Towards machine learning models robust to adversarial examples and backdoor attacks

by

Aleksandar Makelov

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2022

© Massachusetts Institute of Technology 2022. All rights reserved.

Author . . . .
Department of Electrical Engineering and Computer Science
August 26, 2022

Certified by . . . .
Aleksander Mądry
Cadence Design Systems Professor of Computing
Thesis Supervisor

Accepted by . . . .
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Towards machine learning models robust to adversarial examples and backdoor attacks

by

Aleksandar Makelov

Submitted to the Department of Electrical Engineering and Computer Science
on August 26, 2022, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

In the past decade, machine learning spectacularly succeeded on many challenging benchmarks. However, are our machine learning models ready to leave this lab setting and be safely deployed in high-stakes real-world applications? In this thesis, we take steps towards making this vision a reality by developing and applying new frameworks for making modern machine learning systems more robust. In particular, we make progress on two major modes of brittleness of such systems: adversarial examples and backdoor data poisoning attacks.

Specifically, in the first part of the thesis, we build a methodology for defending against adversarial examples that is the first one to provide non-trivial adversarial robustness against an adaptive adversary.

In the second part, we develop a framework for backdoor data poisoning attacks, and show how, under natural assumptions, our theoretical results motivate an algorithm to flag and remove potentially poisoned examples that is empirically successful. We conclude with a brief exploration of preliminary evidence that this framework can also be applied to other data modalities, such as tabular data, and other machine learning models, such as ensembles of decision trees.

Thesis Supervisor: Aleksander Mądry
Title: Cadence Design Systems Professor of Computing

Acknowledgments

This thesis would not have been possible without the support of many people, among whom are family, friends and colleagues.

First, I want to thank my advisor, Aleksander, for supporting me through these years – which included a healthy dose of exploration – no matter where my interests took me. He played a major role in shaping my taste and judgment for what makes a good research direction.

I want to also thank all my academic collaborators: Dimitris, Adrian, Ludwig, Alaa, Kristian, Guillaume, Andrew, Hadi, Vivek, Calvin, Alex, Nicholas, Chenyang and Kyriakos – I have learned from all of you and had fun working together. More broadly, I am grateful to the extended Mądry Lab for serving as my academic home while at MIT: Logan, Shibani, Brandon, Kai, Sam, Saachi, Eric, Jerry, Samanta, Natalia, Kamila – and Debbie in particular for making everything easier.

I want to also thank Martin and Srini for serving on my thesis committee and their advice throughout the process. I want to also thank Salil for first sparking my interest in research in theoretical computer science back in college, and believing in me.

There is much to the story of this PhD that cannot be found on these pages. In this context, I want to first and foremost thank Petar for creating the escher programming language and outlining such a brave vision for it. Its design has greatly influenced the direction of my work, and the way I think about computation and the world. Next, I want to thank Nicholas for engaging with, believing in, and collaborating on the software project that ultimately came out of my fascination with programming languages, and for helping popularize it. I want to also thank all the people who have shared their feedback along the long (and ongoing) journey, thus making the trip less lonely, in particular: Nicholas, Petar, Nina, Stefan, David, Evan, Ben, Melody, Kostadin, Sam, Sandeep, Alex, Kristian, Brian and Arpon.

This part of my life would not have been the same without my friends. Here I want to first and foremost thank my brother from another motherland Dylan for playing music together, and in general just being there to hang out. I want to also thank friends from high school and college, in particular Melody, Arpon and Stefan. At MIT, I want to single out the CS Theory music night regulars - Ryan and Virginia for hosting, and Dylan, Nicole, Jerry, Thomas, Rio, Prashant, and many others for attending.

Finally, I am grateful to my family – my brother Martin, and my parents Rozalina and Alexander – for being by my side over these years, and for all the years before. Most of all, I want to thank Nina for her unwavering emotional support and optimism through the difficult parts of these years – I hope you know how much it meant to me.

Contents

Introduction **15**

1 Defenses Against Adversarial Examples **25**

1.1 Introduction 26

1.2 An Optimization View on Adversarial Robustness 28

1.2.1 A Unified View on Attacks and Defenses 30

1.3 Towards Universally Robust Networks 31

1.3.1 The Landscape of Adversarial Examples 32

1.3.2 First-Order Adversaries 35

1.3.3 Descent Directions for Adversarial Training 36

1.4 Network Capacity and Adversarial Robustness 37

1.5 Experiments: Adversarially Robust Deep Learning Models 40

1.6 Related Work 45

1.7 Statement and Application of Danskin's Theorem 47

1.8 Transferability 49

1.9 MNIST Inspection 55

1.10 Conclusion 57

1.11 Supplementary Figures 58

2 Defenses Against Backdoor Attacks **61**

2.1 Introduction 61

2.2 A Feature-Based Perspective on Backdoor Attacks 63

2.3 Inspecting Data Using Datamodels 66

2.4 When Can We Detect Poisoned Samples? 67
2.4.1 Towards a Definition of Model Sensitivity 68
2.4.2 Approximating Sensitivity using Datamodels 71
2.4.3 The Highest-Sensitivity Feature for an Example 72
2.5 Detecting Poisoned Samples Using Datamodels 73
2.5.1 Motivation 73
2.5.2 Algorithm 74
2.6 Experiments and Results 74
2.6.1 Poisoning Attacks 76
2.6.2 Efficiency of the Proposed Defense Mechanism 77
2.6.3 Principal Components of the Datamodels Matrix 78
2.7 Related Work 79
2.7.1 Related Threat Models for Machine Learning 79
2.7.2 Overview of Backdoor Attacks and Defenses 80
2.7.3 Techniques for Influence Estimation 81
2.8 Discussion and Conclusion 82
2.9 Deferred proofs 82
2.9.1 Proof of Lemma 1. 82
2.9.2 Proof of Lemma 2.4.2 86

3 Beyond Image Classification and Neural Networks 89
3.1 Preliminaries 89
3.1.1 The MIMIC-III Dataset and MIMIC-Extract 90
3.1.2 Gradient-boosted Decision Trees 90
3.1.3 Empirical Influence Estimation 91
3.2 Application: Decision Support 92
3.2.1 Motivation 92
3.2.2 Methodology 93
3.2.3 Baselines 93
3.2.4 Results 95

3.3 Application: Detecting Backdoor Attacks 96

List of Figures

1 An adversarial example example . . . 16

1.1 Loss versus number of PGD steps for creating adversarial examples . 33
1.2 Concentration of loss values for local maxima found via PGD 34
1.3 Conceptual illustration of the role of classifier capacity for robustness 38
1.4 Robustness versus network capacity . . . 40
1.5 Loss of the empirical adversarial risk minimization problem during training . . . 41
1.6 Robustness versus perturbation size for ℓ_2 and ℓ_∞ attackers 45
1.7 Restriction of the loss in the direction of white/black-box adversarial examples on CIFAR10 . . . 53
1.8 Transferability experiments for robust and non-robust networks . . . 54
1.9 Inspecting a robust MNIST model . . . 56
1.10 Softmax layer examination of robust and non-robust networks 57
1.11 Loss function value over PGD iterations for robust and non-robust networks . . . 58
1.12 Adversarial examples in the ℓ_2 norm for MNIST . . . 59

2.1 Trigger or naturally occurring feature? . . . 65
2.2 Leveraging features already present in the dataset as backdoors . . . 66
2.3 Total effect of feature versus number of examples of the feature in training set . . . 70
2.4 Checking lemma 2.4.1 empirically . . . 72
2.5 Illustration of backdoor attacks . . . 76

2.6 Efficacy of backdoor removal from training set 78
2.7 Visualizing images along PCA components of datamodels 79
2.8 Visualizing the space of the top two PCA components of datamodels 80

3.1 Decision support comparison between influences and baselines on MIMIC-III 96
3.2 t-SNE visualization of the space of empirical influence estimates for a dataset with a backdoor 97

List of Tables

1.1 MNIST robustness evaluation 43
1.2 CIFAR10 robustness evaluation 44
1.3 CIFAR10 black-box FGSM transfer evaluation 52
1.4 CIFAR10 black-box PGD transfer evaluation 52
1.5 CIFAR10 extensive white-box attack evaluation 53

2.1 Defense against backdoor attacks: evaluation 75
2.2 Defense against backdoor attacks: comparison with baselines 88

3.1 MIMIC-III statistics 91

Introduction

Machine learning (henceforth ML), and deep learning in particular, have in recent years made great strides on challenging artificial intelligence benchmarks, with impressive results ranging widely from computer vision [KSH12] to playing games [SHS+18] to natural language processing [BMR+20] to robotics [ABC+20] to self-driving cars. These successes give us hope for a future in which pervasive ML systems automate tedious menial and mental tasks, and even augment and improve our health, intelligence and society.

However, are these impressive research results and demonstrations ready to be turned into equally impressive and impactful applications in a messy, heterogeneous and at times adversarial world? While our state-of-the-art ML models universally achieve great results across domains and modalities, they turn out to be equally universally *brittle* to shifts – both benign and adversarial – in their training data. This thesis is about developing principled ways to defend against particular kinds of such brittleness. In the following sections, we give a roadmap to the thesis and our main contributions.

Part I: Defending Against Adversarial Examples

Adversarial examples [SZS+14] are a striking phenomenon whereby deliberate, humanly-imperceptible perturbations to an example can cause a state-of-the-art ML model to misclassify the example (for an illustration, see Figure 1).

The existence of adversarial examples may seem quite exotic, but it is only the beginning of their story. Adversarial examples possess many other intriguing

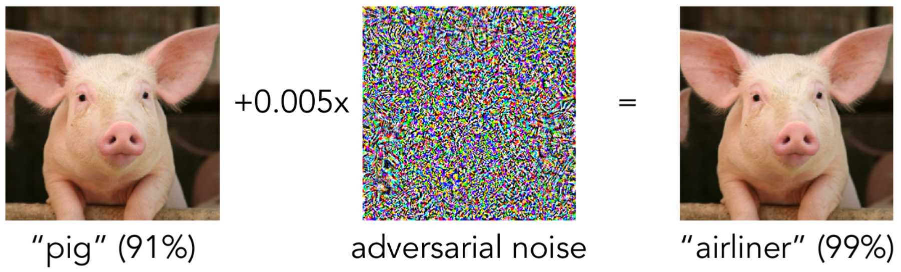


Figure 1: An adversarial example example

properties, most interestingly:

- **Abundant**: adversarial examples can be found in abundance in the vicinity of almost all examples in the dataset [SZS+14].

- **Transferable**: adversarial examples crafted to fool one ML model often fool other models trained for the same task – even when trained on different data [SZS+14].

- **Robust to real-world conditions**: it is possible to manufacture real-world objects that function as adversarial examples from most camera angles [AEIK18]

- **Easy to craft**: adversarial examples can be generated using simple instantiations of standard first-order optimization methods [GSS15].

Taken together, these empirical observations from the literature suggest that adversarial examples present a potent and immediate practical threat to the real-world deployment of ML models in potentially adversarial conditions. On a more conceptual level, the phenomenon of adversarial examples suggests that ML models and humans may use vastly different ways to represent the world. This raises concerns about the *alignment* between humans and ML agents of the future: if our machines *see* their environment so differently from us, how can we be sure that there won't be cases when they also *behave* differently from what we would expect?

One wonders if these problems are inevitable. Below, we give a brief overview of the state of the field prior to our work, and then present a high-level outline of our principled method to defend neural networks against adversarial examples.

Prior Work

Prior to our work, approaches for crafting and defending against adversarial examples followed a particular pattern: given an existing attack, a new defense would be devised that is robust to this attack, yet some time later a *new* attack would be crafted that breaks this particular defense.

For example, distillation [PMW+16] was proposed as a defense against adversarial examples that effectively prevents simple first-order attacks [GSS15] by smoothing the loss landscape. However, it was later shown that this defense, while effective against these particular attacks, was still vulnerable to imperceptible perturbations crafted in other ways [CW17b]. This kind of back-and-forth occurred multiple times throughout the literature (see e.g., [XEQ18], [AG17] and then [HWC+17]; or [GMP+17] and then [CW17a]).

A common theme emerges: defenses eventually fail because they are tailored – either explicitly or implicitly – towards preventing *specific* attacks. To end this cycle, a more rigorous approach is needed: we need a formal notion of what constitutes an imperceptible perturbation, and a method that prevents the existence of *any* such perturbation that fools the model – not just the kinds of attacks we can think of today! This is the starting point of our framework.

Our Framework: Adversarial Risk Minimization

As a first step towards the vision of a robustness guarantee against any attack, we postulate a *threat model* that formally captures all possible perturbations to an example x. Namely, we use some 'small' set Δ so that the allowable attacks for example x are the set $\{x + \delta \big| \delta \in \Delta\}$. In practice, we often use small norm balls for the $\ell_\infty, \ell_2, \ldots$ norms.

Traditionally in ML classification, our goal is to find a classifier with parameters θ that minimizes the *risk* of the classifier on the underlying data distribution:

$$\min_{\theta} \left[\mathbb{E}_{(x,y)\sim\mathcal{D}} \left[\text{loss}\left(x, y; \theta\right) \right] \right]$$

This paradigm has been conceptually and empirically successful. Importantly, if we manage to find θ such that this risk is small, this *guarantees* (under reasonable assumptions on the loss function) that we have solved the underlying classification problem well. In practice, we do not have complete knowledge of the 'true distribution' $\mathcal{D}$; instead, we use *empirical risk minimization* (ERM) over a finite dataset:

$$\min_{\theta}\left[\frac{1}{n}\sum_{i=1}^{n}\operatorname{loss}(x_i, y_i; \theta)\right].$$

Empirically, we observe that with neural networks solving the ERM problem often suffices to find a solution with low risk. How may we reproduce this success in the context of our problem – *adversarially robust* classification?

A natural way is to augment the risk minimization framework with an adversary. Namely, we consider the *adversarial risk minimization problem*:

$$\min_{\theta}\left[\mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\max_{\delta\in\Delta}\left[\operatorname{loss}\left(x+\delta, y; \theta\right)\right]\right]\right]$$

Crucially, if we find θ so that this value is small, we have trained an adversarially robust machine learning classifier. Mirroring the success of the ERM framework, one would hope that solving the empirical version of this robust learning problem would similarly result in robust classifiers that generalize *robustly*.

