornblith, M. R. Walter, M. Maire, and M. Khademi (2020). "Boosting Contrastive Self-Supervised Learning with False Negative Cancellation". *arXiv preprint arXiv:2011.11765*.

Ilyas, A., S. M. Park, L. Engstrom, G. Leclerc, and A. Madry (2022). "Datamodels: Predicting predictions from training data". In:

Ippolito, D., F. Tramèr, M. Nasr, C. Zhang, M. Jagielski, K. Lee, C. A. Choquette-Choo, and N. Carlini (2023). "Preventing generation of verbatim memorization in language models gives a false sense of privacy". In: *Proceedings of the 16th International Natural Language Generation Conference*. Association for Computational Linguistics, pp. 28–53.

Kearns, M. and M. Li (1988). "Learning in the presence of malicious errors". In: *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pp. 267–280.

Khare, N., P. S. Thakur, P. Khanna, and A. Ojha (2021). "Analysis of Loss Functions for Image Reconstruction Using Convolutional Autoencoder". In: *International Conference on Computer Vision and Image Processing*. Springer, pp. 338–349.

Koh, P. W. and P. Liang (2017). "Understanding black-box predictions via influence functions". In: *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pp. 1885–1894.

Koh, P. W., J. Steinhardt, and P. Liang (2022). "Stronger Data Poisoning Attacks Break Data Sanitization Defenses". *Machine Learning*, vol. 111, pp. 1–47.

Krizhevsky, A. (2009). "Learning multiple layers of features from tiny images". tech. report.

Kumar, R. S. S., M. Nyström, J. Lambert, A. Marshall, M. Goertzel, A. Comissoneru, M. Swann, and S. Xia (2020). "Adversarial machine learning-industry perspectives". In: *IEEE Security and Privacy Workshops (SPW)*, pp. 69–75.

Lai, K. A., A. B. Rao, and S. Vempala (2016). "Agnostic Estimation of Mean and Covariance". In: *Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science*. FOCS '16. Washington, DC, USA: IEEE Computer Society, pp. 665–674.

Levine, A. and S. Feizi (2021). "Deep Partition Aggregation: Provable Defense against General Poisoning Attacks". In: *International Conference on Learning Representations (ICLR)*.

Liu, A. et al. (2024). "Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model". *arXiv preprint arXiv:2405.04434*.

Liu, W. and S. Chawla (2009). "A game theoretical model for adversarial learning". In: *IEEE International Conference on Data Mining Workshops*, pp. 25–30.

— (2010). "Mining adversarial patterns via regularized loss minimization". *Machine learning*, vol. 81, no. 1, pp. 69–83.

Loshchilov, I. and F. Hutter (2017). "SGDR: Stochastic gradient descent with warm restarts". In: *International Conference on Learning Representations*.

Lu, Y., G. Kamath, and Y. Yu (2022). "Indiscriminate Data Poisoning Attacks on Neural Networks". *Transactions on Machine Learning Research*.

— (2023). "Exploring the limits of model-targeted indiscriminate data poisoning attacks". In: *International Conference on Machine Learning*. PMLR.

Lu, Y., M. Y. Yang, G. Kamath, and Y. Yu (2024). "Indiscriminate Data Poisoning Attacks on Pre-trained Feature Extractors". In: *Secure and Trustworthy Machine Learning*. IEEE.

Lu, Y., M. Y. Yang, Z. Liu, G. Kamath, and Y. Yu (2024). "Disguised Copyright Infringement of Latent Diffusion Model". In: *International Conference on Machine Learning*. PMLR.

Lu, Y., Y. Yu, X. Li, and V. Partovi Nia (2024). "Understanding Neural Network Binarization with Forward and Backward Proximal Quantizers". In: *Advances in Neural Information Processing Systems*. Vol. 36.

Lu, Y., G. Zhang, S. Sun, H. Guo, and Y. Yu (2023). "f-MICL: Understanding and Generalizing InfoNCE-based Contrastive Learning". *Transactions on Machine Learning Research*.

Lyu, L., H. Yu, and Q. Yang (2020). "Threats to federated learning: A survey". arXiv preprint arXiv:2003.02133.

Martens, J. (2010). "Deep learning via hessian-free optimization". In: *ICML*. Vol. 27, pp. 735–742.

Metz, L., B. Poole, D. Pfau, and J. Sohl-Dickstein (2017). "Unrolled Generative Adversarial Networks". In: *International Conference on Learning Representation (ICLR)*.

Muñoz-González, L., B. Biggio, A. Demontis, A. Paudice, V. Wongrassamee, E. C. Lupu, and F. Roli (2017). "Towards Poisoning of Deep Learning Algorithms with Back-gradient Optimization". In: *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security (AISec)*, pp. 27–38.

Muñoz-González, L., B. Pfitzner, M. Russo, J. Carnerero-Cano, and E. C. Lupu (2019). "Poisoning attacks with generative adversarial nets". arXiv preprint arXiv:1906.07773.

Natarajan, N., I. S. Dhillon, P. K. Ravikumar, and A. Tewari (2013). "Learning with noisy labels". *Advances in neural information processing systems*, vol. 26.

Nelson, B., M. Barreno, F. J. Chi, A. D. Joseph, B. I. Rubinstein, U. Saini, C. Sutton, J. D. Tygar, and K. Xia (2008). "Exploiting machine learning to subvert your spam filter." *LEET*, vol. 8, pp. 1–9.

Oord, A. van den, Y. Li, and O. Vinyals (2018). "Representation learning with contrastive predictive coding". *arXiv preprint arXiv:1807.03748*.

Paszke, A. et al. (2019). "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*, pp. 8026–8037.

Pawelczyk, M., J. Z. Di, Y. Lu, G. Kamath, A. Sekhari, and S. Neel (2025). "Machine Unlearning Fails to Remove Data Poisoning Attacks". In: *International Conference on Learning Representations*.

Radford, A. et al. (2021). "Learning transferable visual models from natural language supervision". In: *International conference on machine learning*. PMLR, pp. 8748–8763.

Radiya-Dixit, E., S. Hong, N. Carlini, and F. Tramer (2022). "Data Poisoning Won't Save You From Facial Recognition". In: *Proceedings of the 10th International Conference on Learning Representations*. ICLR '22.

Rolnick, D., A. Veit, S. Belongie, and N. Shavit (2017). "Deep learning is robust to massive label noise". *arXiv preprint arXiv:1705.10694*. arXiv preprint arXiv:1705.10694.

Rombach, R., A. Blattmann, D. Lorenz, P. Esser, and B. Ommer (2022). "High-resolution image synthesis with latent diffusion models". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10684–10695.

Ronneberger, O., P. Fischer, and T. Brox (2015). "U-net: Convolutional networks for biomedical image segmentation". In: *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*. Springer, pp. 234–241.

Saha, A., A. Subramanya, and H. Pirsiavash (2020). "Hidden trigger backdoor attacks". In: *Proceedings of the AAAI Conference on Artificial Intelligence*.

Sandoval-Segura, P., V. Singla, J. Geiping, M. Goldblum, T. Goldstein, and D. W. Jacobs (2022). "Autoregressive Perturbations for Data Poisoning". In: *Advances in Neural Information Processing Systems*.

Schuhmann, C., R. Beaumont, et al. (2022). "Laion-5b: An open large-scale dataset for training next generation image-text models". *Advances in Neural Information Processing Systems*, vol. 35, pp. 25278–25294.

Schuhmann, C., R. Kaczmarczyk, A. Komatsuzaki, A. Katta, R. Vencu, R. Beaumont, J. Jitsev, T. Coombes, and C. Mullis (2021). "LAION-400M: Open Dataset of CLIP-Filtered 400 Million Image-Text Pairs". In: *NeurIPS Workshop Datacentric AI*. FZJ-2022-00923. Jülich Supercomputing Center.

Schwarzschild, A., M. Goldblum, A. Gupta, J. P. Dickerson, and T. Goldstein (2021). "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)*.

Shafahi, A., W. R. Huang, M. Najibi, O. Suciu, C. Studer, T. Dumitras, and T. Goldstein (2018). "Poison Frogs! Targeted Clean-Label Poisoning Attacks on Neural Networks". In: *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 6103–6113.

Shan, S., W. Ding, J. Passananti, S. Wu, H. Zheng, and B. Y. Zhao (2024). "Nightshade: Prompt-Specific Poisoning Attacks on Text-to-Image Generative Models". In: *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, pp. 212–212.

Shejwalkar, V., A. Houmansadr, P. Kairouz, and D. Ramage (2022). "Back to the Drawing Board: A Critical Evaluation of Poisoning Attacks on Production Federated Learning". In: *IEEE Symposium on Security and Privacy (SP)*, pp. 1354–1371.

Shumailov, I., Z. Shumaylov, D. Kazhdan, Y. Zhao, N. Papernot, M. A. Erdogdu, and R. Anderson (2021). "Manipulating SGD with Data Ordering Attacks". In: arXiv:2104.09667.

Sohl-Dickstein, J., E. Weiss, N. Maheswaranathan, and S. Ganguli (2015). "Deep unsupervised learning using nonequilibrium thermodynamics". In: *International conference on machine learning*. PMLR, pp. 2256–2265.

Somepalli, G., V. Singla, M. Goldblum, J. Geiping, and T. Goldstein (2023a). "Diffusion art or digital forgery? investigating data replication in diffusion models". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6048–6058.

— (2023b). "Understanding and Mitigating Copying in Diffusion Models". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Stackelberg, H. von (1934). "Market structure and equilibrium". Springer.

"Steinberg v. Columbia Pictures Industries, Inc." (1987).

Steinhardt, J., P. W. Koh, and P. Liang (2017). "Certified defenses for data poisoning attacks". In: *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 3520–3532.

Sun, X., Z. Zhang, X. Ren, R. Luo, and L. Li (2020). "Exploring the vulnerability of deep neural networks: A study of parameter corruption". In: *Proceedings of the AAAI Conference on Artificial Intelligence*.

Suya, F., S. Mahloujifar, A. Suri, D. Evans, and Y. Tian (2021). "Model-targeted poisoning attacks with provable convergence". In: *Proceedings of the 38th International Conference on Machine Learning (ICML)*, pp. 10000–10010.

Szegedy, C., W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus (2014). "Intriguing properties of neural networks". In: International Conference on Learning Representation.

Team, G. et al. (2023). "Gemini: a family of highly capable multimodal models". *arXiv preprint arXiv:2312.11805*.

Tran, B., J. Li, and A. Madry (2018). "Spectral Signatures in Backdoor Attacks". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Truong, T., S. Mohammadi, and M. Lenga (2021). "How transferable are self-supervised features in medical image classification tasks?" In: *Machine Learning for Health*. PMLR, pp. 54–74.

Veeling, B. S., J. Linmans, J. Winkens, T. Cohen, and M. Welling (2018). "Rotation equivariant CNNs for digital pathology". In: *Medical Image Computing and Computer Assisted Intervention–MICCAI 2018: 21st International Conference, Granada, Spain, September 16-20, 2018, Proceedings, Part II 11*. Springer, pp. 210–218.

Vicol, P., J. P. Lorraine, F. Pedregosa, D. Duvenaud, and R. B. Grosse (2022). "On implicit bias in overparameterized bilevel optimization". In: *International Conference on Machine Learning*, pp. 22234–22259.

Vyas, N., S. Kakade, and B. Barak (2023). "On provable copyright protection for generative models". In: *Proceedings of the 40 th International Conference on Machine Learning*.

Wakefield, J. (2016). "Microsoft chatbot is taught to swear on Twitter". *BBC News*.

Wang, W., A. Levine, and S. Feizi (2022). "Lethal Dose Conjecture on Data Poisoning". In: *Advances in Neural Information Processing Systems*.