In Part I of this thesis, we motivate the steps outlined above, and provide empirical and theoretical evidence that this optimization problem, while in principle intractable, can indeed be well approximated in practical situations. We turn these observations into a practical defense algorithm that has resulted in the first non-trivial robustness against adversarial examples for neural networks that has stood the test of time.

A Conceptual Outlook

As a parting note on the topic of adversarial examples, we briefly revisit the more philosophical question alluded to above:

Why do adversarial examples exist in the first place?

As it turns out, subsequent research [IST+19] proposes the hypothesis that the existence of adversarial examples is not a fluke, but rather a reflection of a more fundamental phenomenon: neural networks learn to use *non-robust* features – for example, the texture of an image – that can be changed by humanly imperceptible perturbations, and that humans don't seem to leverage as much in their thought process.

This intriguing idea raises many questions about the kinds of features we want our ML models to learn, and how to prevent them from learning potentially unsafe features. This brings us to our next topic: backdoor attacks.

Part II: Defending Against Backdoor Attacks

Backdoor data poisoning attacks [GDGG17] are another kind of attack on ML models. To mount a backdoor attack, an adversary modifies a small subset of the training inputs in a systematic way by adding a 'trigger' pattern, and (optionally) changing the labels of the affected examples. This intervention allows the adversary to manipulate the resulting model's predictions at test time by inserting the trigger into test inputs.

Mirroring our discussion of adversarial examples, backdoor attacks have some intriguing properties of their own:

- **small sample size**: an attacker can plant a backdoor by manipulating as much as 1% of the dataset. With current models' ever-growing appetite for data, model trainers often collect data from many different sources. It is enough for just one of these source to be corrupted for the entire model to be compromised.
- **difficult to detect**: to the model trainer, everything will typically look right with the model. Indeed, under a naive evaluation, the test performance on clean data (i.e., data where the trigger is absent) will be as good as for a model

trained on clean data. Furthermore, data poisoning attacks exist that are difficult to detect even with human inspection of the data.

Overall, backdoor attacks are a plausible practical threat to ML models. How can we defend our models against this threat model? A natural approach for defenses against backdoor attacks is to flag some fraction (e.g., 10%) of the dataset as potentially containing a trigger, and training on the remainder of the dataset with the hopes of arriving at a model where the backdoor is absent, or at least greatly diminished in efficacy.

This is also the goal we aim for in our work. We first give an overview of prior work in the area, which to some extent mirrors the development in the field of adversarial examples. Then, we outline our framework for detecting backdoors in training data. It centers on the idea of seeing the trigger as *just another feature in the data* – albeit one to which the model is most sensitive to among other features naturally occurring in the data.

Prior Work

Backdoor attacks and defenses are an active research area in machine learning. The development of the field mirrors to some extent the back-and-forth between attacks and defenses we discussed above in the context of adversarial examples.

For example, the first backdoor attacks used simple trigger patterns and mislabeled the poisoned images [GDGG17]. However, due to the blatantly mislabeled examples, it turned out that it is easy to detect this attack by manually inspecting a small fraction of the dataset for triggers, and training a classifier based on these labels to determine the poisoned examples in the rest of the dataset [TTM19].

This in turn motivated the stealthier *clean-label* backdoor attack [TTM19, SSP20] which can be effective without mislabeling images. However, this attack can be defended against using outlier detection in the *latent space* of trained neural networks [TLM18, CCB+18, HKSO21a].

However, if the attacker knows what kind of defense will be deployed, they can craft *adaptive attacks* [S^+20, QXMM22] to bypass the above defenses by making the latent representations of the poisoned examples hard to separate from those of the clean examples.

How can we end this cycle and develop defenses against backdoor attacks that have conceptually sound guarantees?

Backdoor Triggers as Features

A perspective that emerges from the literature on backdoor attacks and defenses is that the only *necessary* quality of backdoor triggers is not a specific visual or latent marker, but rather the *effect* they have on model predictions. Moreover, as we demonstrate in chapter 2, backdoor triggers can take forms *conceptually indistinguishable* from features naturally occurring in the data.

This makes the detection of backdoor triggers a seemingly impossible problem, for there would be nothing distinguishing them from all the other 'ordinary' features in the data. To overcome this obstacle, we make the natural assumption that backdoor triggers are features to which *the model is particularly sensitive*. After all, the goal of the attacker is to introduce the smallest number of corrupted examples with the largest effect on classification of other samples with the trigger.

Adopting this perspective, we develop a formal notion of *learning algorithm sensitivity* to a feature in a training dataset D. Intuitively, the sensitivity of a learning algorithm $\mathcal{A}$ to a feature f on example x is the expected change in the trained model's behavior on x when a number of examples of f are added to the algorithm's training set. Here the expectation is taken over random choices of subsets of D on which $\mathcal{A}$ is trained (conditioned on having a certain number of examples of f).

While at first glance this expectation is intractable, by leveraging a recently discovered empirical phenomenon, it turns out that we can feasibly obtain approximations to the sensitivity quantity. Specifically, we use the datamodeling framework [IPE^+22], and we theoretically derive estimates of sensitivity to a feature within this framework.

We then develop a theoretically motivated algorithm to detect backdoor examples and deploy it against a battery of attack scenarios. We show empirically that our algorithm is on par with, and in many cases outperforms, state of the art defenses.

Part III: Beyond Image Data and Neural Networks

Finally, we present some preliminary results on how the datamodeling framework [IPE+22] can be applied to other data modalities (beyond images) and machine learning models (beyond neural networks) by doing a study on a simple benchmark task (in-ICU mortality) for the MIMIC-III tabular medical dataset [JPS+16] using tree ensembles as the learning algorithm.

Specifically, we investigate the following questions. First, how brittle are predictions on this dataset to the removal of examples from the training set? By leveraging the approach from [IPE+22], we show that, despite the relatively fast saturation of performance on this dataset with increasing training set size, for a given test example x there often exist small (as few as 100 examples) *worst-case* subsets of the training set whose removal causes misclassification of x. Second, can we use the same methods to detect data poisoning attacks? For this quesiton, we show that the space of datamodel vectors contains information that can be used to detect poisoned examples in simple poisoning scenarios.

Outlook: Towards Robust Machine Learning

While in recent years we have made much progress towards the vision of reliable ML systems we can safely deploy and benefit from in the real world, many challenges remain. Beyond threats like adversarial examples and data poisoning, fundamental problems include spurious correlations, brittleness to distribution shift, and difficulty in interpreting model predictions, to name only a few. What are concrete takeaways from the work presented in this thesis that can help us identify promising future directions for progress on these problems?

In the two main case studies presented in this thesis – adversarial examples and data poisoning – having a principled conceptual framework of the phenomena under study has proven to be a crucial ingredient in untangling the complexities of the many different guises in which these phenomena manifest empirically. What may be other similarly powerful conceptual frameworks for making sense of these and other key phenomena in ML? How might we design experiments to identify and separate the fundamental aspects of these problems from the accidental? These are questions to keep in mind as we continue our exploration.

Thesis Organization

Chapter 1 describes our methodology for defending machine learning models against adversarial examples. It is based on joint work with Aleksander Mądry, Dimitris Tsipras, Ludwig Schmidt and Adrian Vladu.

Chapter 2 describes our approach for quantifying model sensitivity to features in the data, and using this framework to detect backdoor data poisoning attacks. It is based on joint work with Alaa Khaddaj, Guillaume Leclerc, Kristian Georgiev, Andrew Ilyas, Hadi Salman and Aleksander Mądry.

Chapter 3 includes material on applications of the datamodeling framework [IPE+22] to other data modalities and learning algorithms. This is based on joint work with Aleksander Mądry.

Chapter 1

Defenses Against Adversarial Examples

Recent work has demonstrated that deep neural networks are vulnerable to adversarial examples—inputs that are almost indistinguishable from natural data and yet classified incorrectly by the network. In fact, some findings have suggested that the existence of adversarial attacks may be an inherent weakness of deep learning models. To address this problem, we study the adversarial robustness of neural networks through the lens of robust optimization. This approach provides us with a broad and unifying view on much of the prior work on this topic. Its principled nature also enables us to identify methods for both training and attacking neural networks that are reliable and, in a certain sense, universal. In particular, they specify a concrete security guarantee that would protect against *any* adversary. These methods let us train networks with significantly improved resistance to a wide range of adversarial attacks. They also suggest the notion of security against a *first-order adversary* as a natural and broad security guarantee. We believe that robustness against such well-defined classes of adversaries is an important stepping stone towards fully resistant deep learning models.[1]

[1]Code and pre-trained models are available at https://github.com/MadryLab/mnist_challenge and https://github.com/MadryLab/cifar10_challenge.

1.1 Introduction

Breakthroughs in computer vision [KSH12, HZRS15b] and natural language processing [CW08] are bringing trained classifiers into the center of security-critical systems. Important examples include vision for autonomous cars, face recognition, and malware detection. These developments make security aspects of machine learning increasingly important. In particular, resistance to *adversarially chosen inputs* is becoming a crucial design goal. While trained models tend to be very effective in classifying inputs with benign noise, another line of work [BCM+13, SZS+14, NYC15] shows that an adversary is often able to manipulate the input so that the model produces an incorrect output.

This phenomenon has received particular attention in the context of deep neural networks, and there is now a quickly growing body of work on this topic [GSS15, FFF18, SGSR17, KGB17, PMG16, TPGDB17]. Computer vision presents a particularly striking challenge: very small changes to the input image can fool state-of-the-art neural networks with high confidence [SZS+14, MFF16]. This holds even when the benign example was classified correctly, and the change is imperceptible to a human. Apart from the security implications, this phenomenon also demonstrates that our current models are not learning the underlying concepts in a robust manner. All these findings raise a fundamental question:

How can we train deep neural networks that are robust to adversarial inputs?

There is now a sizable body of work proposing various attack and defense mechanisms for the adversarial setting. Examples include defensive distillation [PMW+16, CW17b], feature squeezing [XEQ18, HWC+17], and several other adversarial example detection approaches [CW17a]. These works constitute important first steps in exploring the realm of possibilities in this field. They, however, do not offer a good understanding of the *guarantees* they provide. We can never be certain that a given attack finds the "most adversarial" example in the context, or that a particular defense mechanism prevents the existence of some well-defined *class* of adversarial attacks. This makes it difficult to navigate the landscape of adversarial robustness or to fully evaluate the possible security implications.

In this chapter, we study the adversarial robustness of neural networks through the lens of robust optimization. We use a natural saddle point (min-max) formulation to capture the notion of security against adversarial attacks in a principled manner. This formulation allows us to be precise about the type of security *guarantee* we would like to achieve, i.e., the broad *class* of attacks we want to be resistant to (in contrast to defending only against specific known attacks). The formulation also enables us to cast both *attacks* and *defenses* into a common theoretical framework, naturally encapsulating most prior work on adversarial examples. In particular, adversarial training directly corresponds to optimizing this saddle point problem. Similarly, prior methods for attacking neural networks correspond to specific algorithms for solving the underlying constrained optimization problem.

Equipped with this perspective, we make the following contributions.

1. We conduct a careful experimental study of the optimization landscape corresponding to this saddle point formulation. Despite the non-convexity and non-concavity of its constituent parts, we find that the underlying optimization problem *is* tractable after all. In particular, we provide strong evidence that first-order methods can reliably solve this problem. We supplement these insights with ideas from real analysis to further motivate projected gradient descent (PGD) as a universal "first-order adversary", i.e., the strongest attack utilizing the local first order information about the network.

2. We explore the impact of network architecture on adversarial robustness and find that model capacity plays an important role here. To reliably withstand strong adversarial attacks, networks require a larger capacity than for correctly classifying benign examples only. This shows that a robust decision boundary of the saddle point problem can be significantly more complicated than a decision boundary that simply separates the benign data points.

3. Building on the above insights, we train networks on MNIST [LeC98] and

CIFAR10 [Kri09] that are robust to a wide range of adversarial attacks. Our approach is based on optimizing the aforementioned saddle point formulation and uses PGD as a reliable first-order adversary. Our best MNIST model achieves an accuracy of more than 89% against the strongest adversaries in our test suite. In particular, our MNIST network is even robust against *white box* attacks of an *iterative* adversary. Our CIFAR10 model achieves an accuracy of 46% against the same adversary. Furthermore, in case of the weaker *black box/transfer* attacks, our MNIST and CIFAR10 networks achieve the accuracy of more than 95% and 64%, respectively. (More detailed overview can be found in Tables 1.1 and1.2.) To the best of our knowledge, we have been the first to achieve these levels of robustness on MNIST and CIFAR10 against such a broad set of attacks, and they have not been broken in any meaningful way as of today.

In order to further support this claim, we have invited the community to attempt attacks against our MNIST and CIFAR10 networks in the form of a challenge. This has let us evaluate their robustness more thoroughly. The complete code, along with the description of the challenge, is available at `https://github.com/MadryLab/mnist_challenge` and `https://github.com/MadryLab/cifar10_challenge`.

1.2 An Optimization View on Adversarial Robustness

Much of our discussion will revolve around an optimization view of adversarial robustness. This perspective not only captures the phenomena we want to study in a precise manner, but will also inform our investigations. To this end, let us consider a standard classification task with an underlying data distribution $\mathcal{D}$ over pairs of examples $x \in \mathbb{R}^d$ and corresponding labels $y \in [k]$. We also assume that we are given a suitable loss function $L(\theta, x, y)$, for instance the cross-entropy loss for a neural network. As usual, $\theta \in \mathbb{R}^p$ is the set of model parameters. Our goal then is to find model parameters θ that minimize the risk $\mathbb{E}_{(x,y)\sim\mathcal{D}}[L(x, y, \theta)]$.

Empirical risk minimization (ERM) has been tremendously successful as a recipe for finding classifiers with small population risk. Unfortunately, ERM often does not yield models that are robust to adversarially crafted examples [BCM+13, SZS+14]. Formally, there are efficient algorithms ("adversaries") that take an example x belonging to class c_1 as input and find examples x^{adv} such that x^{adv} is very close to x but the model incorrectly classifies x^{adv} as belonging to class $c_2 \neq c_1$.

In order to *reliably* train models that are robust to adversarial attacks, it is necessary to augment the ERM paradigm appropriately. Instead of resorting to methods that directly focus on improving the robustness to specific attacks, our approach is to first propose a concrete *guarantee* that an adversarially robust model should satisfy. We then adapt our training methods towards achieving this guarantee.

The first step towards such a guarantee is to specify an *attack model*, i.e., a precise definition of the attacks our models should be resistant to. For each data point x, we introduce a set of allowed perturbations $\mathcal{S} \subseteq \mathbb{R}^d$ that formalizes the manipulative power of the adversary. In image classification, we choose $\mathcal{S}$ so that it captures perceptual similarity between images. For instance, the ℓ_∞-ball around x has recently been studied as a natural notion for adversarial perturbations [GSS15]. While we focus on robustness against ℓ_∞-bounded attacks in this chapter, we remark that more comprehensive notions of perceptual similarity are an important direction for future research.

Next, we modify the definition of population risk $\mathbb{E}_{\mathcal{D}}[L]$ by incorporating the above adversary. Instead of feeding samples from the distribution $\mathcal{D}$ directly into the loss L, we allow the adversary to perturb the input first. This gives rise to the following saddle point problem, which is our central object of study:

$$\min_{\theta} \rho(\theta), \quad \text{where} \quad \rho(\theta) = \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\max_{\delta\in\mathcal{S}} L(\theta, x+\delta, y)\right] . \tag{1.2.1}$$

Formulations of this type (and their finite-sample counterparts) have a long history in robust optimization, going back to Wald [Wal45]. It turns out that this formulation is also particularly useful in our context.

First, this formulation gives us a unifying perspective that encompasses much prior work on adversarial robustness. Our perspective stems from viewing the saddle point problem as the composition of an *inner maximization* problem and an *outer minimization* problem. Both of these problems have a natural interpretation in our context. The inner maximization problem aims to find an adversarial version of a given data point x that achieves a high loss. This is precisely the problem of attacking a given neural network. On the other hand, the goal of the outer minimization problem is to find model parameters so that the "adversarial loss" given by the inner attack problem is minimized. This is precisely the problem of training a robust classifier using adversarial training techniques.

Second, the saddle point problem specifies a clear goal that an ideal robust classifier should achieve, as well as a quantitative measure of its robustness. In particular, when the parameters θ yield a (nearly) vanishing risk, the corresponding model is perfectly robust to attacks specified by our attack model.

This chapter investigates the structure of this saddle point problem in the context of deep neural networks. These investigations then lead us to training techniques that produce models with high resistance to a wide range of adversarial attacks. Before turning to our contributions, we briefly review prior work on adversarial examples and describe in more detail how it fits into the above formulation.

1.2.1 A Unified View on Attacks and Defenses

Prior work on adversarial examples has focused on two main questions:

1. How can we produce strong adversarial examples, i.e., adversarial examples that fool a model with high confidence while requiring only a small perturbation?

2. How can we train a model so that there are no adversarial examples, or at least so that an adversary cannot find them easily?

Our perspective on the saddle point problem (1.2.1) gives answers to both these questions. On the attack side, prior work has proposed methods such as the

Fast Gradient Sign Method (FGSM) [GSS15] and multiple variations of it [KGB17]. FGSM is an attack for an ℓ_∞-bounded adversary and computes an adversarial example as

$$x + \varepsilon \operatorname{sgn}(\nabla_x L(\theta, x, y)).$$

One can interpret this attack as a simple one-step scheme for maximizing the inner part of the saddle point formulation. A more powerful adversary is the multi-step variant, which is essentially projected gradient descent (PGD) on the negative loss function

$$x^{t+1} = \Pi_{x+\mathcal{S}} \left(x^t + \alpha \operatorname{sgn}(\nabla_x L(\theta, x, y)) \right).$$

Other methods like FGSM with random perturbation have also been proposed [TPGDB17]. Clearly, all of these approaches can be viewed as specific attempts to solve the inner maximization problem in (1.2.1).

On the defense side, the training dataset is often augmented with adversarial examples produced by FGSM. This approach also directly follows from (1.2.1) when linearizing the inner maximization problem. To solve the simplified robust optimization problem, we replace every training example with its FGSM-perturbed counterpart. More sophisticated defense mechanisms such as training against multiple adversaries can be seen as better, more exhaustive approximations of the inner maximization problem.

1.3 Towards Universally Robust Networks

Past work on adversarial examples has usually focused on specific defensive mechanisms, or on attacks against such defenses. An important feature of formulation (1.2.1) is that attaining small adversarial loss gives a *guarantee* that no allowed attack will fool the network. By definition, no adversarial perturbations are possible because the loss is small for *all* perturbations allowed by our attack model. Hence, we now focus our attention on obtaining a good solution to (1.2.1).

Unfortunately, while the overall guarantee provided by the saddle point problem is evidently useful, it is not clear whether we can actually find a good solution in reasonable time. Solving the saddle point problem (1.2.1) involves tackling both a non-convex outer minimization problem *and* a non-concave inner maximization problem. One of our key contributions is demonstrating that, in practice, one *can* solve the saddle point problem after all. In particular, we now discuss an experimental exploration of the structure given by the non-concave inner problem. We argue that the loss landscape corresponding to this problem has a surprisingly tractable structure of local maxima. This structure also points towards projected gradient descent as the "ultimate" first-order adversary. Sections 1.4 and 2.6 then show that the resulting trained networks are indeed robust against a wide range of attacks, provided the networks are sufficiently large.

1.3.1 The Landscape of Adversarial Examples

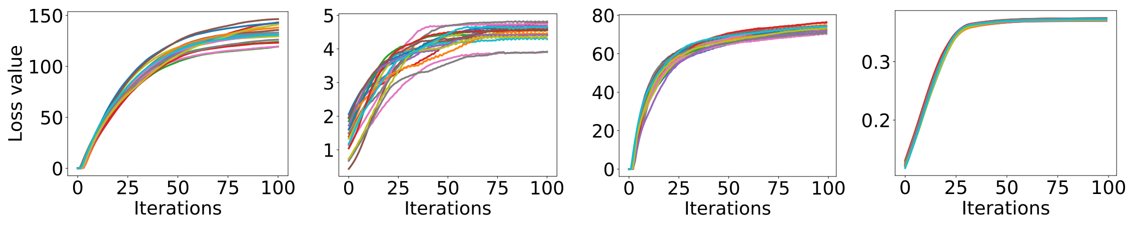
Recall that the inner problem corresponds to finding an adversarial example for a given network and data point (subject to our attack model). As this problem requires us to maximize a highly non-concave function, one would expect it to be intractable. Indeed, this is the conclusion reached by prior work which then resorted to linearizing the inner maximization problem [HXSS15, SYN18]. As pointed out above, this linearization approach yields well-known methods such as FGSM. While training against FGSM adversaries has shown some successes, recent work also highlights important shortcomings of this one-step approach [TPGDB17]—slightly more sophisticated adversaries can still find points of high loss.

To understand the inner problem in more detail, we investigate the landscape of local maxima for multiple models on MNIST and CIFAR10. The main tool in our experiments is projected gradient descent (PGD), since it is the standard method for large-scale constrained optimization. In order to explore a large part of the loss landscape, we re-start PGD from many points in the ℓ_∞ balls around data points from the respective evaluation sets.

Surprisingly, our experiments show that the inner problem *is* tractable after all, at least from the perspective of first-order methods. While there are many local maxima spread widely apart within $x_i + \mathcal{S}$, they tend to have very *well-concentrated* loss *values*. This echoes the folklore belief that training neural networks is possible because the loss (as a function of model parameters) typically has many local minima with very similar values.

Specifically, in our experiments we found the following phenomena:

- We observe that the loss achieved by the adversary increases in a fairly consistent way and plateaus rapidly when performing projected ℓ_∞ gradient descent for randomly chosen starting points inside $x + \mathcal{S}$ (see Figure 1.1).



(a) MNIST Standard training (b) MNIST Adversarial training (c) CIFAR10 Natural training (d) CIFAR10 Adversarial training

Figure 1.1: Cross-entropy loss values while creating an adversarial example from the MNIST and CIFAR10 evaluation datasets. The plots show how the loss evolves during 20 runs of projected gradient descent (PGD). Each run starts at a uniformly random point in the ℓ_∞-ball around the same natural example (additional plots for different examples appear in Figure 1.11). The adversarial loss plateaus after a small number of iterations. The optimization trajectories and final loss values are also fairly clustered, especially on CIFAR10. Moreover, the final loss values on adversarially trained networks are significantly smaller than on their standard counterparts.

- Investigating the concentration of maxima further, we observe that over a large number of random restarts, the loss of the final iterate follows a well-concentrated distribution without extreme outliers (see Figure 1.2; we verified this concentration based on 10^5 restarts).

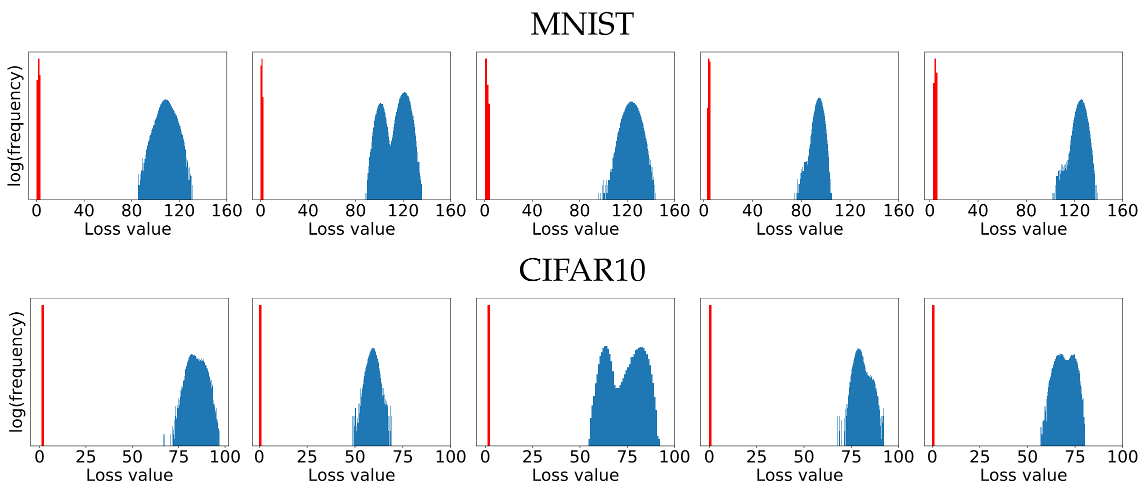


Figure 1.2: Values of the local maxima given by the cross-entropy loss for five examples from the MNIST and CIFAR10 evaluation datasets. For each example, we start projected gradient descent (PGD) from 10^5 uniformly random points in the ℓ_∞-ball around the example and iterate PGD until the loss plateaus. The blue histogram corresponds to the loss on a standard network, while the red histogram corresponds to the adversarially trained counterpart. The loss is significantly smaller for the adversarially trained networks, and the final loss values are very concentrated without any outliers.

- To demonstrate that maxima are noticeably distinct, we also measured the ℓ_2 distance and angles between all pairs of them and observed that distances are distributed close to the expected distance between two random points in the ℓ_∞ ball, and angles are close to 90°. Along the line segment between local maxima, the loss is convex, attaining its maximum at the endpoints and is reduced by a constant factor in the middle. Nevertheless, for the entire segment, the loss is considerably higher than that of a random point.

- Finally, we observe that the distribution of maxima suggests that the recently developed subspace view of adversarial examples is not fully capturing the richness of attacks [TPGDB17]. In particular, we observe adversarial perturbations with negative inner product with the gradient of the example, and deteriorating overall correlation with the gradient direction as the scale of perturbation increases.

All of this evidence points towards PGD being a "universal" adversary among first-order approaches, as we will see next.

1.3.2 First-Order Adversaries

Our experiments show that the local maxima found by PGD all have similar loss values, both for normally trained networks and adversarially trained networks. This concentration phenomenon suggests an intriguing view on the problem in which robustness against the PGD adversary yields robustness against *all* first-order adversaries, i.e., attacks that rely only on first-order information. As long as the adversary only uses gradients of the loss function with respect to the input, we conjecture that it will not find significantly better local maxima than PGD. We give more experimental evidence for this hypothesis in Section 2.6: if we train a network to be robust against PGD adversaries, it becomes robust against a wide range of other attacks as well.

Of course, our exploration with PGD does not preclude the existence of some isolated maxima with much larger function value. However, our experiments suggest that such better local maxima are *hard to find* with first order methods: even a large number of random restarts did not find function values with significantly different loss values. Incorporating the computational power of the adversary into the attack model should be reminiscent of the notion of *polynomially bounded* adversary that is a cornerstone of modern cryptography. There, this classic attack model allows the adversary to only solve problems that require at most polynomial computation time. Here, we employ an *optimization-based* view on the power of the adversary as it is more suitable in the context of machine learning. After all, we have not yet developed a thorough understanding of the computational complexity of many recent machine learning problems. However, the vast majority of optimization problems in ML is solved with first-order methods, and variants of SGD are the most effective way of training deep learning models in particular. Hence we believe that the class of attacks relying on first-order information is, in some sense, universal for the current practice of deep learning.

Put together, these two ideas chart the way towards machine learning models with *guaranteed* robustness. If we train the network to be robust against PGD adversaries, it will be robust against a wide range of attacks that encompasses all current approaches.

In fact, this robustness guarantee would become even stronger in the context of *black-box attacks*, i.e., attacks in which the adversary does not have a direct access to the target network. Instead, the adversary only has less specific information such as the (rough) model architecture and the training data set. One can view this attack model as an example of "zero order" attacks, i.e., attacks in which the adversary has no direct access to the classifier and is only able to evaluate it on chosen examples without gradient feedback.

We discuss transferability in Section 1.8. We observe that increasing network capacity and strengthening the adversary we train against (FGSM or PGD training, rather than standard training) improves resistance against transfer attacks. Also, as expected, the resistance of our best models to such attacks tends to be significantly larger than to the (strongest) first order attacks.

1.3.3 Descent Directions for Adversarial Training

The preceding discussion suggests that the inner optimization problem can be successfully solved by applying PGD. In order to train adversarially robust networks, we also need to solve the *outer* optimization problem of the saddle point formulation (1.2.1), that is find model parameters that minimize the "adversarial loss", the value of the inner maximization problem.