Wang, W., A. J. Levine, and S. Feizi (2022). "Improved certified defenses against data poisoning with (deterministic) finite aggregation". In: *International Conference on Machine Learning*. PMLR, pp. 22769–22783.

Wang, Y., G. Zhang, and J. Ba (2020). "On Solving Minimax Optimization Locally: A Follow-the-Ridge Approach". In: *The 8th International Conference on Learning Representations (ICLR)*.

Wang, Z., E. P. Simoncelli, and A. C. Bovik (2003). "Multiscale structural similarity for image quality assessment". In: *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*. Vol. 2. Ieee, pp. 1398–1402.

Yu, D., H. Zhang, W. Chen, J. Yin, and T.-Y. Liu (2022). "Availability Attacks Create Shortcuts". In: *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 2367–2376.

Yu, Y., Ö. Aslan, and D. Schuurmans (2012). "A Polynomial-time Form of Robust Regression". In: *Advances in Neural Information Processing Systems 26 (NIPS)*.

Zhang, C., D. Ippolito, K. Lee, M. Jagielski, F. Tramèr, and N. Carlini (2021). "Counterfactual memorization in neural language models". arXiv preprint arXiv:2112.12938.

Zhang, G., K. Wu, P. Poupart, and Y. Yu (2021). "Newton-type Methods for Minimax Optimization". In: *ICML workshop on Beyond First-Order Methods in ML Systems*.

Zhang, Z., R. Luo, X. Ren, Q. Su, L. Li, and X. Sun (2021). "Adversarial parameter defense by multi-step risk minimization". *Neural Networks*, vol. 144, pp. 154–163.

Zhu, C., W. R. Huang, H. Li, G. Taylor, C. Studer, and T. Goldstein (2019). "Transferable clean-label poisoning attacks on deep neural nets". In: *International Conference on Machine Learning*, pp. 7614–7623.

APPENDICES

Appendix A

Appendix of Chapter 2

A.1 Indiscriminate data poisoning attacks

We first show that perfect knowledge attacks and training-only attacks can be executed by solving a non-zero-sum bi-level optimization problem.

A.1.1 Non-zero-sum setting

For perfect knowledge and training-only attacks, recall that we aim at the following bi-level optimization problem:

$$\max_{\mathcal{D}_p} \ \mathcal{L}(\mathcal{D}_v, \mathbf{w}_*), \quad \text{s.t.} \quad \mathbf{w}_* \in \operatorname*{argmin}_{\mathbf{w}\in\mathbb{W}} \ \mathcal{L}(\mathcal{D}_{tr} \cup \mathcal{D}_p, \mathbf{w}), \tag{A.1}$$

where we constrain $|\mathcal{D}_p| = \varepsilon_d |\mathcal{D}_{tr}|$ to limit the amount of poisoned data the attacker can inject. The attacker can solve (A.1) in the training only attack setting. With a stronger assumption where $\mathcal{D}_{test}$ is available, we substitute $\mathcal{D}_v$ with $\mathcal{D}_{test}$ and arrive at the perfect knowledge attack setting.

Existing attacks generate poisoned points one by one by considering the problem:

$$\max_{x_p} \ \mathcal{L}(\mathcal{D}_v, \mathbf{w}_*), \quad \text{s.t.} \quad \mathbf{w}_* \in \operatorname*{argmin}_{\mathbf{w}\in\mathbb{W}} \ \mathcal{L}(\mathcal{D}_{tr} \cup \{x_p, y_p\}, \mathbf{w}).$$

While the inner minimization problem can be solved via gradient descent, the outer maximization problem is non-trivial as the dependency of $\mathcal{L}(\mathcal{D}_v, \mathbf{w}_*)$ on x_p is indirectly encoded through the parameter $\mathbf{w}$ of the poisoned model. As a result, we rewrite the desired derivative using the chain rule:

$$\frac{\partial \mathcal{D}(\mathcal{D}_v, \mathbf{w}_*)}{\partial x_p} = \frac{\partial \mathcal{D}(\mathcal{D}_v, \mathbf{w}_*)}{\partial \mathbf{w}_*} \frac{\partial \mathbf{w}_*}{\partial x_p},$$

where the difficulty lies in computing $\frac{\partial \mathbf{w}_*}{\partial x_p}$, i.e., measuring how much the model parameter $\mathbf{w}$ changes with respect to the poisoning point x_p. Various approaches compute $\frac{\partial \mathbf{w}_*}{\partial x_p}$ by solving this problem exactly, using either influence functions (Koh and Liang 2017) (Influence attack) or KKT conditions (Biggio et al. 2011) (PoisonSVM attack[1]). Another solution is to approximate the problem using gradient descent (Muñoz-González, Biggio, et al. 2017). We discuss each of these approaches below.

Influence attack. The influence function (Hampel 1974) tells us how the model parameters change as we modify a training point by an infinitesimal amount. Borrowing the presentation from Koh and Liang 2017, we compute the desired derivative as:

$$\frac{\partial \mathbf{w}_*}{\partial x_p} = -H_{\mathbf{w}*}^{-1} \frac{\partial^2 \mathcal{L}(\{x_p, y_p\}, \mathbf{w}_*)}{\partial \mathbf{w}_* \partial x_p},$$

where $H_{\mathbf{w}_*}$ is the Hessian of the training loss at $\mathbf{w}_*$:

$$H_{\mathbf{w}_*} := \lambda I + \frac{1}{|\mathcal{D}_{tr} \cup \mathcal{D}_p|} \sum_{(x,y) \in \mathcal{D}_{tr} \cup \mathcal{D}_p} \frac{\partial^2 \mathcal{L}((x,y), \mathbf{w}_*)}{\partial (\mathbf{w}_*)^2}$$

Influence functions are well-defined for convex models like SVMs and are generally accurate for our settings. However, they have been showed to be inaccurate for neural networks (Basu et al. 2021).

PoisonSVM attack. Biggio et al. (2012) replaces the inner problem with its stationary (KKT) conditions. According to the KKT condition, we write the implicit function:

$$\frac{\partial \mathcal{L}(\mathcal{D}_{tr} \cup \{x_p, y_p\}, \mathbf{w}_*)}{\partial \mathbf{w}_*} = 0,$$

which yields the linear system:

$$\frac{\partial^2 \mathcal{L}(\mathcal{D}_{tr} \cup \{x_p, y_p\}, \mathbf{w}_*)}{\partial \mathbf{w}_* \partial x_p} + \frac{\partial^2 \mathcal{L}(\mathcal{D}_{tr} \cup \{x_p, y_p\}, \mathbf{w}_*)}{\partial (\mathbf{w}_*)^2} \frac{\partial \mathbf{w}_*}{\partial x_p} = 0,$$

and thus we can solve the desired derivative as:

$$\frac{\partial \mathbf{w}_*}{\partial x_p} = -\left(\frac{\partial^2 \mathcal{L}(\mathcal{D}_{tr} \cup \{x_p, y_p\}, \mathbf{w}_*)}{\partial (\mathbf{w}_*)^2} \right)^{-1} \frac{\partial^2 \mathcal{L}(\mathcal{D}_{tr} \cup \{x_p, y_p\}, \mathbf{w}_*)}{\partial \mathbf{w}_* \partial x_p}.$$

Note that despite their differences in approaching the derivative, both the influence attack and PoisonSVM attack involve the inverse Hessian.

[1]While this might naturally suggest the name "KKT attack," this name is reserved for a different attack covered in Appendix A.1.3.

Back-gradient attack. Muñoz-González, Biggio, et al. (2017) avoid solving the outer maximization problem exactly by replacing it with a set of iterations performed by an optimization method such as gradient descent. This incomplete optimization of the inner problem allows the algorithm to run faster than the two above methods, and poisoning neural networks.

A.1.2 Zero-sum Setting

Steinhardt et al. 2017 reduce equation A.1 to a zero-sum game by replacing $\mathcal{L}(\mathcal{D}_v, \mathbf{w}_*)$ with $\mathcal{L}(\mathcal{D}_{tr} \cup \mathcal{D}_p, \mathbf{w}_*)$, and the original problem can be written as:

$$\max_{\mathcal{D}_p} \ \mathcal{L}(\mathcal{D}_{tr} \cup \mathcal{D}_p, \mathbf{w}_*), \quad \text{s.t.} \quad \mathbf{w}_* \in \operatorname*{argmin}_{\mathbf{w} \in \mathbb{W}} \ \mathcal{L}(\mathcal{D}_{tr} \cup \mathcal{D}_p, \mathbf{w}).$$

which is identical to the saddle-point or zero-sum problem:

$$\max_{\mathcal{D}_p} \min_{\mathbf{w}} \ \mathcal{L}(\mathcal{D}_{tr} \cup \mathcal{D}_p, \mathbf{w}) \tag{A.2}$$

For an SVM model, given that the loss function is convex, we can solve (A.2) by swapping the min and max and expand the problem to:

$$\min_{\mathbf{w}} \sum_{(x,y) \in \mathcal{D}_{tr}} \mathcal{L}(\{x, y\}, \mathbf{w}) + \max_{\{x_p, y_p\}} \mathcal{L}(\{x_p, y_p\}, \mathbf{w}),$$

However, we emphasize that this relaxed gradient-based attack is problematic and could be ineffective since the loss on the clean data $\mathcal{D}_{tr}$ could still be low. In other words, the inner maximization does not address the true objective where we want to change the model parameter to cause wrong predictions on clean data. This can be addressed by keeping the loss on the poisoned data small, but this contradicts the problem formulation. One solution to this is to use target parameters in Appendix A.1.3.

A.1.3 Zero-sum Setting with Target parameters

Gradient-based attacks solve a difficult optimization problem in which the poisoned data $\mathcal{D}_p$ affects the objective through the model parameter $\mathbf{w}_*$. As a result, evaluating the gradient usually involves computing a Hessian, a computationally expensive operation which can not be done in many realistic settings. Koh, Steinhardt, et al. 2022 propose that if we have a target parameter $\mathbf{w}_*^{tar}$ which maximizes the loss on the test data $\mathcal{L}(\mathcal{D}_{test}, \mathbf{w}_*)$, then the problem simplifies to:

$$\text{find } \mathcal{D}_p, \quad \text{s.t.} \quad \mathbf{w}_*^{tar} = \operatorname*{argmin}_{\mathbf{w} \in \mathbb{W}} \ \mathcal{L}(\mathcal{D}_{tr} \cup \mathcal{D}_p, \mathbf{w}),$$

KKT attack. Since the target parameter $\mathbf{w}_*^{tar}$ is pre-specified, the condition can be rewritten as:

$$\begin{aligned}\mathbf{w}_*^{tar} &= \underset{\mathbf{w}\in\mathbb{W}}{\operatorname{argmin}}\ \mathcal{L}(\mathcal{D}_{tr}\cup\mathcal{D}_p,\mathbf{w}) \\ &= \underset{\mathbf{w}\in\mathbb{W}}{\operatorname{argmin}} \sum_{\{x,y\}\in\mathcal{D}_{tr}} \mathcal{L}(\{x,y\},\mathbf{w}) + \sum_{\{x_p,y_p\}\in\mathcal{D}_p} \mathcal{L}(\{x_p,y_p\},\mathbf{w}),\end{aligned}$$

Again we can use the KKT optimality condition to solve the argmin problem for convex losses:

$$\sum_{\{x,y\}\in\mathcal{D}_{tr}} \nabla\mathcal{L}(\{x,y\},\mathbf{w}_*^{tar}) + \sum_{\{x_p,y_p\}\in\mathcal{D}_p} \nabla\mathcal{L}(\{x_p,y_p\},\mathbf{w}_*^{tar}) = 0$$

Thus we can rewrite the problem as:

$$\text{find } \mathcal{D}_p, \quad \text{s.t.} \quad \sum_{\{x,y\}\in\mathcal{D}_{tr}} \nabla\mathcal{L}(\{x,y\},\mathbf{w}_*^{tar}) + \sum_{\{x_p,y_p\}\in\mathcal{D}_p} \nabla\mathcal{L}(\{x_p,y_p\},\mathbf{w}_*^{tar}) = 0.$$

If this problem has a solution, we can find it by solving the equivalent norm-minimization problem:

$$\min_{\mathcal{D}_p} \left\| \sum_{\{x,y\}\in\mathcal{D}_{tr}} \nabla\mathcal{L}(\{x,y\},\mathbf{w}_*^{tar}) + \sum_{\{x_p,y_p\}\in\mathcal{D}_p} \nabla\mathcal{L}(\{x_p,y_p\},\mathbf{w}_*^{tar}) \right\|_2^2,$$

where the problem can only be minimized if the KKT condition is satisfied. This attack is called the KKT attack.