In the context of training neural networks, the main method for minimizing the loss function is Stochastic Gradient Descent (SGD). A natural way of computing the gradient of the outer problem, $\nabla_\theta \rho(\theta)$, is computing the gradient of the loss function at a maximizer of the inner problem. This corresponds to replacing the input points by their corresponding adversarial perturbations and normally training the network on the perturbed input. A priori, it is not clear that this is a valid descent direction for the saddle point problem. However, for the case of continuously differentiable functions, Danskin's theorem—a classic theorem in optimization—states this is indeed true and gradients at inner maximizers corresponds to descent directions for the saddle point problem.

Despite the fact that the exact assumptions of Danskin's theorem do not hold for our problem (the function is not continuously differentiable due to ReLU and max-pooling units, and we are only computing approximate maximizers of the inner problem), our experiments suggest that we can still use these gradients to optimize our problem. By applying SGD using the gradient of the loss at adversarial examples we can consistently reduce the loss of the saddle point problem during training, as can be seen in Figure 1.5. These observations suggest that we reliably optimize the saddle point formulation (1.2.1) and thus train robust classifiers. We formally state Danskin's theorem and describe how it applies to our problem in Section 1.7.

1.4 Network Capacity and Adversarial Robustness

Solving the problem from Equation (1.2.1) successfully is not sufficient to guarantee robust and accurate classification. We need to also argue that the *value* of the problem (i.e. the final loss we achieve against adversarial examples) is small, thus providing guarantees for the performance of our classifier. In particular, achieving a very small value corresponds to a perfect classifier, which is robust to adversarial inputs.

For a fixed set $\mathcal{S}$ of possible perturbations, the value of the problem is entirely dependent on the architecture of the classifier we are learning. Consequently, the architectural capacity of the model becomes a major factor affecting its overall performance. At a high level, classifying examples in a robust way requires a stronger classifier, since the presence of adversarial examples changes the decision boundary of the problem to a more complicated one (see Figure 1.3 for an illustration).

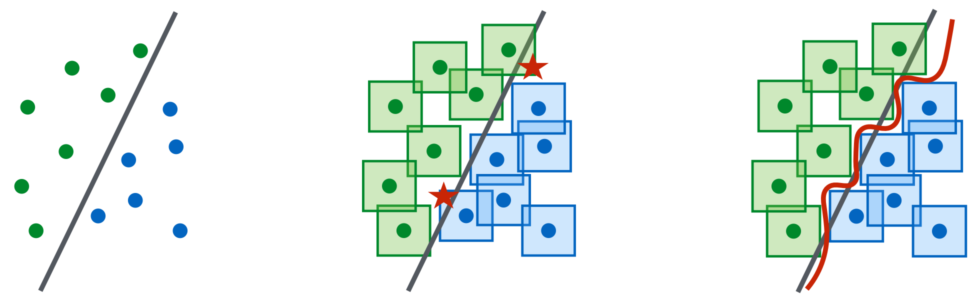


Figure 1.3: A conceptual illustration of standard vs. adversarial decision boundaries. Left: A set of points that can be easily separated with a simple (in this case, linear) decision boundary. Middle: The simple decision boundary does not separate the ℓ_∞-balls (here, squares) around the data points. Hence there are adversarial examples (the red stars) that will be misclassified. Right: Separating the ℓ_∞-balls requires a significantly more complicated decision boundary. The resulting classifier is robust to adversarial examples with bounded ℓ_∞-norm perturbations.

Our experiments verify that capacity is crucial for robustness, as well as for the ability to successfully train against strong adversaries. For the MNIST dataset, we consider a simple convolutional network and study how its behavior changes against different adversaries as we keep doubling the size of network (i.e. double the number of convolutional filters and the size of the fully connected layer). The initial network has a convolutional layer with 2 filters, followed by another convolutional layer with 4 filters, and a fully connected hidden layer with 64 units. Convolutional layers are followed by 2×2 max-pooling layers and adversarial examples are constructed with $\varepsilon = 0.3$. The results are in Figure 1.4.

For the CIFAR10 dataset, we used a ResNet model [HZRS16]. We performed data augmentation using random crops and flips, as well as per image standarization. To increase the capacity, we modified the network incorporating wider layers by a factor of 10. This results in a network with 5 residual units with (16, 160, 320, 640) filters each. This network can achieve an accuracy of 95.2% when trained with natural examples. Adversarial examples were constructed with $\varepsilon = 8$. Results on capacity experiments appear in Figure 1.4.

We observe the following phenomena:

Capacity alone helps. We observe that increasing the capacity of the network when training using only natural examples (apart from increasing accuracy on these examples) increases the robustness against one-step perturbations. This effect is greater when considering adversarial examples with smaller ε.

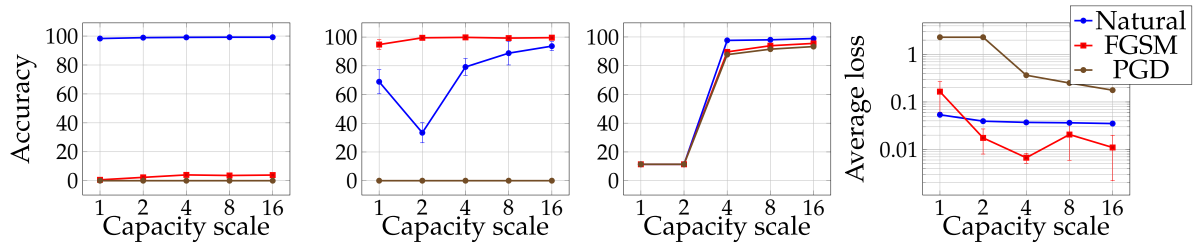
FGSM adversaries don't increase robustness (for large ε). When training the network using adversarial examples generated with the FGSM, we observe that the network overfits to these adversarial examples. This behavior is known as label leaking [KGB17] and stems from the fact that the adversary produces a very restricted set of adversarial examples that the network can overfit to. These networks have poor performance on natural examples and don't exhibit any kind of robustness against PGD adversaries. For the case of smaller ε the loss is ofter linear enough in the ℓ_∞-ball around natural examples, that FGSM finds adversarial examples close to those found by PGD thus being a reasonable adversary to train against.

Weak models may fail to learn non-trivial classifiers. In the case of small capacity networks, attempting to train against a strong adversary (PGD) prevents the network from learning anything meaningful. The network converges to always predicting a fixed class, even though it could converge to an accurate classifier through standard training. The small capacity of the network forces the training procedure to sacrifice performance on natural examples in order to provide any kind of robustness against adversarial inputs.

The value of the saddle point problem decreases as we increase the capacity. Fixing an adversary model, and training against it, the value of (1.2.1) drops as capacity increases, indicating the the model can fit the adversarial examples increasingly well.

More capacity and stronger adversaries decrease transferability. Either increasing the capacity of the network, or using a stronger method for the inner optimization problem reduces the effectiveness of transferred adversarial inputs. We validate this experimentally by observing that the correlation between gradients from the source and the transfer network, becomes less significant as capacity increases. We describe our experiments in Section 1.8.

MNIST



CIFAR10

	(a) Simple	(a) Wide	(b) Simple	(b) Wide	(c) Simple	(c) Wide	(d) Simple	(d) Wide
Natural	92.7%	95.2%	87.4%	90.3%	79.4%	87.3%	0.00357	0.00371
FGSM	27.5%	32.7%	90.9%	95.1%	51.7%	56.1%	0.0115	0.00557
PGD	0.8%	3.5%	0.0%	0.0%	43.7%	45.8%	1.11	0.0218

(a) Standard training (b) FGSM training (c) PGD training (d) Training Loss

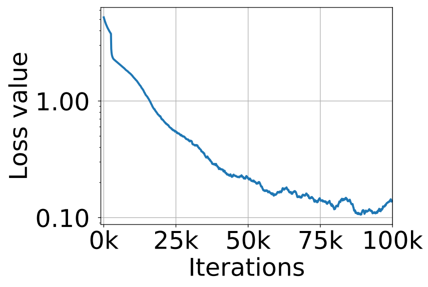
Figure 1.4: The effect of network capacity on the performance of the network. We trained MNIST and CIFAR10 networks of varying capacity on: (a) natural examples, (b) with FGSM-made adversarial examples, (c) with PGD-made adversarial examples. In the first three plots/tables of each dataset, we show how the standard and adversarial accuracy changes with respect to capacity for each training regime. In the final plot/table, we show the value of the cross-entropy loss on the adversarial examples the networks were trained on. This corresponds to the value of our saddle point formulation (1.2.1) for different sets of allowed perturbations.

1.5 Experiments: Adversarially Robust Deep Learning Models

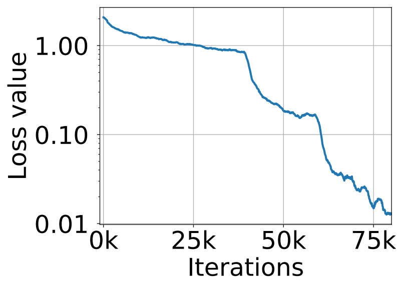
Following the understanding of the problem we developed in previous sections, we can now apply our proposed approach to train robust classifiers. As our experiments so far demonstrated, we need to focus on two key elements: a) train a sufficiently high capacity network, b) use the strongest possible adversary.

For both MNIST and CIFAR10, the adversary of choice will be projected gradient descent (PGD) starting from a random perturbation around the natural example. This corresponds to our notion of a "complete" first-order adversary, an algorithm that can efficiently maximize the loss of an example using only first order information. Since we are training the model for multiple epochs, there is no benefit from restarting PGD multiple times per batch—a new start will be chosen the next time each example is encountered.

When training against that adversary, we observe a steady decrease in the training loss of adversarial examples, illustrated in Figure 1.5. This behavior indicates that we are indeed successfully solving our original optimization problem during training.



(a) MNIST



(b) CIFAR10

Figure 1.5: Cross-entropy loss on adversarial examples during training. The plots show how the adversarial loss on training examples evolves during training the MNIST and CIFAR10 networks against a PGD adversary. The sharp drops in the CIFAR10 plot correspond to decreases in training step size. These plots illustrate that we can consistently reduce the value of the inner problem of the saddle point formulation (1.2.1), thus producing an increasingly robust classifier.

We evaluate the trained models against a range of adversaries. We illustrate our results in Table 1.1 for MNIST and Table 1.2 for CIFAR10. The adversaries we consider are:

- White-box attacks with PGD for a different number of of iterations and restarts, denoted by source A.

- White-box attacks with PGD using the Carlini-Wagner (CW) loss function [CW17b] (directly optimizing the difference between correct and incorrect logits). This is denoted as CW, where the corresponding attack with a high confidence parameter ($\kappa = 50$) is denoted as CW+.

- Black-box attacks from an independently trained copy of the network, denoted A′.
- Black-box attacks from a version of the same network trained only on natural examples, denoted A_{nat}.
- Black-box attacks from a different convolution architecture, denoted B, described in Tramer et al. 2017 [TPGDB17].

MNIST. We run 40 iterations of projected gradient descent as our adversary, with a step size of 0.01 (we choose to take gradient steps in the ℓ_∞-norm, i.e. adding the sign of the gradient, since this makes the choice of the step size simpler). We train and evaluate against perturbations of size $\varepsilon = 0.3$. We use a network consisting of two convolutional layers with 32 and 64 filters respectively, each followed by 2×2 max-pooling, and a fully connected layer of size 1024. When trained with natural examples, this network reaches 99.2% accuracy on the evaluation set. However, when evaluating on examples perturbed with FGSM the accuracy drops to 6.4%. The resulting adversarial accuracies are reported in Table 1.1. Given that the resulting MNIST model is very robust to ℓ_∞-bounded adversaries, we investigated the learned parameters in order to understand how they affect adversarial robustness. The results of the investigation are presented in 1.9. In particular, we found that the first convolutional layer of the network is learning to threshold input pixels while other weights tend to be sparser.

CIFAR10. For the CIFAR10 dataset, we use the two architectures described in 1.4 (the original ResNet and its $10\times$ wider variant). We trained the network against a PGD adversary with ℓ_∞ projected gradient descent again, this time using 7 steps of size 2, and a total $\varepsilon = 8$. For our hardest adversary we chose 20 steps with the same

Method	Steps	Restarts	Source	Accuracy
Natural	-	-	-	98.8%
FGSM	-	-	A	95.6%
PGD	40	1	A	93.2%
PGD	100	1	A	91.8%
PGD	40	20	A	90.4%
PGD	100	20	A	**89.3%**
Targeted	40	1	A	92.7%
CW	40	1	A	94.0%
CW+	40	1	A	93.9%
FGSM	-	-	A′	96.8%
PGD	40	1	A′	96.0%
PGD	100	20	A′	**95.7%**
CW	40	1	A′	97.0%
CW+	40	1	A′	96.4%
FGSM	-	-	B	**95.4%**
PGD	40	1	B	96.4%
CW+	-	-	B	95.7%

Table 1.1: MNIST: Performance of the adversarially trained network against different adversaries for $\varepsilon = 0.3$. For each model of attack we show the most successful attack with bold. The source networks used for the attack are: the network itself (A) (white-box attack), an indepentenly initialized and trained copy of the network (A′), architecture B from [TPGDB17] (B).

settings, since other hyperparameter choices didn't offer a significant decrease in accuracy. The results of our experiments appear in Table 1.2.

The adversarial robustness of our network is significant, given the power of iterative adversaries, but still far from satisfactory. We believe that these results can be improved by further pushing along these directions, and training networks of larger capacity.

Resistance for different values of ε and ℓ_2-bounded attacks. In order to perform a broader evaluation of the adversarial robustness of our models, we run two additional experiments. On one hand, we investigate the resistance to ℓ_∞-bounded attacks for different values of ε. On the other hand, we examine the resistance of our model to attacks that are bounded in ℓ_2-norm as opposed to ℓ_∞-norm. In the

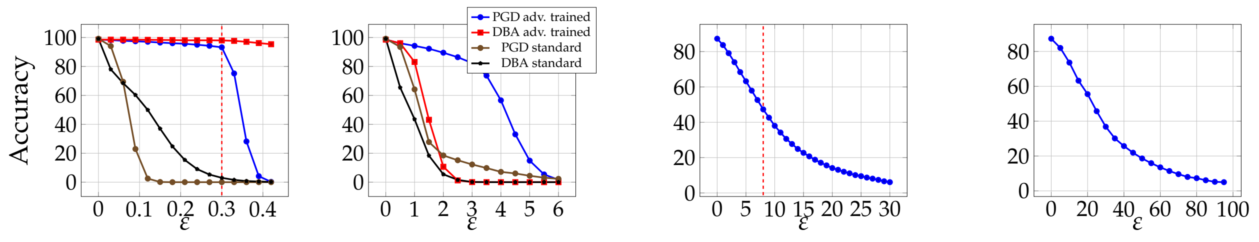
Method	Steps	Source	Accuracy
Natural	-	-	87.3%
FGSM	-	A	56.1%
PGD	7	A	50.0%
PGD	20	A	**45.8%**
CW	30	A	46.8%
FGSM	-	A′	67.0%
PGD	7	A′	**64.2%**
CW	30	A′	78.7%
FGSM	-	A_{nat}	85.6%
PGD	7	A_{nat}	86.0%

Table 1.2: CIFAR10: Performance of the adversarially trained network against different adversaries for $\varepsilon = 8$. For each model of attack we show the most effective attack in bold. The source networks considered for the attack are: the network itself (A) (white-box attack), an independtly initialized and trained copy of the network (A′), a copy of the network trained on natural examples (A_{nat}).

case of ℓ_2-bounded PGD we take steps in the gradient direction (not the sign of it) and normalize the steps to be of fixed norm to facilitate step size tuning. For all PGD attacks, we use 100 steps and set the step size to be $2.5 \cdot \varepsilon/100$ to ensure that we can reach the boundary of the ε-ball from any starting point within it (and still allow for movement on the boundary). Note that the models were training against ℓ_∞-bounded attacks with the original value of $\varepsilon = 0.3$, for MNIST, and $\varepsilon = 8$ for CIFAR10. The results appear in Figure 1.6.

We observe that for smaller ε than the one used during training the models achieve equal or higher accuracy, as expected. For MNIST, we notice a large drop in robustness for slightly large ε values, potentially due to the fact that the threshold operators learned are tuned to the exact value of ε used during training (1.9). In contrast, the decay for the case of CIFAR10 is smoother.

For the case of ℓ_2-bounded attacks on MNIST, we observe that PGD is unable to find adversarial examples even for quite large values of ε, e.g., $\varepsilon = 4.5$. To put this value of ε into perspective, we provide a sample of corresponding adversarial examples in Figure 1.12 of 1.11. We observe that these perturbations are significant enough that they would change the ground-truth label of the images



(a) MNIST, ℓ_∞-norm (b) MNIST, ℓ_2-norm (c) CIFAR10, ℓ_∞-norm (d) CIFAR10, ℓ_2-norm

Figure 1.6: Performance of our adversarially trained networks against PGD adversaries of different strength. The MNIST and CIFAR10 networks were trained against $\varepsilon = 0.3$ and $\varepsilon = 8$ PGD ℓ_∞ adversaries respectively (the training ε is denoted with a red dashed lines in the ℓ_∞ plots). In the case of the MNIST adversarially trained networks, we also evaluate the performance of the Decision Boundary Attack (DBA) [BRB17] with 2000 steps and PGD on standard and adversarially trained models. We observe that for ε less or equal to the value used during training, the performance is equal or better. For MNIST there is a sharp drop shortly after. Moreover, we observe that the performance of PGD on the MNIST ℓ_2-trained networks is poor and significantly overestimates the robustness of the model. This is potentially due to the threshold filters learned by the model masking the loss gradients (the decision-based attack does not utilize gradients).

and it is thus unlikely that our models are actually that robust. Indeed, subsequent work [LCWC18, SRBB19] has found that PGD is in fact overestimating the ℓ_2-robustness of this model. This behavior is potentially due to the fact that the learned threshold filters (1.9) mask the gradient, preventing PGD from maximizing the loss. Attacking the model with a decision-based attack [BRB17] which does not rely on model gradients reveals that the model is significantly more brittle against ℓ_2-bounded attacks. Nevertheless, the ℓ_∞-trained model is still more robust to ℓ_2 attacks compared to a standard model.

1.6 Related Work

Due to the large body of work on adversarial examples we focus only on the most related papers here. Before we compare our contributions, we remark that robust optimization has been studied outside deep learning for multiple decades (see [BTEGN09] for an overview of this field). We also want to note that the study of adversarial ML predates the widespread use of deep neural networks [DDSV04, GR06] (see [BR18] for an overview of earlier work).

Adversarial training was introduced in [GSS15], however the adversary utilized was quite weak—it relied on linearizing the loss around the data points. As a result, while these models were robust against this particular adversary, they were completely vulnerable to slightly more sophisticated adversaries utilizing iterative attacks.

Recent work on adversarial training on ImageNet also observed that the model capacity is important for adversarial training [KGB17]. In contrast to this paper, we find that training against multi-step methods (PGD) *does* lead to resistance against such adversaries.

In [HXSS15] and [SYN18] a version of the min-max optimization problem is also considered for adversarial training. There are, however, three important differences between the formerly mentioned result and the present chapter. Firstly, the authors claim that the inner maximization problem can be difficult to solve, whereas we explore the loss surface in more detail and find that randomly re-started projected gradient descent often converges to solutions with comparable quality. This shows that it is possible to obtain sufficiently good solutions to the inner maximization problem, which offers good evidence that deep neural network can be immunized against adversarial examples. Secondly, they consider only one-step adversaries, while we work with multi-step methods. Additionally, while the experiments in [SYN18] produce promising results, they are only evaluated against FGSM. However, FGSM-only evaluations are not fully reliable. One evidence for that is that [SYN18] reports 70% accuracy for $\varepsilon = 0.7$, but any adversary that is allowed to perturb each pixel by more than 0.5 can construct a uniformly gray image, thus fooling any classifier.

A more recent paper [TPGDB17] also explores the transferability phenomenon. This exploration focuses mostly on the region around natural examples where the loss is (close to) linear. When large perturbations are allowed, this region does not give a complete picture of the adversarial landscape. This is confirmed by our experiments, as well as pointed out by [TPGDB17].

1.7 Statement and Application of Danskin's Theorem

Recall that our goal is to minimize the value of the saddle point problem

$$\min_\theta \rho(\theta), \quad \text{where} \quad \rho(\theta) = \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\max_{\delta\in\mathcal{S}} L(\theta, x+\delta, y)\right] .$$

In practice, we don't have access to the distribution $\mathcal{D}$ so both the gradients and the value of $\rho(\theta)$ will be computed using sampled input points. Therefore we can consider –without loss of generality– the case of a single random example x with label y, in which case the problem becomes

$$\min_\theta \max_{\delta\in\mathcal{S}} g(\theta,\delta), \quad \text{where} \quad g(\theta,\delta) = L(\theta, x+\delta, y) .$$

If we assume that the loss L is continuously differentiable in θ, we can compute a descent direction for θ by utilizing the classical theorem of Danskin.

Theorem 1.7.1 (Danskin). *Let $\mathcal{S}$ be nonempty compact topological space and $g : \mathbb{R}^n \times \mathcal{S} \to \mathbb{R}$ be such that $g(\cdot,\delta)$ is differentiable for every $\delta \in \mathcal{S}$ and $\nabla_\theta g(\theta,\delta)$ is continuous on $\mathbb{R}^n \times \mathcal{S}$. Also, let $\delta^*(\theta) = \{\delta \in \arg\max_{\delta\in\mathcal{S}} g(\theta,\delta)\}$.*

Then the corresponding max-function

$$\phi(\theta) = \max_{\delta\in\mathcal{S}} g(\theta,\delta)$$

is locally Lipschitz continuous, directionally differentiable, and its directional derivatives satisfy

$$\phi'(\theta,h) = \sup_{\delta\in\delta^*(\theta)} h^\top \nabla_\theta g(\theta,\delta) .$$

In particular, if for some $\theta \in \mathbb{R}^n$ the set $\delta^*(\theta) = \{\delta^*_\theta\}$ is a singleton, the the max-function is differentiable at θ and

$$\nabla\phi(\theta) = \nabla_\theta g(\theta,\delta^*_\theta)$$

The intuition behind the theorem is that since gradients are local objects, and the function $\phi(\theta)$ is locally the same as $g(\theta, \delta^*_\theta)$ their gradients will be the same. The theorem immediately gives us the following corollary, stating the we can indeed compute gradients for the saddle point by computing gradients at the inner optimizers.

Corollary 1.7.2. *Let $\overline{\delta}$ be such that $\overline{\delta} \in \mathcal{S}$ and is a maximizer for $\max_\delta L(\theta, x + \delta, y)$. Then, as long as it is nonzero, $-\nabla_\theta L(\theta, x + \overline{\delta}, y)$ is a descent direction for $\phi(\theta) = \max_{\delta \in \mathcal{S}} L(\theta, x + \delta, y)$.*

Proof of Corollary 1.7.2. We apply Theorem 1.7.1 to $g(\theta, \delta) := L(\theta, x + \delta, y)$ and $\mathcal{S} = B_{\|\cdot\|}(\varepsilon)$. We see that the directional derivative in the direction of $h = \nabla_\theta L(\theta, x + \overline{\delta}, y)$ satisfies

$$\phi'(\theta, h) = \sup_{\delta \in \delta^*(\theta)} h^\top \nabla_\theta L(\theta, x + \delta, y) \geq h^\top h = \|\nabla_\theta L(\theta, x + \overline{\delta}, y)\|_2^2 \geq 0 \, .$$

If this gradient is nonzero, then the inequality above is strict. Therefore it gives a descent direction. □

A technical issue is that, since we use ReLU and max-pooling units in our neural network architecture, the loss function is not continuously differentiable. Nevertheless, since the set of discontinuities has measure zero, we can assume that this will not be an issue in practice, as we will never encounter the problematic points.

Another technical issue is that, due to the not concavity of the inner problem, we are not able to compute global maximizers, since PGD will converge to local maxima. In such cases, we can consider a subset $\mathcal{S}'$ of $\mathcal{S}$ such that the local maximum is a global maximum in the region $\mathcal{S}'$. Applying the theorem for $\mathcal{S}'$ gives us that the gradient corresponds to a descent direction for the saddle point problem when the adversary is constrained in $\mathcal{S}'$. Therefore if the inner maximum is a true adversarial example for the network, then SGD using the gradient at that point will decrease the loss value at this particular adversarial examples, thus making progress towards a robust model.

These arguments suggest that the conclusions of the theorem are still valid in our saddle point problem, and –as our experiments confirm– we can solve it reliably.

1.8 Transferability

A lot of recent literature on adversarial training discusses the phenomenon of transferability [SZS+14, GSS15, TPGDB17]—adversarial examples transfer between independently trained networks. This raises concerns for practical applications, since it suggests that deep networks are vulnerable to attacks, *even when there is no direct access to the target network*.

This phenomenon is further confirmed by our current experiments. [2] Moreover, we notice that the extent to which adversarial examples transfer decreases as we increase either network capacity or the power of the adversary used for training the network. This serves as evidence for the fact that the transferability phenomenon can be alleviated by using high capacity networks in conjunction with strong oracles for the inner optimization problem.

MNIST. In an attempt to understand these phenomena we inspect the loss functions corresponding to the trained models we used for testing transferability. More precisely, we compute angles between gradients of the loss functions evaluated over a large set of input examples, and plot their distribution. Similarly, we plot the value of the loss functions between clean and perturbed examples for both the source and transfer networks. In Figure 1.8 we plot our experimental findings on the MNIST dataset for $\varepsilon = 0.3$. We consider a large standard network (two convolutional layers of sizes 32 and 64, and a fully connected layer of size 1024), which we train twice starting with different initializations. We plot the distribution

[2]Our experiments involve transferability between networks with the same architecture (potentially with layers of varying sizes), trained with the same method, but with different random initializations. The reason we consider these models rather than highly different architectures is that they are likely the worst case instances for transferability.

of angles between gradients for the same test image in the two resulting networks (orange histograms), noting that they are somewhat correlated. As opposed to this, we see that pairs of gradients for random pairs of inputs for one architecture are as uncorrelated as they can be (blue histograms), since the distribution of their angles looks Gaussian.

Next, we run the same experiment on a very large standard network (two convolutional layers of sizes 64 and 128, and a fully connected layer of size 1024). We notice a mild increase in classification accuracy for transferred examples.

Finally, we repeat the same set of experiments, after training the large and very large networks against the FGSM adversary. We notice that gradients between the two architectures become significantly less correlated. Also, the classification accuracy for transferred examples increases significantly compared to the standard networks.

We further plot how the value of the loss function changes when moving from the natural input towards the adversarially perturbed input (in Figure 1.8 we show these plots for four images in the MNIST test dataset), for each pair of networks we considered. We observe that, while for the naturally trained networks, when moving towards the perturbed point, the value of the loss function on the transfer architecture tends to start increasing soon after it starts increasing on the source architecture. In contrast, for the stronger models, the loss function on the transfer network tends to start increasing later, and less aggressively.

CIFAR10. For the CIFAR10 dataset, we investigate the transferability of the FGSM and PGD adversaries between our simple and wide architectures, each trained on natural, FGSM and PGD examples. Transfer accuracies for the FGSM adversary and PGD adversary between all pairs of such configurations (model + training method) with independently random weight initialization are given in tables 1.3 and 1.4 respectively. The results exhibit the following trends:

- **Stronger adversaries decrease transferability:** In particular, transfer attacks between two PGD-trained models are less successful than transfer attacks between their standard counterparts. Moreover, adding PGD training helps with transferability from all adversarial datasets, *except* for those with source a PGD-trained model themselves. This applies to both FGSM attacks and PGD attacks.

- **Capacity decreases transferability:** In particular, transfer attacks between two PGD-trained wide networks are less successful than transfer attacks between their simple PGD-trained counterparts. Moreover, with few close exceptions, changing the architecture from simple to wide (and keeping the training method the same) helps with transferability from all adversarial datasets.

We additionally plotted how the loss of a network behaves in the direction of FGSM and PGD examples obtained from itself and an independently trained copy; results for the simple standard network and the wide PGD trained network are given in Table 1.7. As expected, we observe the following phenomena:

- sometimes, the FGSM adversary manages to increase loss faster near the natural example, but as we move towards the boundary of the ℓ_∞ box of radius ε, the PGD attack always achieves higher loss.
- the transferred attacks do worse than their white-box counterparts in terms of increasing the loss;
- and yet, the transferred PGD attacks dominate the white-box FGSM attacks for the standard network (and sometimes for the PGD-trained one too).