Of course, the success of this attack relies on the target parameter $\mathbf{w}_*^{tar}$. Koh, Steinhardt, et al. 2022 propose to use the label flip attack for such purpose where we use the trained parameter as the target. This attack achieves comparable results to other attacks while being much faster since it can be solved efficiently using grid search for binary classification. Note that for multi-class classification, this algorithm quickly become infeasible.

Improved min-max. Koh, Steinhardt, et al. 2022 applies the target parameters to address the issue for the relaxed gradient-based attack, where we add the following constraint during training:

$$\mathcal{L}(\{x,y\},\mathbf{w}_*^{tar}) \le \tau,$$

where τ is a fixed threshold. Thus the attacker can search for poisoned points that maximize loss under the current parameter $\mathbf{w}$ while keeping low loss on the target parameter $\mathbf{w}_*^{tar}$.

Model Targeted Poisoning. Suya et al. 2021 propose another algorithm for generating poisoned points using target parameters. This attack considers a different attack strategy from the others, where the attacker adopts an online learning procedure. In this case, the attacker does not have a poison fraction ε_d to generate a specific amount of poisoned data. Instead, the attacker aims at reaching a stopping criteria (can be either a desired accuracy drop or desired distance to the target parameter).

A.1.4 Training-data-only attack

In the training-data-only attack setting, since the attacker does not have access to the training procedure, the bi-level optimization methods are not applicable. The remaining strategies focus either on modifying the labels only (i.e., label flip attacks).

Random label flip attack. Random label flipping is a very simple attack, which constructs a set of poisoned points by randomly selecting training points and flipping their labels:

$$\mathcal{D}_p = \{(\mathbf{x}_i, \bar{y}_i) : (\mathbf{x}_i, y_i) \in \mathcal{D}_{tr}\} \quad \text{s.t.} \quad |\mathcal{D}_p| = \varepsilon_d |\mathcal{D}_{tr}|,$$

where for each class $j = 1, \ldots, c$, we set

$$\bar{y}_i = j \text{ with probability } p_j.$$

Note that the weights $\{p_j\}$ may depend on the true label y_i. For instance, for binary classification (i.e., $c = 2$), we may set $p_{c+1-y_i} = 1$ in which case $\bar{y}_i$ simply flips the true label y_i.

Adversarial label flip attack. Biggio et al. (2011) consider an adversarial variant of the label flip attack, where the choice of the poisoned points is not random. This attack requires access to the model and training procedure, and thus is not a training-data-only attack. Biggio et al. (2011) design an attack focused on SVMs. They choose to poison non-support vectors, as these are likely to become support vectors when an SVM is trained on the dataset including these points with flipped labels.

Label flip for multi-class classification For binary classification, label flip is trivial. Koh, Steinhardt, et al. 2022 provides a solution for multi-class classification problem. For marginal based models (for example, SVM), we can write the multi-class hinge loss, where we have (Dogan et al. 2016):

$$\mathcal{L}(\mathbf{w}, (x_i, y_i)) = \max\{0, 1 + \max_{j \neq y_i} \mathbf{w}_j x_i - \mathbf{w}_{y_i} x_i\},$$

where the choice of j is obvious: we choose the index with the highest function score except the target class y_i. Naturally, we can use this index j as the optimal label flip. As for non-convex models, the choice of optimal label flip is not clear. In this case, one can use a heuristic by choosing the class with the biggest training loss.

A.2 Unlearnable Examples

A.2.1 Stackelberg Game on Unlearnable Examples

We recall the general Stackelberg game in Section 2.4, where the follower F chooses $\mathbf{w}$ to best respond to the action $\mathbf{x}$ of the leader L, through minimizing its loss function f:

$$\forall \mathbf{x} \in \mathbb{X} \subseteq \mathbb{R}^d, \quad \mathbf{w}_*(\mathbf{x}) \in \operatorname*{argmin}_{\mathbf{w} \in \mathbb{W}} f(\mathbf{x}, \mathbf{w}),$$

and the leader L chooses $\mathbf{x}$ to maximize its loss function ℓ:

$$\mathbf{x}_* \in \operatorname*{argmax}_{\mathbf{x} \in \mathbb{X}} \ell(\mathbf{x}, \mathbf{w}_*(\mathbf{x})),$$

where $(\mathbf{x}_*, \mathbf{w}_*(\mathbf{x}_*))$ is a Stackelberg equilibrium.

We then formulate unlearnable examples as a non-zero-sum Stackelberg formulation (Fowl, Chiang, et al. 2021; Fowl, Goldblum, et al. 2021; Fu et al. 2021; H. Huang et al. 2021; W. Liu and Chawla 2010; Sandoval-Segura et al. 2022; D. Yu et al. 2022):

$$\max_{\mathcal{D}_p} \mathcal{L}(\mathcal{D}_v, \mathbf{w}_*), \text{ s.t. } \mathbf{w}_* \in \operatorname*{argmin}_{\mathbf{w}} \mathcal{L}(\mathcal{D}_p, \mathbf{w}).$$

where $\mathcal{D}_p = \{(x_i + \sigma_i, y_i)\}_{i=1}^N$, σ_i is the bounded sample-wise perturbation ($\|\sigma_i\|_p \leq \varepsilon_d^\sigma$), which can be generalized to class-wise perturbation (H. Huang et al. 2021) . Similar to indiscriminate data poisoning attacks, this primal formulation is difficult, as for the outer maximization problem, the dependence of $\mathcal{L}(\mathcal{D}_v, \mathbf{w}_*)$ on $\mathcal{D}_p$ or σ is *indirectly* through the parameter $\mathbf{w}$ of the poisoned model.

However, in practice, we can perform a similar zero-sum reduction in Section 2.4 (W. Liu and Chawla 2010):

$$\max_{\mathcal{D}_p} \min_{\mathbf{w}} \mathcal{L}(\mathcal{D}_p, \mathbf{w}). \tag{A.3}$$

We recall that in indiscriminate data poisoning attacks, such formulation is problematic as the attacker may simply perform well on poisoned points $\mathcal{D}_p$ but poorly on clean points. However, we would not encounter such a problem here as the perturbations are applied across the entire training set $\mathcal{D}_{tr}$.

Now we are ready to categorize existing algorithms on unlearnable examples:

- Error-Minimizing Noise (EMN) (H. Huang et al. 2021): By slightly modifying equation A.3 to

$$\min_{\mathbf{w}} \min_{\mathcal{D}_p} \mathcal{L}(\mathcal{D}_p, \mathbf{w}), \tag{A.4}$$

  Intuitively, H. Huang et al. (2021) construct the perturbation σ to fool the model into learning a strong correlation between σ and the labels.

- Robust Unlearnable Examples (Fu et al. 2021): Fu et al. (2021) further propose a min-min-max formulation following equation A.4:

$$\min_{\mathbf{w}} \min_{\sigma^u} \max_{\sigma^a} \mathcal{L}(\mathcal{D}'_p, \mathbf{w}),$$

where $\|\sigma_i^u\|_p \leq \varepsilon_u$ is the defensive perturbation, which is forced to be imperceptible; $\|\sigma_i^a\|_p \leq \varepsilon_a$ is the adversarial perturbation, which controls the robustness against adversarial training; $\mathcal{D}'_p = \{(x_i + \sigma_i^u + \sigma_i^a, y_i)\}_{i=1}^N$. Fu et al. (2021) find this formulation generates robust unlearnable examples against adversarial training.

- Adversarial poisoning (Error maximizing) (Fowl, Goldblum, et al. 2021): By freezing the follower entirely in equation A.3, Fowl, Goldblum, et al. (2021) propose to solve the maximization problem:

$$\max_{\mathcal{D}_p} \mathcal{L}(\mathcal{D}_p, \mathbf{w}), \tag{A.5}$$

such that it is similar to an adversarial example problem.

- Gradient Matching (Fowl, Chiang, et al. 2021): Fowl, Chiang, et al. (2021) solve the same maximization problem in equation A.5 and apply the gradient matching algorithm in Geiping et al. 2021 (see more details in Section 2.4).

A.2.2 Comparison with Indiscriminate Data Poisoning Attacks

Despite their differences in problem formulation, it is possible to compare algorithms for unlearnable examples (we take EMN as an example here) with our TGDA attack. We identify two possible scenarios where we may fairly compare TGDA and EMN empirically:

- Indiscriminate Data Poisoning Attacks: for EMN, we first craft perturbations using the original algorithm. After the attack, we take $\mathcal{D}_p = \{(x_i + \delta_i), y_i\}_{i=1}^{\varepsilon_d N}$, recall ε_d is the attack budget (or poison rate). Then, we follow our experimental protocol to perform the attack.

- Unlearnable Examples: for TGDA, we follow equation 2.2 and perform the zero-sum reduction of TGDA to perturb the entire training set. Note that we only consider sample-wise perturbation across all experiments. Similar to our test protocol, we retrain the perturbed model and test the performance of the attack on the test set.

We report the experimental results in Table A.1 and Table A.2, where we observe that:

- For indiscriminate data poisoning attacks: In Table A.1, although EMN is efficient, its attack efficacy is poor. Such poor performance is expected as the objective of EMN does not reflect the true influence of an attack on clean test data.

Table A.1: Indiscriminate data poisoning attacks: the attack accuracy/accuracy drop (%) and attack running time (hours) on CIFAR-10.

Model	Clean	**Label Flip**		**EMN**		**TGDA(ours)**	
	Acc	Acc/Drop	Time	Acc/Drop	Time	Acc/Drop	Time
CNN	69.44	68.99/0.45	0 hrs	69.00/0.44	2.2 hrs	65.15/**4.29**	42 hrs
ResNet-18	94.95	94.79/0.16	0 hrs	94.76/0.19	8.4 hrs	89.41/**5.54**	162 hrs

Table A.2: Unlearnable examples: model accuracy (%) under different unlearnable percentages on CIFAR-10 with ResNet-18 model. Percentage of unlearnable examples is defined as $\frac{|\mathcal{D}_p|}{|\mathcal{D}_{tr}+\mathcal{D}_p|}$.

Method	0%	20%	40%	60%	80%	100%
EMN	94.95	94.38	93.10	91.90	86.85	19.93
TGDA	94.95	93.22	92.80	91.85	85.77	16.65

- For unlearnable examples: In Table A.2, we observe that TGDA (after simplification) can be directly comparable with EMN, and the zero-sum simplification allows it to scale up to large models (i.e., ResNet) easily (training time for 100% unlearnable examples is 1.8 hours). However, the perturbation introduced by TGDA is not explicitly bounded.

A.3 Other solvers than TGDA

We recall that in Section 2.4, we solve equation 2.1 and approximate the calculation of $\frac{\partial \mathbf{w}_*}{\partial \mathcal{D}_p}$ using the total gradient descent ascent (TGDA) algorithm (Evtushenko 1974; Fiez et al. 2020):

$$\begin{aligned}\mathbf{x}_{t+1} &= \mathbf{x}_t + \eta_t \mathsf{D}_{\mathbf{x}} \ell(\mathbf{x}_t, \mathbf{w}_t),\\ \mathbf{w}_{t+1} &= \mathbf{w}_t - \eta_t \nabla_{\mathbf{w}} f(\mathbf{x}_t, \mathbf{w}_t)\end{aligned}$$

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 $\mathbf{x}$.