Source / Target	Simple (standard training)	Simple (FGSM training)	Simple (PGD training)	Wide (natural training)	Wide (FGSM training)	Wide (PGD training)
Simple (standard training)	32.9%	74.0%	73.7%	27.6%	71.8%	76.6%
Simple (FGSM training)	64.2%	90.7%	60.9%	61.5%	90.2%	67.3%
Simple (PGD training)	77.1%	78.1%	60.2%	77.0%	77.9%	66.3%
Wide (standard training)	34.9%	78.7%	80.2%	21.3%	75.8%	80.6%
Wide (FGSM training)	64.5%	93.6%	69.1%	53.7%	92.2%	72.8%
Wide (PGD training)	85.8%	86.6%	73.3%	85.6%	86.2%	67.0%

Table 1.3: CIFAR10: black-box FGSM attacks. We create FGSM adversarial examples with $\varepsilon = 8$ from the evaluation set on the source network, and then evaluate them on an independently initialized target network.

Source / Target	Simple (standard training)	Simple (FGSM training)	Simple (PGD training)	Wide (natural training)	Wide (FGSM training)	Wide (PGD training)
Simple (standard training)	6.6%	71.6%	71.8%	1.4%	51.4%	75.6%
Simple (FGSM training)	66.3%	40.3%	58.4%	65.4%	26.8%	66.2%
Simple (PGD training)	78.1%	78.2%	57.7%	77.9%	78.1%	65.2%
Wide (standard training)	10.9%	79.6%	79.1%	0.0%	51.3%	79.7%
Wide (FGSM training)	67.6%	51.7%	67.4%	56.5%	0.0%	71.6%
Wide (PGD training)	86.4%	86.8%	72.1%	86.0%	86.3%	64.2%

Table 1.4: CIFAR10: black-box PGD attacks. We create PGD adversarial examples with $\varepsilon = 8$ for 7 iterations from the evaluation set on the source network, and then evaluate them on an independently initialized target network.

Model \ Adversary	Natural	FGSM	FGSM random	PGD (7 steps)	PGD (20 steps)
Simple (standard training)	92.7%	27.5%	19.6%	1.2%	0.8%
Simple (FGSM training)	87.4%	90.9%	90.4%	0.0%	0.0%
Simple (PGD training)	79.4%	51.7%	55.9%	47.1%	43.7%
Wide (standard training)	95.2%	32.7%	25.1%	4.1%	3.5%
Wide (FGSM training)	90.3%	95.1%	95.0%	0.0%	0.0%
Wide (PGD training)	87.3%	56.1%	60.3%	50.0%	45.8%

Table 1.5: CIFAR10: white-box attacks for $\varepsilon = 8$. For each architecture and training method, we list the accuracy of the resulting network on the full CIFAR10 evaluation set of 10,000 examples. The FGSM random method is the one suggested by [TPGDB17], whereby we first do a small random perturbation of the natural example, and the apply FGSM to that.

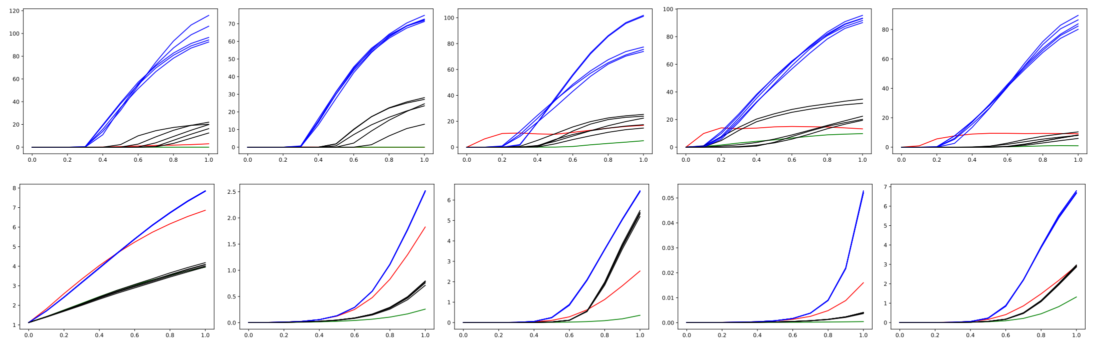
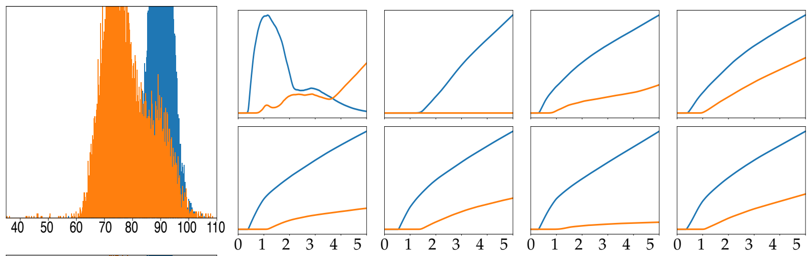


Figure 1.7: CIFAR10: change of loss function in the direction of white-box and black-box FGSM and PGD examples with $\varepsilon = 8$ for the same five natural examples. Each line shows how the loss changes as we move from the natural example to the corresponding adversarial example. Top: simple naturally trained model. Bottom: wide PGD trained model. We plot the loss of the original network in the direction of the FGSM example for the original network (red lines), 5 PGD examples for the original network obtained from 5 random starting points (blue lines), the FGSM example for an independently trained copy network (green lines) and 5 PGD examples for the copy network obtained from 5 random starting points (black lines). All PGD attacks use 100 steps with step size 0.3.

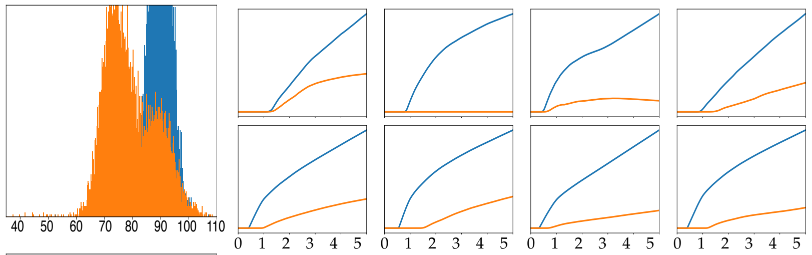
	Source	Transfer
Clean	99.2%	99.2%
FGSM	3.9%	41.9%
PGD	0.0%	26.0%

Large network, standard training



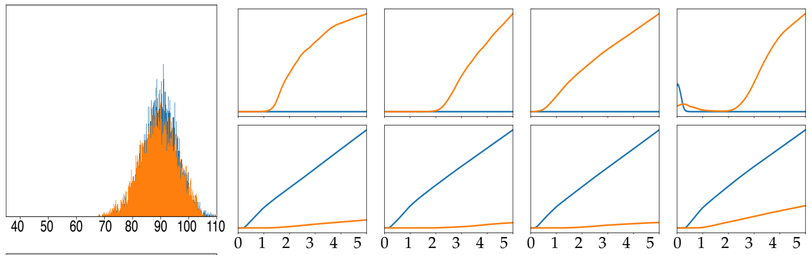
	Source	Transfer
Clean	99.2%	99.3%
FGSM	7.2%	44.6%
PGD	0.0%	35.0%

Very large network, standard training



	Source	Transfer
Clean	92.9%	96.1%
FGSM	99.9%	62.0%
PGD	0.0%	54.1%

Large network, FGSM training



	Source	Transfer
Clean	96.4%	97.8%
FGSM	99.4%	71.6%
PGD	0.0%	60.6%

Very large network, FGSM training

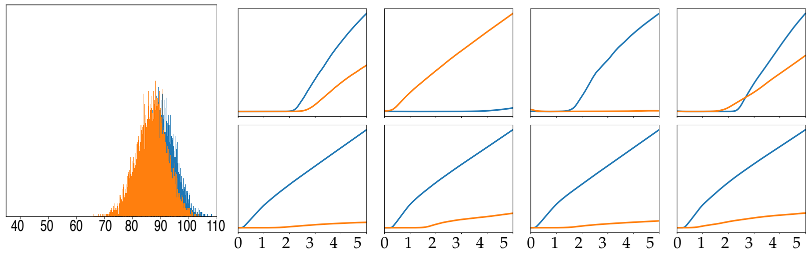


Figure 1.8: Transferability experiments for four different instances (standard large and very large networks, and FGSM-trained large and very large networks, respectively). For each instance we ran the same training algorithm twice, starting from different initializations. Tables on the left show the accuracy of the networks against three types of input (clean, perturbed with FGSM, perturbed with PGD ran for 40 steps); the first column shows the resilience of the first network against examples produced using its own gradients, the second column shows resilience of the second network against examples transferred from the former network. The histograms reflect angles between pairs of gradients corresponding to the same inputs versus the baseline consisting of angles between gradients from random pairs of points. Images on the right hand side reflect how the loss functions of the native and the transfer network change when moving in the direction of the perturbation; the perturbation is at 1 on the horizontal axis. Plots in the top row are for FGSM perturbations, plots in the bottom row are for PGD perturbations produced over 40 iterations.

1.9 MNIST Inspection

The robust MNIST model described so far is small enough that we can visually inspect most of its parameters. Doing so will allow us to understand how it is different from a standard network and what are the general characteristics of a network that is robust against ℓ_∞ adversaries. We will compare three different networks: a standard model, and two adversarially trained ones. The latter two models are identical, modulo the random weight initialization, and were used as the public and secret models used for our robustness challenge.

Initially, we examine the first convolutional layer of each network. We observe that the robust models only utilize 3 out of the total 32 filters, and for each of these filters only one weight is non-zero. By doing so, the convolution degrades into a scaling of the original image. Combined with the bias and the ReLU that follows, this results in a *thresholding filter*, or equivalently $\mathrm{ReLU}(\alpha x - \beta)$ for some constants α, β. From the perspective of adversarial robustness, thresholding filters are immune to any perturbations on pixels with value less than $\beta - \varepsilon$. We visualize a sample of the filters in Figure 1.9 (plots a, c, and e).

Having observed that the first layer of the network essentially maps the original image to three copies thresholded at different values, we examine the second convolutional layer of the classifier. Again, the filter weights are relatively sparse and have a significantly wider value range than the standard version. Since only three channels coming out of the first layer matter, is follows (and is verified) that the only relevant convolutional filters are those that interact with these three channels. We visualize a sample of the filters in Figure 1.9 (plots b, d, and f).

Finally, we examine the softmax/output layer of the network. While the weights seem to be roughly similar between all three version of the network, we notice a significant difference in the class biases. The adversarially trained networks heavily utilize class biases (far from uniform), and do so in a way very similar to each other. A plausible explanation is that certain classes tend to be very vulnerable to adversarial perturbations, and the network learns to be more conservative in

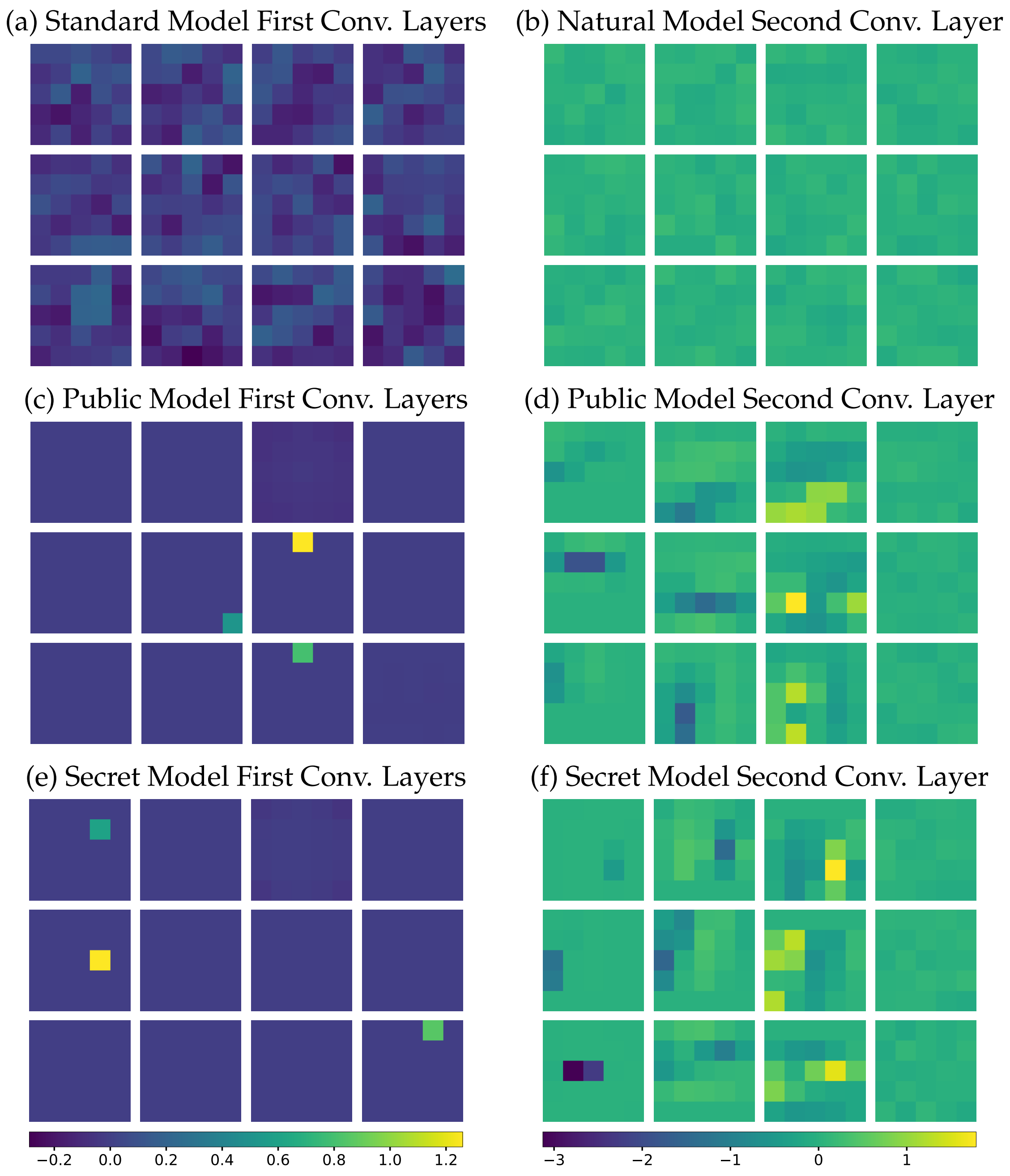
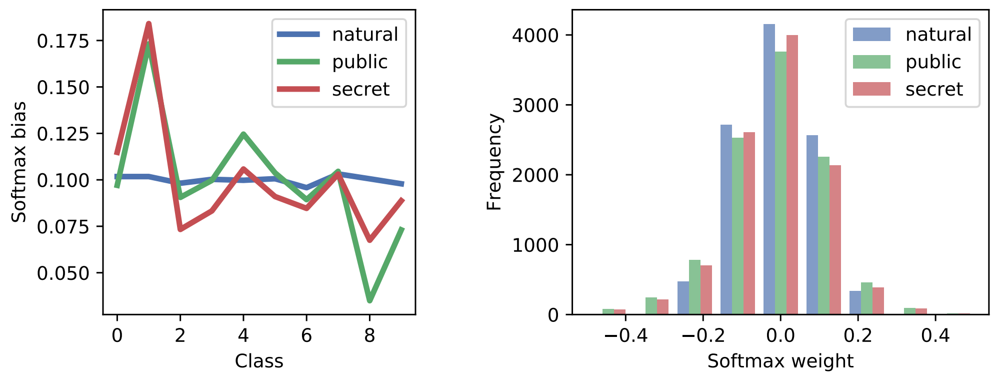


Figure 1.9: Visualizing a sample of the convolutional filters. For the standard model (a,b) we visualize random filters, since there is no observable difference in any of them. For the first layer of robust networks we make sure to include the 3 non-zero filters. For the second layer, the first three columns represent convolutional filters that utilize the 3 non-zero channels, and we choose the most interesting ones (larger range of values). We observe that adversarially trained networks have significantly more concentrated weights. Moreover, the first convolutional layer degrades into a few thresholding filters.

predicting them. The plots can be found in Figure 1.10.