Furthermore, it is possible to apply two other algorithms to solve equation 2.1:

- Follow the ridge (FR) (Evtushenko 1974; Y. Wang et al. 2020):

$$\begin{aligned}\mathbf{x}_{t+1} &= \mathbf{x}_t + \eta_t \mathsf{D}_{\mathbf{x}} \ell(\mathbf{x}_t, \tilde{\mathbf{w}}_t),\\ \mathbf{w}_{t+1} &= \mathbf{w}_t - \eta_t \nabla_{\mathbf{w}} f(\mathbf{x}_t, \mathbf{w}_t) + \eta_t \nabla_{\mathbf{ww}}^{-1} f \cdot \nabla_{\mathbf{xw}} f \cdot \mathsf{D}_{\mathbf{x}} \ell(\mathbf{x}_t, \mathbf{w}_t),\end{aligned}$$

- Gradient descent Newton (GDN) (Evtushenko 1974; G. Zhang et al. 2021):

$$\begin{aligned}\mathbf{x}_{t+1} &= \mathbf{x}_t + \eta_t \mathsf{D}_{\mathbf{x}} \ell(\mathbf{x}_t, \tilde{\mathbf{w}}_t),\\ \mathbf{w}_{t+1} &= \mathbf{w}_t - \eta_t \nabla_{\mathbf{ww}}^{-1} f \cdot \nabla_{\mathbf{w}} f(\mathbf{x}_t, \mathbf{w}_t),\end{aligned}$$

  G. Zhang et al. (2021) showed that both TGDA and FR are first-order approximations of GDN, despite having similar computational complexity of all three.

In our preliminary experiments, TGDA appears to be most effective which is why we chose it as our main algorithm.

Appendix B

Appendix of Chapter 3

B.1 Proofs

Theorem 5 (Linear Models)**.** *Consider* $\ell((\mathbf{x}, y); \mathbf{w}) = l(\mathbf{w}^\top \mathbf{x}, y)$ *for some univariate loss* l*. Then,*

$$\mathbf{g}(\mathbf{x}, y) = \mathbf{x} \cdot l'(\mathbf{w}^\top \mathbf{x}, y),$$

and $\mathbf{w}$ *is* λ*-poisoning reachable iff there exists* $\nu \in \Gamma$ *such that*

$$0 \in (1 - \lambda)\mathbf{g}(\mu) + \lambda \mathbf{g}(\nu).$$

Suppose $\langle \mathbf{w}, \mathbf{g}(\mu) \rangle \neq 0$*. Consider* $\Pi \subseteq \mathcal{P}$ *and let*

$$\mathfrak{g} := \{\mathbb{E}_{(\mathbf{x},y)\sim\nu}(\mathbf{w}^\top \mathbf{x}) \cdot l'(\mathbf{w}^\top \mathbf{x}, y) : \nu \in \Pi\} \subseteq \mathbb{R}.$$

Then, $\mathbf{w}$ *is* λ*-poisoning reachable if* [1] $\Gamma \supseteq T_{\#}\Pi$ *and*

$$0 \in (1 - \lambda) \langle \mathbf{w}, \mathbf{g}(\mu) \rangle + \lambda \mathfrak{g}, \tag{B.1}$$

where the transformation $T(\mathbf{x}, y) := \left(\frac{\langle \mathbf{w}, \mathbf{x} \rangle}{\langle \mathbf{w}, \mathbf{g}(\mu) \rangle} \mathbf{g}(\mu), y \right)$*. Conversely, equation B.1 holds if* $\mathbf{w}$ *is* λ*-poisoning reachable and* $\Pi \supseteq \Gamma$*.*

Proof. The gradient computation is straightforward while the first claim follows from Theorem 1.

Suppose now $\langle \mathbf{w}, \mathbf{g}(\mu) \rangle \neq \mathbf{0}$.

Suppose first equation B.1 holds, so we can choose $\nu \in \Pi$ such that

$$0 = (1 - \lambda) \langle \mathbf{w}, \mathbf{g}(\mu) \rangle + \lambda \mathbb{E}_{(\mathbf{x},y)\sim\nu}(\mathbf{w}^\top \mathbf{x}) \cdot l'(\mathbf{w}^\top \mathbf{x}, y).$$

[1] $T_{\#}\nu$ denotes the distribution of $T(\mathbf{z})$ when $\mathbf{z} \sim \nu$.

Consider the transformation $T(\mathbf{x}, y) = \left(\frac{\langle \mathbf{w}, \mathbf{x} \rangle}{\langle \mathbf{w}, \mathbf{g}(\mu) \rangle} \mathbf{g}(\mu), y\right)$ and let $\tilde{\nu} = T_{\#}\nu$, which is in Γ due to our assumption $\Gamma \supseteq T_{\#}\Pi$. We then have

$$\mathbb{E}_{(\tilde{\mathbf{x}}, \tilde{y}) \sim \tilde{\nu}} l'(\mathbf{w}^\top \tilde{\mathbf{x}}, \tilde{y}) \tilde{\mathbf{x}} = \mathbb{E}_{(\mathbf{x}, y) \sim \nu} l'(\mathbf{w}^\top \mathbf{x}, y) \frac{\langle \mathbf{w}, \mathbf{x} \rangle}{\langle \mathbf{w}, \mathbf{g}(\mu) \rangle} \mathbf{g}(\mu),$$

and hence

$$\begin{aligned}(1-\lambda)\mathbf{g}(\mu) + \lambda \mathbb{E}_{(\tilde{\mathbf{x}}, \tilde{y}) \sim \tilde{\nu}} \nabla \ell((\tilde{\mathbf{x}}, \tilde{y}); \mathbf{w}) &= \left[(1-\lambda)\langle \mathbf{w}, \mathbf{g}(\mu) \rangle + \lambda \mathbb{E}_{(\mathbf{x}, y) \sim \nu} l'(\mathbf{w}^\top \mathbf{x}, y) \langle \mathbf{w}, \mathbf{x} \rangle\right] \frac{\mathbf{g}(\mu)}{\langle \mathbf{w}, \mathbf{g}(\mu) \rangle} \\ &= \mathbf{0},\end{aligned}$$

thanks to our choice of ν. Applying Theorem 1 again we know $\mathbf{w}$ is λ-poisoning reachable.

Conversely, if $\mathbf{w}$ is λ-poisoning reachable, then from Theorem 1 it follows that

$$\mathbf{0} \in (1-\lambda)\mathbf{g}(\mu) + \lambda \mathbb{E}_{(\mathbf{x}, y) \sim \nu} l'(\mathbf{w}^\top \mathbf{x}, y) \mathbf{x}.$$

Taking inner product with the model $\mathbf{w}$ on both sides and noting that $\Gamma \subseteq \Pi$ we verify equation B.1. □

Remark 1. *The condition* $\langle \mathbf{w}, \mathbf{g}(\mu) \rangle \neq 0$ *can be easily checked* a priori. *In case it fails, two possibilities arise:*

- $\mathbf{g}(\mu) = \mathbf{0}$, *in which case poisoning is trivial: simply let* $\nu = \mu$ *for any* λ.
- $\mathbf{g}(\mu) \neq \mathbf{0}$, *in which case we may let* ν *concentrate on the line* $L := \{\alpha \mathbf{g}(\mu) : \alpha \in \mathbb{R}\}$. *Thus, data poisoning succeeds if*

  $$0 = (1-\lambda) + \lambda \mathbb{E}_{(\alpha, y) \sim \nu} l'(0, y) \alpha,$$

  where we identify $\alpha \mathbf{g}(\mu)$ *as* α *for* ν. *As long as* Γ *contains some distribution that puts nonzero mass on* L *and sufficiently large* $l'(0, y)$, $\mathbf{w}$ *is again* λ*-poisoning reachable.*

Once we identify an appropriate subset Π of poisoning distributions, we can even estimate the interval $\mathfrak{g}$ using Monte Carlo algorithms. Moreover, we may restrict the search of a poisoning distribution to the potentially much smaller subset $T_{\#}\Pi$ (where $\mathbf{x}$ lies on the line spanned by $\mathbf{g}(\mu)$).

Remark 2 (Connection to breakdown point)**.** *For simplicity consider* $\mathbb{Z} = \mathbb{R}^d \times \mathbb{R}$. *It is well-known that unbounded convex losses* $(t, y) \mapsto l(t - y)$, *such as the square loss in Example 1, have 0 breakdown point (and hence not robust): even adding a single poisoning point can perturb the model norm* $\|\mathbf{w}\|$ *unboundedly (e.g. Y. Yu et al. 2012, Theorem 5). Theorem 2 gives a much more detailed characterization: In fact, any target model* $\mathbf{w}$ *can be induced by a diminishing amount of poisoning (even if* ν *is supported on a single point)! Indeed, since* l *is unbounded and convex, there exists some* $\tau \in \mathbb{R}$ *such that* $|l'(\tau)| \neq 0$. *It follows then* $a = -\infty$ *and* $b = \infty$, *and hence the threshold in equation 3.3 is trivially 0, for any target model* $\mathbf{w}$. *Of course, our characterization in Theorem 2 continues to hold for any domain* $\mathbb{Z}$, *unbounded or not.*

Example 4 (Dichotomy). *Consider the smooth loss*[2]

$$l(t) = \begin{cases} -(4t+1)\exp(-2), & \text{if } t \le -\frac{1}{2} \\ \exp(\frac{1}{t}), & \text{if } t \in [-\frac{1}{2}, 0] \\ 0, & \text{if } t \ge 0 \end{cases}. \tag{B.2}$$

Clearly, we have $a = 0$ *and* $b = \infty$. *Thus, we arrive at a remarkable dichotomy:*

- *If* $\langle \mathbf{w}, \mathbf{g}(\mu) \rangle = 0$ *(in particular any separating* $\mathbf{w}$*), then data poisoning succeeds with any* $\lambda > 0$*;*
- *If* $\langle \mathbf{w}, \mathbf{g}(\mu) \rangle \neq 0$ *(and hence* $\mathbf{w}$ *cannot separate* μ*), then data poisoning fails with any* $\lambda < 1$.

Note that l in equation B.2 is not calibrated since $l'(0) = 0$ (Bartlett et al. 2006), so it may not be a sensible loss to use in practice. For a calibrated margin loss l, i.e., one that is differentiable at 0 with $l'(0) < 0$, we necessarily have $b > 0$ and $a < 0$, so the threshold in equation 3.3 usually lies strictly in $(0, 1)$, incurring a nontrivial phase transitioning.

Theorem 6 (Multiclass). *Consider* $\ell(\mathbf{x}, \mathbf{y}; W) = l(W^\top \mathbf{x}, \mathbf{y})$ *for some loss* l. *Then*[3],

$$G(\mathbf{x}, \mathbf{y}) := \nabla_W \ell(\mathbf{x}, \mathbf{y}; W) = \mathbf{x} \otimes \nabla l(W^\top \mathbf{x}, \mathbf{y}),$$

and W *is* λ*-poisoning reachable iff there exists* $\nu \in \Gamma$ *such that*

$$\mathbf{0} \in (1-\lambda) G(\mu) + \lambda G(\nu).$$

Suppose $W^\top G(\mu)$ *is non-degenerate and let*

$$\mathfrak{g} := \{\mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\nu}(W^\top \mathbf{x}) \otimes \nabla l(W^\top \mathbf{x}, \mathbf{y}) : \nu \in \Pi\}.$$

Then, W *is* λ*-poisoning reachable if* $\Gamma \supseteq T_\# \Pi$ *and*

$$\mathbf{0} \in (1-\lambda) W^\top G(\mu) + \lambda \mathfrak{g}, \tag{B.3}$$

where the transformation $T(\mathbf{x}, \mathbf{y}) := \big(G(\mu)[W^\top G(\mu)]^{-1} W^\top \mathbf{x}, \mathbf{y}\big)$. *Conversely, equation B.3 holds if* W *is* λ*-poisoning reachable and* $\Pi \supseteq \Gamma$.