(a) Softmax biases for each class (b) Distribution of softmax weights

Figure 1.10: Softmax layer examination. For each network we create a histogram of the layer's weights and plot the per-class bias. We observe that while weights are similar (slightly more concentrated for the standard one) the biases are far from uniform and with a similar pattern for the two adversarially trained networks.

All of the "tricks" described so far seem intuitive to a human and would seem reasonable directions when trying to increase the adversarial robustness of a classifier. We emphasize the none of these modifications were hard-coded in any way and they were all *learned solely through adversarial training*. We attempted to manually introduce these modifications ourselves, aiming to achieve adversarial robustness without adversarial training, but with no success. A simple PGD adversary could fool the resulting models on all the test set examples.

1.10 Conclusion

Our findings provide evidence that deep neural networks can be made resistant to adversarial attacks. As our theory and experiments indicate, we can design reliable adversarial training methods. One of the key insights behind this is the unexpectedly regular structure of the underlying optimization task: even though the relevant problem corresponds to the maximization of a highly non-concave function with many distinct local maxima, their *values* are highly concentrated.

Overall, our findings give us hope that adversarially robust deep learning models may be within current reach.

For the MNIST dataset, our networks are very robust, achieving high accuracy for a wide range of powerful ℓ_∞-bound adversaries and large perturbations. Our experiments on CIFAR10 have not reached the same level of performance yet. However, our results already show that our techniques lead to significant increase in the robustness of the network. We believe that further exploring this direction will lead to adversarially robust networks for this dataset.

1.11 Supplementary Figures

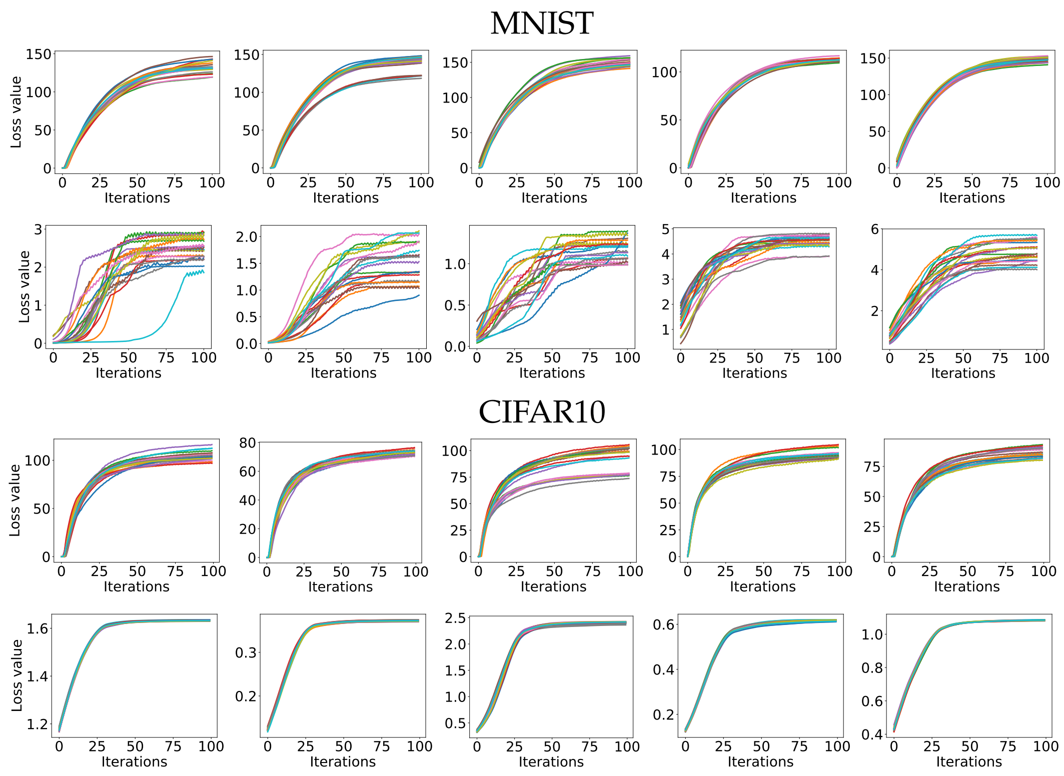


Figure 1.11: Loss function value over PGD iterations for 20 random restarts on random examples. The 1st and 3rd rows correspond to standard networks, while the 2nd and 4th to adversarially trained ones.

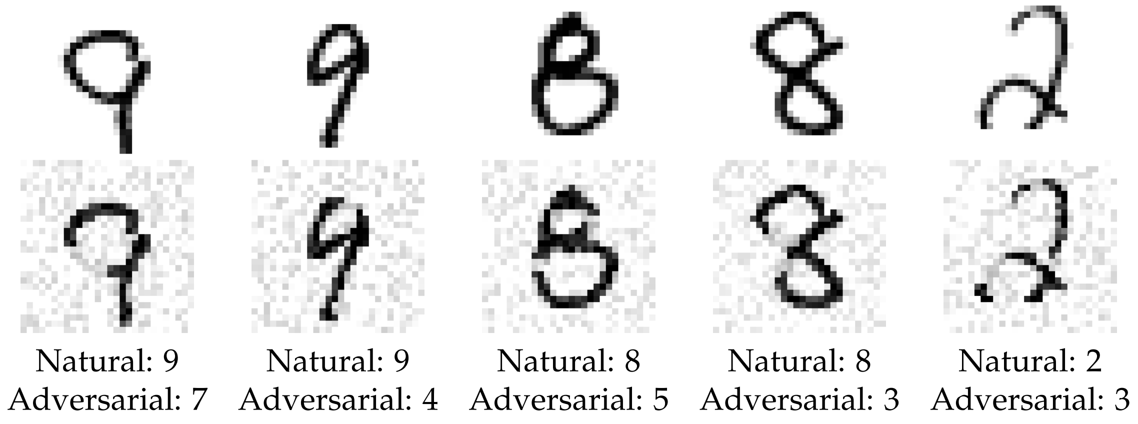


Figure 1.12: Sample adversarial examples with ℓ_2 norm bounded by 4. The perturbations are significant enough to cause misclassification by humans too.

Chapter 2

Defenses Against Backdoor Attacks

Backdoor attacks are a common and pernicious threat to machine learning in which an adversary injects a few maliciously constructed input-label pairs into the training set with the goal of manipulating the model predictions at test time. Multiple methods were proposed to defend against backdoor attacks, however, they are fairly specialized to particular interventions. In this chapter, we present a new perspective on data poisoning. While previous defenses treat the problem as an outlier detection task, we frame the problem as one of detecting features in the data to which models are greatly sensitive. Based on this observation, we develop a new framework for detecting backdoor attacks. Our framework naturally gives rise to an algorithm for detecting backdoor attacks—we demonstrate the effectiveness of the algorithm both theoretically and empirically.

2.1 Introduction

A *backdoor attack* is a technique that allows an adversary to manipulate the predictions of a supervised machine learning model [GDGG17, CLL+17, ABC+18, SHN+18, TTM19]. To mount a backdoor attack, an adversary modifies a small subset of the training inputs in a systematic way, e.g., by adding a fixed "trigger" pattern; the adversary then modifies all the corresponding targets in a systematic way, e.g., by setting them all to some fixed value y_b. This intervention allows the adversary to manipulate the resulting models' predictions at test time, e.g., by inserting the trigger into test inputs.

Given the threat posed by backdoor attacks, there is an increasing interest in defending ML models against them. One such line of work aims to detect and remove the manipulated samples from the training set [JSR21, TLM18, HKSO21b, CCB+18]. Another line of work seeks to directly train ML models that are robust against backdoor attacks (without necessarily removing any training samples [LF21, JCG21]).

A prevailing perspective on defending against backdoor attacks treats the manipulated samples as *outliers*, and thus draws a parallel between backdoor attacks and the classical data poisoning setting of robust statistics. In the latter setting, one receives data that is from a known distribution $\mathcal{D}$ with probability $1-\varepsilon$, and adversarially chosen with probability ε—the goal is to detect (or learn in spite of) the adversarially chosen points. This perspective is natural one to take and has lead to a host of defenses against backdoor attacks, but *is it the right way to approach the problem?*

In this work, we take a step back from the above intuition and offer a new perspective on data poisoning: rather than viewing the manipulated images as outliers, we propose to view the trigger pattern itself as just another feature in the data. Specifically, we demonstrate that backdoors inserted in a dataset can be indistinguishable from features already present in that dataset. On one hand, this immediately pinpoints the difficulty of detecting backdoor attacks, especially when they can correspond to *arbitrary* patterns. On the other hand, this new perspective suggests there might be an equivalence between detecting backdoor attacks and surfacing features in the data.

Equipped with this perspective, we introduce a framework for studying features in input data and characterizing how sensitive a model is to examples with this feature. Within this framework, we can view backdoor attacks simply as particularly strong features. Furthermore, the framework naturally gives rise to an algorithm for detecting—using datamodeling [IPE+22]—the strongest features in a given dataset. The presence/absence of these features drives the performance of the model whenever present in a dataset, and can thus be leveraged to mount backdoor attacks. We provide theoretical justification for our framework. In addition, we demonstrate through a range of experiments the effectiveness of our framework in detecting backdoored samples for a variety of standard backdoor attacks. In the remainder of the chapter, we proceed as follows:

- In Section 2.2, we show through two illustrative examples that in the absence of an explicit probability distribution for natural image data, backdoor attacks are in a natural sense *indistinguishable* from naturally-occuring features in the data. We thus argue that detecting backdoors equates to identifying particular features in the training data.

- In Section 2.4, we formally define a notion of model sensitivity to a feature in the data, and we prove a formal connection to this quantity and a datamodel-related [IPE$^+$22] quantity, allowing us to efficiently approximate sensitivity by leveraging the datamodeling framework.

- In Section 2.5, we use these insights to theoretically motivate a natural algorithm to detect backdoor attacks in the space of datamodel weights for a dataset.

- In Section 2.6, we use our proposed algorithm to identify standard backdoor attacks. We find that our algorithm is effective in nullifying the backdoor attacks in a range of experiments. Furthermore, the algorithm matches previous state-of-the-art algorithms [JSR21] in many experiments, and provides superior performance in one of our experiments.

2.2 A Feature-Based Perspective on Backdoor Attacks

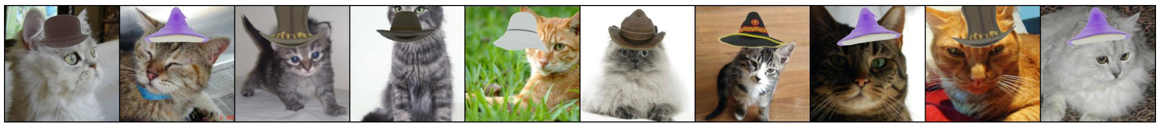
The prevailing perspective on backdoor attacks casts them as an instance of *data poisoning*, a concept with a rich history in robust statistics [HRRS11]. In data poisoning, the goal is to learn from a dataset where most of the points (say, a $1 - \varepsilon$ fraction) are drawn from a distribution $\mathcal{D}$, and the remaining points (an ε-fraction) are chosen by an adversary. The parallel between this "classical" data poisoning setting and that of backdoor attacks is natural. After all, in a backdoor attack we are given a dataset that is primarily drawn from a data distribution $\mathcal{D}$, but partially adversarially chosen in that an adversary has added the trigger pattern to a small fraction of samples.

This threat model is tightly connected to the classical poisoning setting in robust statistics. In the classical settings, the *structure* of the dataset $\mathcal{D}$ is essential to obtaining any theoretical guarantees. For example, algorithms often leverage strong explicit distributional assumptions, e.g. (sub-)Gaussianity [LM19].

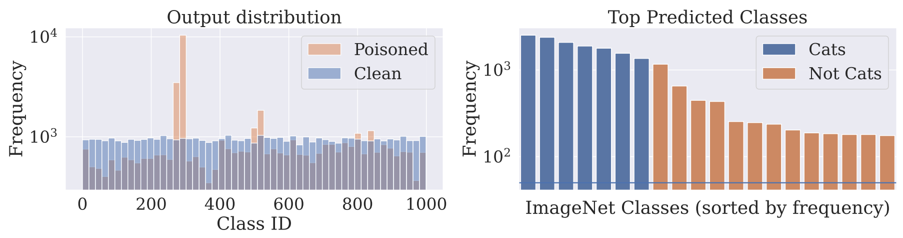
In settings such as computer vision (which we typically use to study backdoor attacks), no such structure exists. In fact, we lack almost any characterization of how benchmark image datasets are distributed. As a result, we argue, backdoors inserted in a dataset are fundamentally indistinguishable from features already present in the dataset.