Proof. The proof is completely similar to that of Theorem 5. □

[2] This is essentially a smoothed version of the perceptron loss $l(t) = \max\{-t, 0\}$.

[3] We use the notation $\mathbf{a} \otimes \mathbf{b} := \mathbf{a}\mathbf{b}^\top$ for two column vectors.

Proof. [of Example 3] We aim to show that for any $\mathbf{h} \in \mathbb{R}^c$ and one-hot $\mathbf{y} \in \mathbb{R}^c$, we have

$$-\mathfrak{W}(\tfrac{c-1}{e}) \le \langle \mathbf{h}, \mathbf{p} - \mathbf{y} \rangle \le \infty, \text{ where recall that } \mathbf{p} := \texttt{softmax}(\mathbf{h}) = \exp(\mathbf{h}) / \sum_k \exp(h_k).$$

The right-hand side is clear: we need only send some h_k to ∞, as long as $y_k \neq 1$. For the left-hand side, we simplify as follows. W.l.o.g. assume $y_i = 1$. Then,

$$\begin{aligned}
\langle \mathbf{h}, \mathbf{p} - \mathbf{y} \rangle = \sum_k h_k \left[\frac{\exp(h_k)}{\sum_j \exp(h_j)} - y_k \right] &= \frac{\sum_k (h_k - h_i) \exp(h_k - h_i)}{1 + \sum_{j \neq i} \exp(h_j - h_i)} \\
&= \sum_{k \neq i} \frac{\frac{1}{c-1} + \exp(h_k - h_i)}{1 + \sum_{j \neq i} \exp(h_j - h_i)} \cdot \frac{(h_k - h_i) \exp(h_k - h_i)}{\frac{1}{c-1} + \exp(h_k - h_i)} \\
&\ge \inf_t \frac{t \exp(t)}{\frac{1}{c-1} + \exp(t)} \qquad \text{(B.4)} \\
&= -\mathfrak{W}(\tfrac{c-1}{e}),
\end{aligned}$$

where the inequality is achieved when $t \equiv h_k - h_i$ minimizes equation B.4. □

B.2 Data poisoning as measure optimization

In this section we discuss a measure optimization approach for solving the gradient canceling problem:

$$\min_{\nu \in \Gamma} \ \tfrac{1}{2} \|\mathbf{g}(\mu) + \varepsilon_d \mathbf{g}(\nu)\|_2^2, \qquad \text{(B.5)}$$

where we recall that

$$\mathbf{g}(\nu) = \mathbb{E}_{\mathbf{z} \sim \nu} \nabla_{\mathbf{w}} \ell(\mathbf{z}; \mathbf{w})$$

is the model gradient computed over the distribution ν. The objective of equation B.5 is a convex quadratic, although living in an infinite dimensional space (the vector space of all signed measures over $\mathbb{Z}$). A particularly suitable way to solve equation B.5 is the well-known Frank-Wolfe algorithm, where we repeatedly perform "atomic" updates to the measure ν:

$$\nu_{t+1} \leftarrow (1 - \eta_t)\nu_t + \eta_t \zeta_t, \qquad \text{(B.6)}$$

where η_t is the step size, e.g., $\eta_t = \frac{2}{t+2}$. The direction ζ_t is found by solving the linear minimization subproblem:

$$\min_{\zeta \in \Gamma} \ \langle \mathbf{g}(\mu) + \varepsilon_d \mathbf{g}(\nu_t), \mathbf{g}(\zeta) \rangle .$$

When $\Gamma = \mathcal{P}$ consists of all distributions over $\mathbb{Z}$, the above subproblem simplifies to:

$$\min_{\mathbf{z} \in \mathbb{Z}} \langle \mathbf{g}(\mu) + \varepsilon_d \mathbf{g}(\nu_t), \nabla_{\mathbf{w}} \ell(\mathbf{z}; \mathbf{w}) \rangle \,, \tag{B.7}$$

i.e., we find a new poisoning point $\mathbf{z}$ to add to the support of the poisoning distribution ν_t, while the step equation B.6 adjusts the probability mass. One particularly appealing part of this algorithm is that after t iterations, the candidate poisoning distribution ν_t is supported at most on $t+1$ points (assuming we start with some ν_0 supported on a single point). We remark that the subproblem equation B.7 is often nonconvex (in particular for neural networks), and could be challenging to solve. The other difficulty is that an attacker often is not allowed to upload an entire poisoning distribution, so a resampling procedure (on ν) will be necessary to create a poisoning set, which is why we opted for a more direct approach in the main paper.

Another possibility is to parameterize ν as the push-forward of some fixed distribution (e.g., the training distribution), i.e., $\nu = [T(\theta)]_{\#}\mu$, and we optimize the push-forward transformation $T(\theta)$.

B.3 Additional Experiments

B.3.1 Additional implementation details

Hardware and package: experiments were run on a cluster with `T4` and `P100` GPUs. The platform we use is PyTorch (Paszke et al. 2019).

Model in details: for the MNIST dataset, we examine three target models: Logistic Regression; a neural network (NN) with three layers, where we choose hidden size as 784 and apply leaky ReLU with `negative_slope` $= 0.2$ for activation; and a convolutional neural network (CNN) with two convolutional layers with kernel size 3, maxpooling and two fully connected layers with hidden size 128.

Synthetic Datasets: in Figure 3.1 and Figure 3.2, we perform experiemnts on two synthetic datasets. (1) OR dataset: we simply use the OR dataset in 2D space in Figure B.1 and repeat each point for 50 times (200 samples in total) with small Gaussian noise. (2) 10-D Gaussian dataset: we use the `make_classification` function in `sklearn.datasets` with 1000 samples and 10 features.

More on GradPC: to choose proper target parameters (specifically, ε_w), we use validation sets described in Section 3.6 for accuracy drop comparison. The GradPC attack never sees the test set during the construction of its perturbed models.

Batch size: for the Gradient Canceling experiments on MNIST, we set batch size as the size of entire training set (60000) for simplicity. For CIFAR-10 and TinyImageNet experiments, we set batch size as 1000 due to CUDA memory size constraint.

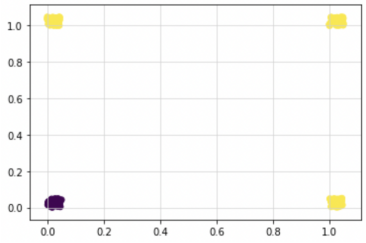


Figure B.1: Here we visualize the OR dataset.

Optimizer, learning rate scheduler and hyperparameters: we use SGD with momentum for optimization and the cosine learning rate scheduler (Loshchilov and Hutter 2017) for the Gradient Canceling algorithm. We set the initial learning rate as 0.5 and run 1000 epochs across every experiment.

B.3.2 More on Parameter Corruption

Recall in Table 3.1 we compare TGDA with GradPC briefly. Here we show the complete results with more choices of ε_w and an additional baseline method called RandomPC in Sun et al. (2020) in Table B.1.

Table B.1: The attack accuracy/accuracy drop (%) on the MNIST dataset.

Target Model	Clean	**TGDA**	**RandomPC**			**GradPC**		
	Accuracy	Accuracy/Drop	$\varepsilon_w = 0.01$	$\varepsilon_w = 0.1$	$\varepsilon_w = 1$	$\varepsilon_w = 0.01$	$\varepsilon_w = 0.1$	$\varepsilon_w = 1$
LR	92.35	89.56 / 2.79 ($\varepsilon_w = 2.45$)	91.94 / 0.41	81.24 / 11.11	24.66 / 67.69	91.91 / 0.44	89.72 / 2.63	**21.48** / **70.87**
NN	98.04	96.54 / 1.50 ($\varepsilon_w = 0.55$)	97.62 / 0.42	82.67 / 15.37	32.77 / 65.27	97.63 / 0.41	97.05 / 0.99	**31.14** / **66.90**
CNN	99.13	98.02 / 1.11 ($\varepsilon_w = 0.74$)	98.84 / 0.29	72.00 / 27.13	19.26 / 79.87	98.74 / 0.39	98.69 / 0.44	**12.98** / **86.15**

B.3.3 Least-square Regression

Recall that from Example 1 we conclude data poisoning with any $\varepsilon_d > 0$ is possible for least-square regression. We perform GC attack on a synthetic 2D Gaussian dataset and visualize the results in Figure B.2. We observe that (1) the algorithm always generates the target parameter regardless of ε_d, which immediately verifies our conclusion; (2) by increasing ε_d, the poison distribution ν gradually moves towards the data distribution μ, which makes intuitive sense. Moreover, recall that we may restrict the search of a poisoning distribution to the potentially much smaller subset $T_{\#}\Pi$ (where $\mathbf{x}$ lies on the line spanned by $\mathbf{g}(\mu)$), while in practice GC does not seem to always follow this theoretical construct.

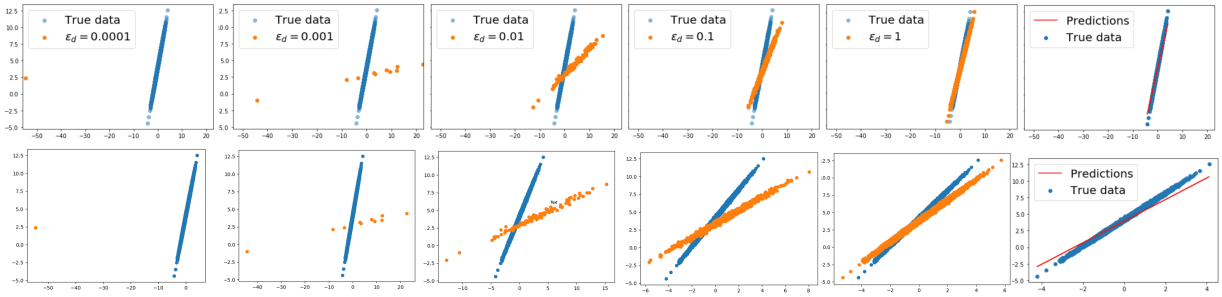


Figure B.2: Here we run the Gradient Canceling algorithm on linear regression on a 2D Gaussian dataset. The first row displays all figures in the same scale for better comparison; and the second row shows the upper figures in their original scale for better viewing. (1) The left five figures show the poisoned points generated with different ε_d. When ε_d is smaller, the poisoned points are farther from the data distribution. (2) The algorithm always generates the target parameter (the prediction) regardless of ε_d.

B.3.4 Comparison with Gradient Matching

As we mentioned in Section 3.5, one of the difference between Gradient Matching and our work is that there is no guarantee that after retraining over $\hat{\nu}$, gradient matching will arrive at the target model while our Algorithm 2 explicitly aims to achieve this goal. We have shown in the Figure B.2 that GC empirically achieve the target model regardless of ε_d. By comparing with Figure B.3, we observe that gradient matching achieves different model parameters for every ε_d.

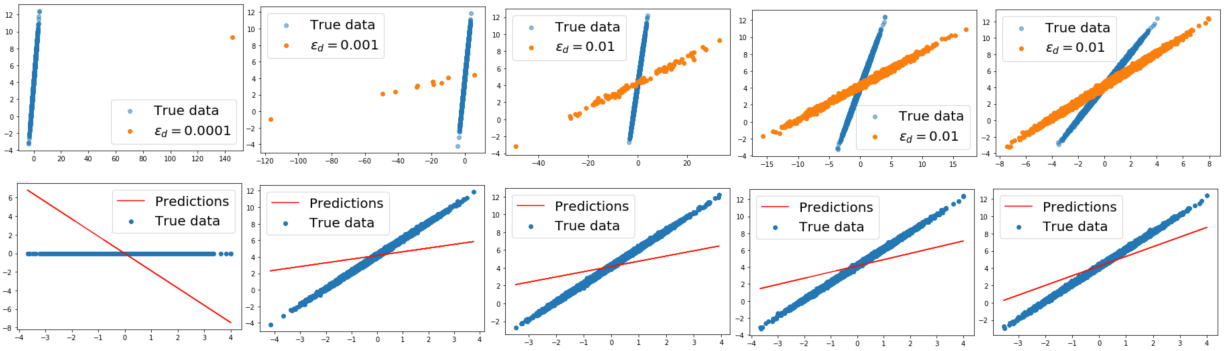


Figure B.3: Here we run the Gradient Matching algorithm on linear regression on a 2D Gaussian dataset. The first row displays the poisoned points generated with different ε_d; and the second row shows the different target parameters generated by the algorithm with different ε_d.

B.3.5 More on Figure 3.3

Recall that in Figure 3.3, we fix ε_d and draw the learning curve for GC optimization for different ε_w, where the y-axis indicates the normalized loss, i.e., $\|\mathbf{g}(\chi)\|$. We observe that when $\tau > \varepsilon_d$, $\|\mathbf{g}(\chi)\|$ converges to a larger value, influenced by the distance between τ and ε_d.

Conversely, we fix ε_w (consequently, τ) for different target models and repeat the MNIST experiments. In Figure B.4, we again observe that when $\tau > \varepsilon_d$, $\|\mathbf{g}(\chi)\|$ converges at a relatively bigger number. Overall, we have confirmed the theoretical limitations proved in Section 3.4.

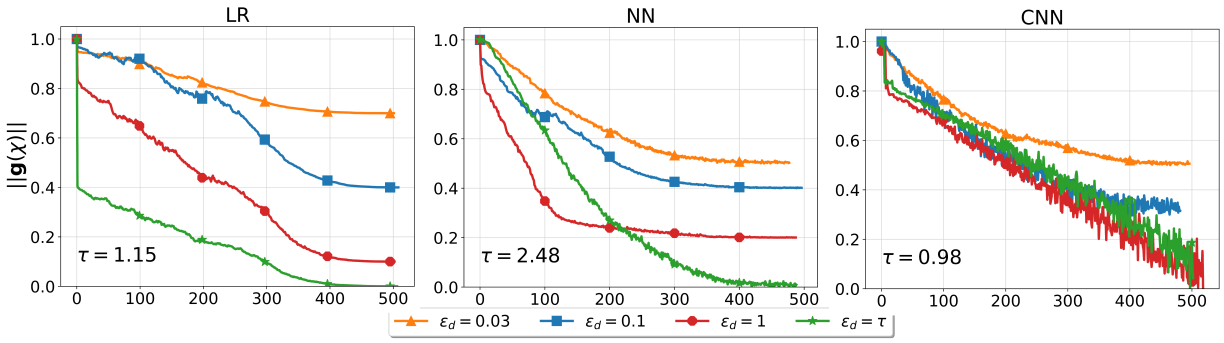


Figure B.4: The learning curve for running GC on MNIST with different target models and ε_d. Note that we fix ε_w for each model and print the respective τ, and the loss indicates $\|\mathbf{g}(\chi)\|$. The figure again confirms that GC cannot achieve $\mathbf{w}$ if $\varepsilon_d < \tau$.

B.3.6 Scaling w

We point out a subtlety in Example 2: by scaling $\mathbf{w}$ towards the origin, we do not change its accuracy (except the confidence it induces). However, the threshold τ tends to 0 and hence data poisoning succeeds in producing the target parameter $\mathbf{w}$ with a smaller λ. In other words, less confident models are easier to poison to, which makes intuitive sense. For verification, we run the GC attack with scaled target parameter $\mathbf{w}/2$ and compare it with the original target parameter $\mathbf{w}$ in Table B.2. With the same target model accuracy, scaling $\mathbf{w}$ significantly reduces its corresponding τ, making it easier to poison to.

The fact that simply scaling $\mathbf{w}$ down could improve its poisoning reachability might seem surprising at first glance. However, this is due to a mismatch between how we train and how we test. It is best to explain this observation in the binary setting, where we note the mismatch between the common prediction rule

$$\hat{y} = \text{sign}(\langle \mathbf{x}, \mathbf{w} \rangle),$$

Table B.2: The GC attack accuracy drop (%) on MNIST when scaling $\mathbf{w}$ by half.

Target Model	clean	GradPC	$\tau(\mathbf{w})$	$\tau(\mathbf{w}/2)$	ε_d	$\mathbf{w}$	$\mathbf{w}/2$
LR	92.35	-70.87	1.15	0.54	0.03	-22.97	-44.11
					0.1	-63.83	-67.22
					1	-67.01	-79.55
NN	98.04	-20.03	2.48	1.41	0.03	-6.10	-9.29
					0.1	-9.77	-11.01
					1	-12.05	-15.33
CNN	99.13	-24.78	0.98	0.42	0.03	-9.55	-12.03
					0.1	-20.10	-21.55
					1	-23.80	-24.56

which is invariant to (positive) scaling (of $\mathbf{w}$), and our training objective in finding a good parameter $\mathbf{w}$, e.g., through logistic regression:

$$\inf_{\mathbf{w}} \frac{1}{n} \sum_i \log[1 + \exp(-y_i \langle \mathbf{x}_i, \mathbf{w} \rangle)], \tag{B.8}$$

which is not invariant to scaling (of $\mathbf{w}$).

Let us give an explicit example to further demonstrate this point. Consider three (cleaning) training points[4] on the real line:

$$\mathbf{x}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, y_1 = +; \quad \mathbf{x}_2 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}, y_2 = +; \quad \mathbf{x}_3 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, y_3 = -; \tag{B.9}$$

where we have padded 1 at the last entry of each $\mathbf{x}$ (so that we can absorb the bias b into $\mathbf{w}$). Setting the derivative of equation B.8 w.r.t. $\mathbf{w}$ to zero we obtain:

$$3 \cdot \mathbf{g}(\mathbf{w}) = -\frac{1}{1 + \exp(w_1 + w_2)} \begin{bmatrix} 1 \\ 1 \end{bmatrix} - \frac{1}{1 + \exp(w_2 - w_1)} \begin{bmatrix} -1 \\ 1 \end{bmatrix} - \frac{1}{1 + \exp(-w_2)} \begin{bmatrix} 0 \\ -1 \end{bmatrix} = \mathbf{0}.$$

Solving the above equation we have $\mathbf{w}_\star = \begin{bmatrix} 0 \\ \ln 2 \end{bmatrix}$.

Now consider the scenario where we are given a target parameter $\mathbf{w} = 2\mathbf{w}_\star$. According to our theory, the poisoning ratio

$$\begin{aligned} \varepsilon_d > \tau :\approx & \max \left\{ 1.2 \left[\frac{-w_1 - w_2}{1 + \exp(w_1 + w_2)} + \frac{w_1 - w_2}{1 + \exp(w_2 - w_1)} + \frac{w_2}{1 + \exp(-w_2)} \right], 0 \right\} \\ = & \max \left\{ 1.2 w_2 \cdot \frac{\exp(w_2) - 2}{1 + \exp(w_2)}, 0 \right\} \end{aligned}$$

[4]This is in fact the smallest example: with 2 or fewer training points, logistic regression does not attain the infimum.

$$= 0.48 \cdot 2 \ln 2 \approx 0.67.$$

In other words, the poisoning set needs to be as large as 67% of the training set, in order to produce $\mathbf{w} = 2\mathbf{w}_\star$. However, if we scale $\mathbf{w}$ down to $\mathbf{w}_\star$, then we do not even need to add any poisoned point, since $\mathbf{w}_\star$ is already stationary (by definition). If we continue to scale $\mathbf{w}$ down to say $0.5\mathbf{w}_\star$ (so that $\langle \mathbf{w}, \mathbf{g}(\mathbf{w}) \rangle < 0$ and hence $\tau = 0$), then for any $\varepsilon > 0$, we may put ε copies of $\mathbf{x} = \alpha \mathbf{g}(\mathbf{w}) \approx \frac{1}{\varepsilon}\mathbf{g}(\mathbf{w})$ as the poisoning set to produce $\mathbf{w}$; see the detailed analysis below.

More generally, consider adding ε copies of a poisoning data point at $\mathbf{x} = \alpha \mathbf{g}(\mathbf{w}), y = +$ (where $\alpha \in \mathbb{R}$ will be determined later) so that the gradient on the clean and poisoned data is proportional to:

$$\begin{aligned} \mathbf{g}(\mathbf{w}) - \varepsilon \frac{1}{1 + \exp(\langle \mathbf{w}, \mathbf{x} \rangle)} \mathbf{x} &= \mathbf{g}(\mathbf{w}) - \varepsilon \frac{1}{1 + \exp(\langle \mathbf{w}, \alpha \mathbf{g}(\mathbf{w}) \rangle)} \alpha \mathbf{g}(\mathbf{w}) \\ &= \mathbf{g}(\mathbf{w}) \cdot \left[1 - \varepsilon \frac{\alpha}{1 + \exp(\alpha \langle \mathbf{w}, \mathbf{g}(\mathbf{w}) \rangle)} \right]. \end{aligned}$$

We can break the analysis into a few cases now:

- $\mathbf{g}(\mathbf{w}) = \mathbf{0}$, i.e., we scale $\mathbf{w}$ down to $\mathbf{w}_\star$, in which case no poisoning point is needed to produce $\mathbf{w} = \mathbf{w}_\star$.
- $\langle \mathbf{w}, \mathbf{g}(\mathbf{w}) \rangle = 0$, in which case the gradient reduces to $\mathbf{g}(\mathbf{w})[1 - \varepsilon\alpha/2]$. Therefore, for any $\varepsilon > 0$, we may produce $\mathbf{w}$ by putting ε copies of $\mathbf{x} = \frac{2}{\varepsilon}\mathbf{g}(\mathbf{w})$.
- $\langle \mathbf{w}, \mathbf{g}(\mathbf{w}) \rangle < 0$, in which case for any $\varepsilon > 0$, the function

  $$\alpha \mapsto 1 + \exp(\alpha \langle \mathbf{w}, \mathbf{g}(\mathbf{w}) \rangle - \varepsilon\alpha$$

  clearly has a zero α_* (easily seen by letting $\alpha \to \pm\infty$ and applying the intermediate value theorem). Thus again, we may produce $\mathbf{w}$ by putting ε copies of $\mathbf{x}$ at $\alpha_* \mathbf{g}(\mathbf{w})$. (Note that $\alpha_* \to \infty$ if $\varepsilon \to 0$; roughly $\alpha_* \approx \frac{1}{\varepsilon}$.)
- $\langle \mathbf{w}, \mathbf{g}(\mathbf{w}) \rangle > 0$, in which case the function

  $$\alpha \mapsto 1 + \exp(\alpha \langle \mathbf{w}, \mathbf{g}(\mathbf{w}) \rangle - \varepsilon\alpha$$

  has a zero α_* iff $\varepsilon \geq \tau$. Indeed,

  $$\begin{aligned} 1 = \varepsilon \frac{\alpha}{1 + \exp(\alpha \langle \mathbf{w}, \mathbf{g}(\mathbf{w}) \rangle)} \iff 1 &= \frac{\varepsilon}{\langle \mathbf{w}, \mathbf{g}(\mathbf{w}) \rangle} \cdot \frac{\alpha \langle \mathbf{w}, \mathbf{g}(\mathbf{w}) \rangle}{1 + \exp(\alpha \langle \mathbf{w}, \mathbf{g}(\mathbf{w}) \rangle)} \\ &\leq \frac{\varepsilon}{\langle \mathbf{w}, \mathbf{g}(\mathbf{w}) \rangle} \cdot \sup_t \frac{t}{1 + \exp(t)} \\ &= \frac{\varepsilon}{\langle \mathbf{w}, \mathbf{g}(\mathbf{w}) \rangle} \cdot \mathcal{W}(1/e). \end{aligned}$$

  Thus again, for any $\varepsilon \geq \tau$, we may produce $\mathbf{w}$ by putting ε copies of $\mathbf{x}$ at $\alpha_* \mathbf{g}(\mathbf{w})$.

In Figure B.5 we observe that GC converged to *nonzero* loss (i.e., unable to produce the target parameter) when $\varepsilon_d < \tau$, for any learning rate we tried, while after scaling $\mathbf{w}$ down so that $\varepsilon_d > \tau$, GC immediately converged to zero loss (without the need of tuning the learning rate), confirming our theoretical analysis above. We plan to further explore the scaling effect in future work.

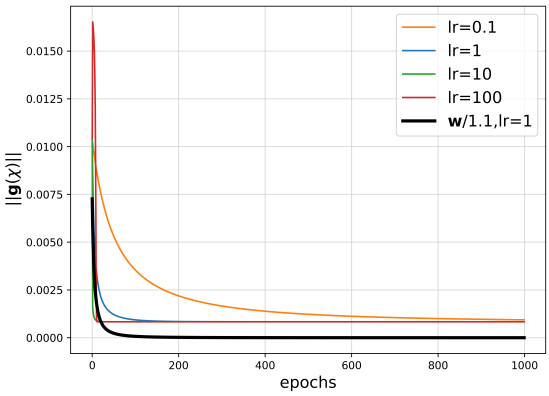


Figure B.5: We plot the training curves of GC on the toy example (see equation B.9). The colored curves represent GC attack on $\mathbf{w} = \begin{bmatrix} 0 \\ 2\ln 2 \end{bmatrix}$ with $\varepsilon = 0.52 < \tau \approx 0.67$ under different learning rates; the black curve represents GC attack on the scaled target parameter $\mathbf{w}/1.1 \approx \begin{bmatrix} 0 \\ 1.82\ln 2 \end{bmatrix}$ with $\varepsilon = 0.52 > \tau \approx 0.51$ under learning rate 1.

B.3.7 Simulating Different Target Parameters

Next, we verify if GC can achieve any desired target parameter. We choose poisoned models generated by TGDA attack as target parameters and perform PC. We discover that such parameters are also achievable by GC in Table B.3, which further confirms that GC may be equipped with any other parameter corruption methods, regardless of how the target parameters are generated.

B.3.8 GC against Defenses

Next, we examine the GC attack against three popular distribution-wise defenses. (1) Influence defense (Koh and Liang 2017) removes ε_d suspicious points according to higher influence functions; (2) Sever (Diakonikolas, Kamath, Kane, J. Li, Steinhardt, et al. 2019) removes ε_d training points with the highest outlier scores, defined using the top singular value in the matrix of gradients; (3) Maxup defense (Gong et al. 2021) generates a set of

Table B.3: Simulating TGDA attack ($\varepsilon_d = 1$) with Gradient Canceling attack on the MNIST dataset.

Target Model	clean	TGDA	τ	ε_d	**GC**
LR	92.35	-8.97	2.33	0.03	-2.66
				0.1	-3.39
				1	-5.53
				τ	-8.35
NN	98.04	-5.49	0.95	0.03	-1.39
				0.1	-1.55
				1	-4.99
CNN	99.13	-4.76	0.49	0.03	-0.98
				0.1	-2.10
				1	-4.68

augmented data with random perturbations and then aims at minimizing the worst case loss over the augmented data.

We present our results on the MNIST dataset in Table B.4 and observe that: (1) Among the three defenses, Sever is the most effective one and can significantly reduce the effectiveness of GC. (2) Clipping the poisoned data to the range of clean training set makes GC more robust against all defenses, with the tradeoff of attack effectiveness. (3) Larger ε_d makes the attack generally more robust, which matches our observation in least-squared regression.

Table B.4: The accuracy drop (%) Gradient Canceling attack (w/wo clipping) introduces on MNIST with Influence/Sever/MaxUp defense (+ indicates the accuracy increased by defenses). GC: original Gradient Canceling attack; GC-c: GC with clipped output; GC-d: GC after defense; GC-cd: GC-c after defense.

Model	Clean Acc	ε_d	GC	GC-c	Influence		Sever		MaxUp	
					GC-d	GC-cd	GC-d	GC-cd	GC-d	GC-cd
LR	92.35	0.03	-22.79	-11.28	-21.99 / +0.80	-11.17 / +0.11	-12.81 / +**9.98**	-9.66 / +**1.62**	-22.59 / +0.20	-11.26 / +0.02
		0.1	-63.83	-26.77	-63.51 / +0.32	-26.67 / +0.10	-59.79 / +**4.04**	-25.53 / +**1.24**	-63.65 / +0.18	-26.67 / +0.10
		1	-67.01	-28.99	-66.75 / +0.26	-26.71 / +0.06	-65.01 / +**2.00**	-27.89 / +**1.10**	-66.02 / +0.09	-28.97 / +0.02
NN	98.04	0.03	-6.10	-3.25	-5.59 / +0.51	-3.16 / +0.09	-3.22 / +**2.88**	-2.26 / +**0.90**	-6.08 / +0.02	-3.24 / +0.01
		0.1	-9.77	-5.10	-9.32 / +0.45	-5.02 / +0.08	-7.66 / +**2.11**	-4.46 / +**0.56**	-9.76 / +0.01	-5.10 / +0.00
		1	-12.05	-6.53	-11.65 / +0.40	-6.48 / +0.05	-10.02 / +**2.03**	-6.11 / +**0.42**	-12.04 / +0.01	-6.53 / +0.00
CNN	99.13	0.03	-9.55	-5.87	-8.57 / +0.98	-5.56 / +0.31	-5.55 / +**4.00**	-4.36 / +**1.51**	-9.39 / +0.16	-5.83 / +0.04
		0.1	-20.10	-12.50	-19.19 / +0.91	-12.35 / +0.15	-16.55 / +**3.55**	-11.32 / +**1.18**	-20.06 / +0.04	-12.48 / +0.02
		1	-23.80	-13.32	-23.10 / +0.70	-13.21 / +0.11	-21.05 / +**2.75**	-12.51 / +**0.81**	-23.79 / +0.01	-13.32 / +0.00

B.3.9 Visualization of Poisoned Images

Finally, we visualize some poisoned images generated by the GC attack in Figure B.6 and Figure B.7.

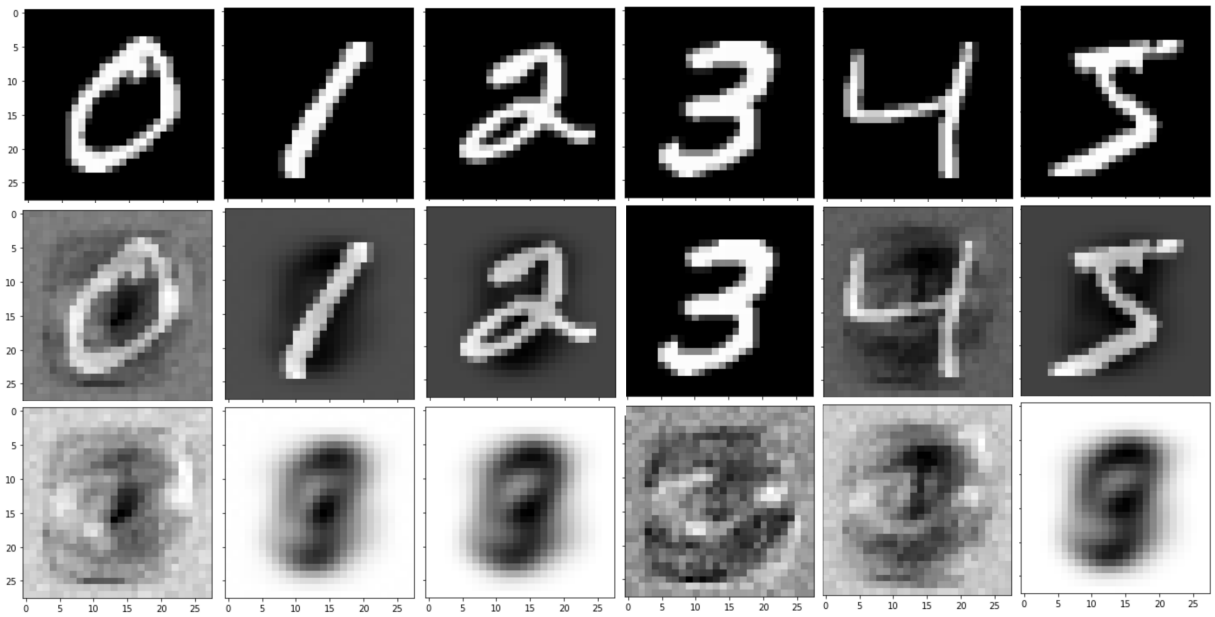


Figure B.6: We visualize some poisoned images generated by the GC attack on the MNIST dataset. The first row shows the clean samples, the second row shows the poisoned samples; the third row displays the perturbation.

B.3.10 Comparison with Replacing Attack

In this work we only consider an adversary who is restricted to *add* corrupted points $\mathcal{D}_p$ to the intact (clean) training set $\mathcal{D}_{tr}$, while an even stronger attacker might consider replacing part of $\mathcal{D}_{tr}$ with $\mathcal{D}_p$ (also closely related to the nasty noise model Bshouty et al. 2002). We first formulate the general case: recall that we consider the mixed distribution $\chi = (1-\lambda)\mu + \lambda\nu$ of the clean distribution μ and poisoned distribution ν, where λ is the proportion of poisoning data. Then, replacing part of the clean training data is equivalent to:

$$\chi = (1-\lambda')\mu' + \lambda'\nu,$$

where μ' is a subset of μ, and $\lambda' = |\mathcal{D}_p|/|\mathcal{D}_{tr}|$. Empirically, with the ability to replace data points we may still apply Gradient Canceling in a straightforward manner: the only difference is that in Algorithm 1 we change μ to μ', a random subset of μ. Following this idea, we perform a simple experiment: we choose $\varepsilon_d = 0.03$, and choose μ' to be a random

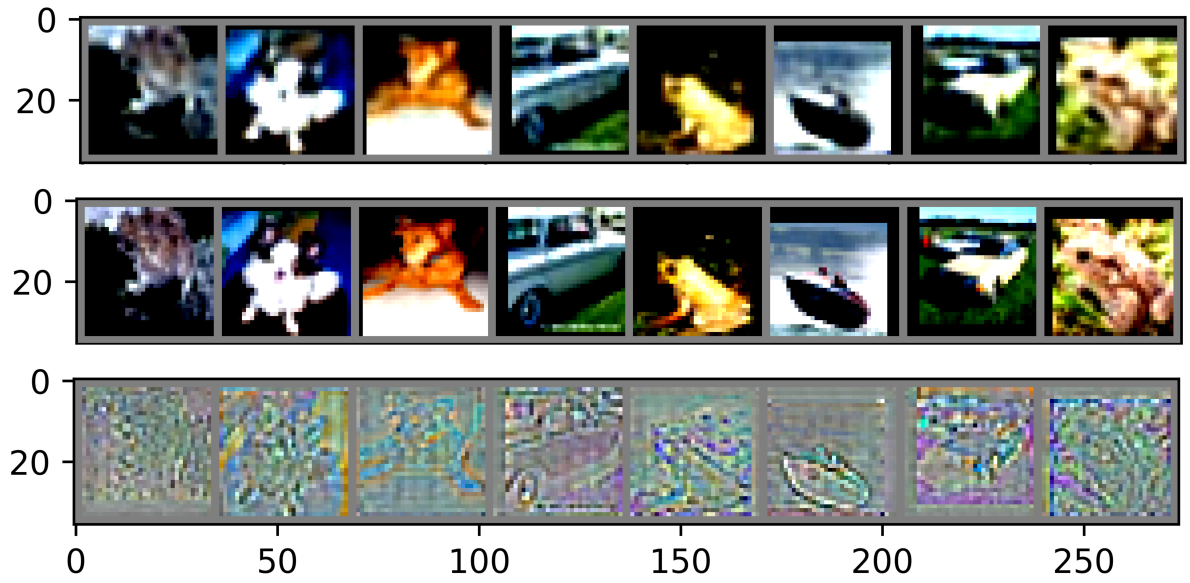


Figure B.7: We visualize some poisoned images generated by the GC attack on the CIFAR-10 dataset. The first row shows the clean samples, the second row shows the poisoned samples; the third row displays the perturbation.

Table B.5: Gradient Canceling attack ($\varepsilon_d = 0.03$) with adding-only vs replacing-only on the MNIST dataset.

Target Model	clean	**GC (adding-only)**	**GC (replacing)**
LR	92.35	-22.97	-23.10
NN	98.04	-6.10	-6.35
CNN	99.13	-9.55	-9.62

subset of $\mathcal{D}_{tr}$, with size $\frac{1}{1+\varepsilon_d}|\mathcal{D}_{tr}| \approx 0.97|\mathcal{D}_{tr}|$. The results on MNIST are presented below in Table B.5:

We observe that the ability to replace clean training data is indeed (slightly) more powerful than the corresponding adding-only attack. Notably, we remove training samples randomly, which may not be relatively weak. Ideally, an adversary would remove the most important points (e.g., in Ilyas et al. 2022) to further reduce the test accuracy. This improved replacing attack might be worth future exploration, although we note that it is less likely to be applicable when an attacker does not have direct access to a victim's infrastructure.

B.4 Selecting Target Parameters

Here we discuss how to select an appropriate target parameter $\mathbf{w}$ for the GC attack. In principle, there are two major factors regarding the selection of target parameters $\mathbf{w}$: (1) strength of $\mathbf{w}$, measured by the test accuracy drop it incurs; (2) poisoning reachability, measured by the optimality condition (i.e., the empirical loss in Equation (3.14)). We want to choose a $\mathbf{w}$ that is both reachable and as strong as possible. Next, we discuss both criteria in details:

- **Strength of $\mathbf{w}$**: (a) existing works (e.g., Koh, Steinhardt, et al. 2022; Suya et al. 2021) only explored rudimentary ways to construct target parameters (e.g., through the label flip attack), and thus are less effective in casting particularly powerful target parameters; (b) with GradPC, we can now easily quantify the strength of a target parameter $\mathbf{w}$ using ε_w (in Table 1); (c) thus in practice, we first prepare a sequence of target parameters $\{\mathbf{w}_k : k \in K\}$ with $|K|$ different ε_w, and then send them all to the reachability test (in the next step).

- **Reachability test**: (a) given the list of $\{\mathbf{w}_k : k \in K\}$, we first calculate every corresponding $\tau(\mathbf{w}_k)$, such that we can already rule out a few choices where $\varepsilon_d < \tau$ (that we know GC cannot achieve with the existing budget ε_d). After this process, we only keep a subset of target parameters $\{\mathbf{w}_k : k \in \bar{K}\}$; (b) next, we run GC for each $\{\mathbf{w}_k : k \in \bar{K}\}$, and examine if GC can achieve them by checking the loss upon convergence. We only keep those $\mathbf{w}_k$'s that return a loss smaller than a margin (this margin is defined by one-tenth of the initial loss) when $\varepsilon_d \approx \tau$. (c) finally, we empirically select a target parameter $\mathbf{w}$ with the largest accuracy drop on the validation set.

Appendix C

Appendix of Chapter 4

Table C.1: Transfer learning results on vision transformers: comparison between input space attacks (GC input space attacks with/without constraints on poisoned samples) and feature matching after removing anomaly points (feature matching-r) w.r.t. different ε_d.

Attack method	Clean	$\varepsilon_d = 0.03$ (1500)	$\varepsilon_d = 0.1$ (5000)	$\varepsilon_d = 0.5$ (25000)	$\varepsilon_d = 1$ (50000)
GradPC (input space)	98.90%	-46.53%	-46.53%	-46.53%	-46.53%
GC-input (no constraints)	98.90%	-40.12%	-42.30%	-44.66%	-46.49%
GC-input (with constraints)	98.90%	- 4.28%	- 5.99%	- 13.12%	-17.66%
GradPC (feature space)	98.90%	-46.53%	-46.53%	-46.53%	-46.53%
GC-feature space	98.90%	-46.51%	-46.51%	-46.51%	-46.51%
Feature matching-r ($\beta = 0.25$)	98.90%	- 6.33%	- 7.89%	- 15.04%	-19.27%

C.1 Perturbations introduced by feature matching

Recall in Figure 4.5, we visualized the poisoned images generated by feature matching attacks with different β. Here in Figure C.1 we also visualize the difference between the clean images and the poisoned images to show the perturbation.

C.2 Additional results on transfer learning

C.2.1 Vision transformer as encoder

Aside from the ResNet architecture, another popular backbone architecture serving as a feature extractor is the recently advanced vision transformers (Dosovitskiy et al. 2020). Here we extend our experiments of poisoning transfer learning on pre-trained vision transformers (specifically, ViT-base) with MoCo V3 (X. Chen, Xie, et al. 2021) on ImageNet to

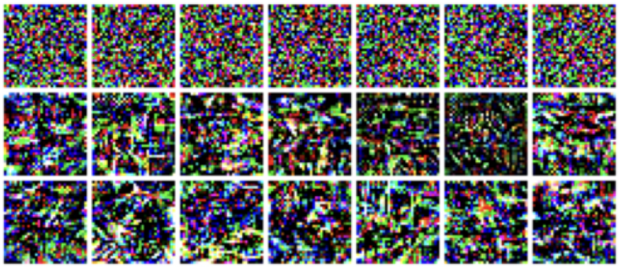


Figure C.1: Here we visualize perturbations introduced by the feature matching algorithm with $\beta = 0.25, 0.1, 0.05$ respectively from the first to the third row.

CIFAR-10. We report our results in Table C.1 and observe that our conclusion still holds for the ViT architecture.

C.2.2 Image encoder trained with CLIP

In this paper, we introduce contrastive learning methods with one modality, i.e., images. Another popular approach to learning feature extractors considers both images and texts, also known as contrastive language-image pre-training (CLIP) (Radford et al. 2021). Instead of choosing different augmentations of the same image as positive pairs and different images as negative pairs, CLIP considers labeled image-text pairs as positive ones, and unmatched pairs as negative ones. Specifically, CLIP trains an image encoder and a text encoder simultaneously, which can be used separately for downstream tasks. Here we examine data poisoning attacks against the image encoder trained by CLIP with a ResNet-50 architecture. We directly adopt the pre-trained weights acquired by OpenAI [1]. We report our result in Table C.2 and observe that feature matching still outperforms GC input space attack (with constraints), even after removing anomaly points.

Table C.2: Transfer learning results on CLIP image encoder: comparison between input space attacks (GC input space attacks with/without constraints on poisoned samples) and feature matching after removing anomaly points (feature matching-r).

Attack method	Clean	$\varepsilon_d = 0.03$ (1500)
GradPC (input space)	92.30%	-44.55%
GC-input (no constraints)	92.30%	-41.20%
GC-input (with constraints)	92.30%	- 4.39%
GradPC (feature space)	92.30%	-44.51%
GC-feature space	92.30%	-42.99%
Feature matching-r ($\beta = 0.25$)	92.30%	- 7.05%

[1] `https://github.com/openai/CLIP`

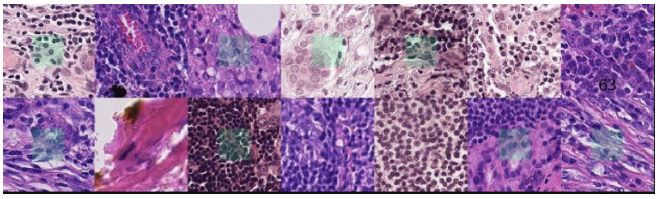


Figure C.2: Example images from the PatchCamelyon dataset.

C.2.3 Transferring to CIFAR-100

In Table 4.2, we show the results of transferring a pre-trained feature extractor (trained on ImageNet) to the CIFAR-10 dataset. As CIFAR-10 only involves 10 classes, the classification problem is rather easy. Here we extend the downstream task to a more difficult setting with 100 classes, i.e. the CIFAR-100 dataset (Krizhevsky 2009). Specifically, CIFAR-100 consists of 100 classes containing 600 images each, with 500 training images and 100 testing images respectively. Again for ImageNet pre-training we apply the ResNet-50 architecture using MoCo V3 (X. Chen, Xie, et al. 2021) and examine the performance of data poisoning attacks against transferring to CIFAR-100. We report the results in Table C.3 and observe that: (1) transferring to CIFAR-100 is more challenging than CIFAR-10 (79.90% clean accuracy compared to CIFAR-10, 93.03%); (2) the attacks are generally weaker under a fine-grained classification setup; (3) feature matching-r still outperform GC input space (with constraints).

Table C.3: Transfer learning results on CIFAR-100: comparison between input space attacks (GC input space attacks with/without constraints on poisoned samples) and feature matching after removing anomaly points (feature matching-r).

Attack method	Clean	$\varepsilon_d = 0.03$ (1500)
GradPC (input space)	79.90%	-38.51%
GC-input (no constraints)	79.90%	-36.78%
GC-input (with constraints)	79.90%	- 3.29%
GradPC (feature space)	79.90%	-38.50%
GC-feature space	79.90%	-37.99%
Feature matching-r ($\beta = 0.25$)	79.90%	- 6.69%

C.2.3.1 Transferring to PatchCamelyon

Next we consider a dataset with a more drastic domain shift from the pre-training dataset, i.e., ImageNet. Here we choose the PatchCamelyon dataset (Bejnordi et al. 2017; Veeling et al. 2018), which consists of 327680 (220025 training samples) color images of size 96×96, extracted from histopathologic scans of lymph node selection. Each image is annotated

with a binary label indicating the presence of metastatic tissue. In Figure C.2, we visualize some samples in the dataset. Again, we choose the feature extractor pre-trained by MoCo V3 (X. Chen, Xie, et al. 2021) (ResNet-50) on ImageNet and examine the performance of data poisoning attacks against transferring to PatchCamelyon. Note that we follow (Truong et al. 2021) and randomly choose 5000 samples from the original dataset for the downstream training. We report our results in Table C.4 and observe that transferring to PatchCamelyon is generally more vulnerable to data poisoning attacks (higher accuracy drop) than CIFAR-10/100, which may indicate that a drastic domain shift is more fragile against such attacks.

Table C.4: Transfer learning results on PatchCamelyon: comparison between input space attacks (GC input space attacks with/without constraints on poisoned samples) and feature matching after removing anomaly points (feature matching-r).

Attack method	Clean	$\varepsilon_d = 0.03$ (150)
GradPC (input space)	79.12%	-50.13%
GC-input (no constraints)	79.12%	-49.05%
GC-input (with constraints)	79.12%	-11.03%
GradPC (feature space)	79.12%	-50.12%
GC-feature space	79.12%	-50.00%
Feature matching-r ($\beta = 0.25$)	79.12%	-15.33%