We illustrate this in two ways. First, we show that one can mount a backdoor attack using a weak feature that is already present in the dataset. In particular, in Figure 2.1, we mount a backdoor attack on ImageNet [DDS+09] by using *hats* in place of a fixed trigger pattern. In particular, we use 3D rendering [LSI+21] to add hats of varying shape, size, and color to a fraction of the training images from the "cat" superclass of ImageNet. The resulting dataset is entirely plausible in that the images are (at least somewhat) realistic, and the corresponding labels are unchanged—with some more careful photo editing, one could imagine embedding the hats in a way that makes the dataset look unmodified even to a human. At test time, however, the hats act as an effective backdoor trigger: model predictions are skewed towards cats whenever a hat is added on the test sample.

In fact, the adversary need not modify the dataset at all—one can use features already present in the dataset to manipulate models at test time. For example, Figure 2.2 shows that since "tennis ball" is a class in the ImageNet training set, a



(a) Sample images of dogs with generated hats



(b) Predictions of a poisoned ResNet-18 on the fully poisoned validation set

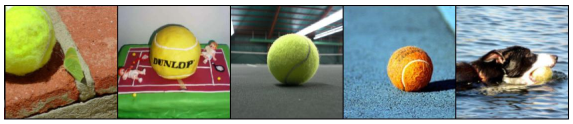
Validation Set	
Clean Accuracy	*Poisoned Accuracy*
63.72%	42.04%

(c) Model accuracy

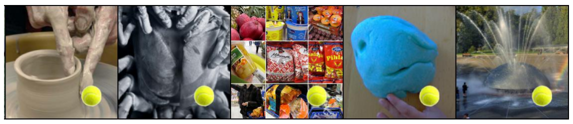
Figure 2.1: An adversary can craft a trigger that is indistinguishable from a natural feature and use it as a backdoor. (a) We "backdoor" the ImageNet training set by generating (using 3DB [LSI+21]) images of hats and pasting them on 20% of the cats images of ImageNet [DDS+09]. We train a ResNet-18 [HZRS15a] on the backdoored training set, and evaluate it on both the *clean* validation set, and on a validation set with the trigger added to each image. (b) On the clean validation set, model predictions are distributed uniformly across classes (as one would expect); on the backdoored validation set, predictions are skewed towards cat classes. (c) Furthermore, the accuracy of the model drops from 63.72% on the clean validation set to 42.04% on the poisoned validation set.

small photo of a tennis ball is a remarkably effective backdoor trigger at test time, despite the training set being "unpoisoned."

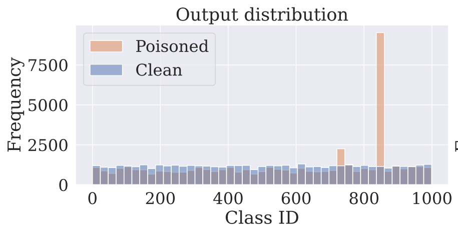
Both of these examples highlight that without making additional assumptions, trigger patterns for backdoor attacks are no more than features in the data, and so detecting them should be no easier than detecting hats or tennis balls. In the next sections, we'll take use this insight to craft more specific conditions under which we can hope to detect backdoor attacks. But first, we will present *datamodeling* [IPE+22], a recent framework that quantifies the interaction between different data points. Datamodeling will be useful in detecting particular trends exhibited by backdoored samples.

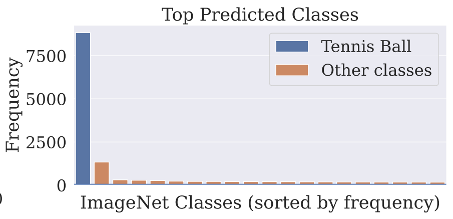


(a) ImageNet images from the class "tennis ball".



(b) ImageNet images with the "tennis balls" trigger.





Validation Set	
Clean Accuracy	*Poisoned Accuracy*
63.88%	50.88%

(d) Model accuracy

(c) Predictions of a ResNet-18 on the fully poisoned validation set

Figure 2.2: An adversary can even leverage the features of a dataset to mount a backdoor attack. (a) The tennis ball feature is already present in the ImageNet training set, so we do *not* modify the dataset at all. (b) Instead, we show that a small picture of a tennis ball is a "pre-existing" backdoor trigger. Plots (c) and (d) are the same as Figures 2.1 (b) and (c).

2.3 Inspecting Data Using Datamodels

In order to quantify the interaction between data points in a given dataset, we try to understand how the composition of this dataset affects the behavior of models trained on (subsets of) the dataset. Multiple approaches in the literature try to estimate this relation by studying the counterfactual effect of excluding particular data points from a dataset and estimating the predictions of a model trained on the subset of the dataset. Such approaches include influence functions [CW82, KL17, FZ20, LZL+22], data shapely values [KZ21, KDI+22, GZ19], and datamodels [IPE+22].

While each approach presents a different perspective on the interaction between the data points, we adopt the datamodeling framework presented in [IPE+22]. By leveraging this framework, we can represent a model's prediction for a given data point as a linear combination of the training data points. We show in section 2.4 how we leverage this linear relation to detect samples that have a strong effect on a model's behavior.

Computing datamodels. Given a dataset, a datamodel for a data point x measures how the prediction of a model f for the data point x changes by including/excluding data point x' from the training set of the model f. This effect can be quantified by solving a regression problem. Specifically, we train a large number of models on different subsets of the dataset. Each subset contains a fraction α of the dataset. Then for each model f_i, we collect the following quantities: 1) a binary mask $\mathbb{1}_{\mathcal{S}_i}$ that indicates which data points were used to train the model f_i, and 2) the prediction $f(x; \mathcal{S}_i)$ of the model f_i for the data point x. Given these two quantities, we form the following auxiliary dataset $\{(\mathbb{1}_{\mathcal{S}_i}, f(x; \mathcal{S}_i))\}_i$, then we fit a linear model g_w that approximates $f(x; \mathcal{S}) \approx w_x^\top \mathbb{1}_{\mathcal{S}}$, the model's prediction for data point x when trained on a random subset $\mathcal{S}$ of the dataset. The whole procedure is outlined in Algorithm 1. Given this approximation, we quantify the effect of a data point x_j on a data point x by $w_x[j]$.

Algorithm 1 Computing datamodels for a given dataset.

```
Require: Dataset $\mathcal{S} = \{x_n\}_{n=1}^N$ with $N$ samples, training algorithm $\mathcal{A}$, subset ratio
    $\alpha$, number of models $m$
1: Sample $m$ random subsets $S_1, S_2, \cdots, S_m \subset \mathcal{S}$ of size $\alpha \cdot |\mathcal{S}|$:
2: for $i \in 1$ to $m$ do
3:     Train model $f_i$ by running algorithm $\mathcal{A}$ on $S_i$
4: end for
5: for $n \in 1$ to $N$ do
6:     Collect datamodels training set $\mathcal{D}_n = \{(\mathbb{1}_{\mathcal{S}_i}, f_i(x_n, \mathcal{S}_i))\}_{i=1}^m$
7:     Compute $w_{x_n}$ by fitting Ridge on $\mathcal{D}_n$
8: end for
9: return $w_{x_n}$ $\forall$ $n \in [N]$
```

2.4 When Can We Detect Poisoned Samples?

Throughout this section, we assume a fixed dataset D. We define a *feature* to be a function $f : \mathcal{X} \to \mathbb{R}$ from the space of inputs $\mathcal{X}$ to the real numbers. More concretely, we work in a simplified binary model where f takes values in $\{-1, 1\}$. Intuitively, $f(x) = 1$ indicates the presence of the feature f, while $f(x) = -1$

indicates its absence. We define the *support* S_f of a feature f over the dataset D as the subset of examples of the dataset D where the feature f is present, *i.e.* $S_f = \{x \in D \big| f(x) = 1\}$.

Motivated by the examples presented in section 2.2, backdoor triggers should be treated as features. However, without additional assumptions, there would be no way to distinguish them from the features naturally occurring in the dataset. To overcome this, we make the natural assumption that the backdoor trigger is a feature that has a *strong effect* on the model's prediction for a backdoored example. Indeed, successful backdoor attacks lead to significant drops in model confidence after insertion of the trigger [GDGG17, CLL+17, ABC+18, SHN+18, TTM19]. In the next section, we make this intuition precise.

2.4.1 Towards a Definition of Model Sensitivity

Successful backdoor attacks rely on triggers that are effective in flipping the predictions of a model trained on a poisoned dataset. This hints that the model is highly *sensitive* to the feature associated with the trigger. However, how can we quantify the sensitivity to a feature? In this section, we seek to answer this question by developing an idealized, formal definition of a sensitivity quantity. In the next section, we show how to feasibly estimate this quantity.

As a point of start, it is natural to consider a model sensitive to a feature f if adding more examples containing the feature to the *training* set has a relatively large effect on the model predictions for *test* examples containing this feature, compared to test examples that do not. By further reasoning through this intuition, we arrive at the following additional assumptions:

- any measure of the feature f's strength must be specific to a given test example x. Indeed, the effect of the same feature f will be different for different test examples, as we will see later on in Figure 2.3.
- empirically estimating our intuitive measure of the feature f's sensitivity will depend not only on the *number* if examples of f in the training data, but also

on *which* particular examples are chosen. To avoid this source of variability and come up with a notion that only depends on the number of examples containing the feature f, we aim for a measure that takes the expectation over the choice of examples containing the feature f.

In line with the above requirements, we start by defining the *total* effect $m_f(x,t)$ of t examples with feature f on the test examples x to be the expected margin of the model on an example x given that there are t training examples containing the feature f. Concretely,

$$m_f(x,t) = \mathbb{E}_S \left[M(x;S) \middle| \left| S \cap S_f \right| = t \right] \tag{2.4.1}$$

where $M(x;S)$ is the margin of a model on the test example x, when the model is trained on a subset S of the training set D. Here, the subset S is sampled from some distribution over subsets to be specified later. Intuitively, $m_f(x,t)$ measures the average effect of having t examples with the feature f on a model's predictions for the example x.

Having defined the total effect of training with t examples of a feature, we can then define the *marginal sensitivity* to formally capture our intuition about the sensitivity of f:

$$s_f(x,a,b) = m_f(x,b) - m_f(x,a) \tag{2.4.2}$$

where $0 \leq a < b \leq \left|S_f\right|$. Intuitively, the marginal sensitivity $s_f(x,a,b)$ correlates with the sensitivity of a model to a feature f. In particular, a large marginal sensitivity indicates that including b examples with the feature f as opposed to a has a strong effect on the margin of a model on examples x, while a small marginal sensitivity hints that including in the training set more examples with the feature f has little effect on the margins of a model on examples x.

In the context of data poisoning, and in the presence of an effective backdoor trigger, the test examples containing the trigger should exhibit a large marginal sensitivity to the trigger, while the examples without the trigger should exhibit a low marginal sensitivity to the trigger. Indeed, we plot in Figure 2.3 the values of the

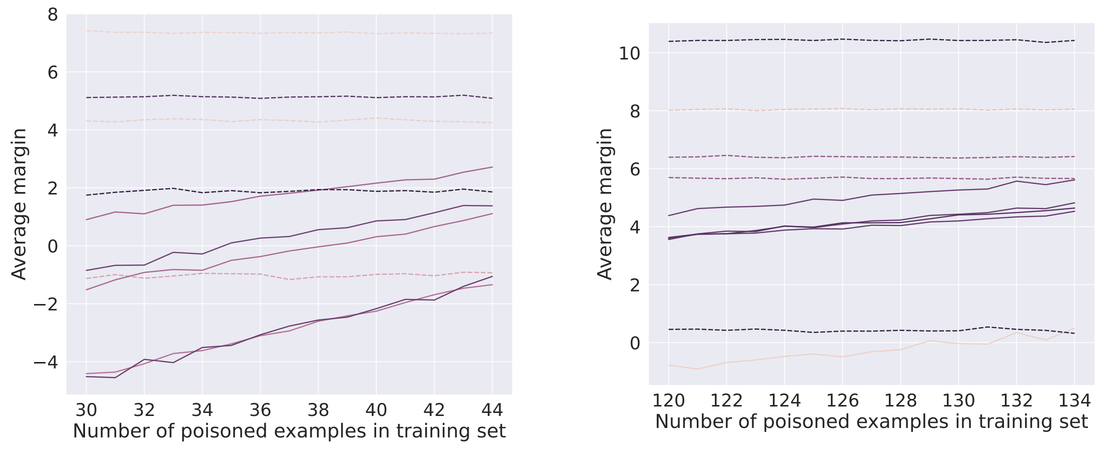


Figure 2.3: Empirical estimates of the total effect $m_{f_{trigger}}(x, t)$ as a function of t for 5 poisoned (solid lines) and 5 clean (dashed lines) examples in two of our poisoning scenarios (left and right).

total effect $m_{f_{trigger}}(x, \cdot)$ as a function of the number of poisoned examples observed when training 100,000 models on different subsets of a poisoned dataset[1]. We observe that for a poisoned examples $x_{trigger}$, the total effect $m_{f_{trigger}}(x_{trigger}, \cdot)$ increases when adding more poisoned examples to the training set (*i.e.* the marginal sensitivity is positive), while for a clean examples x_{clean}, the total effect $m_{f_{trigger}}(x_{clean}, \cdot)$ barely changes when adding more poisoned examples, *i.e.* the marginal sensitivity is almost 0.

In the next section 2.4.2, we describe a method to efficiently estimate the marginal sensitivity $s_f(x, a, b)$ of a examples x to a feature f in a certain regime of the parameters. Based on this, we then develop an algorithm in section 2.5 that detects features that have a strong effect on a model's prediction. We posit that these features – natural and synthetic – could be leveraged as backdoor triggers. Our framework gives rise naturally to a method that limits a model's overall sensitivity to arbitrary trigger-like features. We demonstrate in section 2.6 the effectiveness of our method in countering numerous backdoor attacks.

[1]We defer the implementation details to X

2.4.2 Approximating Sensitivity using Datamodels

At first sight, computing the marginal sensitivity $s_f(x, a, b)$ seems hard. Particularly, the computation of the marginal effect $m_f(x, \cdot)$ as outlined in equation (2.4.1) entails training an exponentially large number of models from scratch, then taking the expectation over the different models.

It turns out, however, that this quantity could be approximated efficiently using the datamodeling framework [IPE+22] presented in section 2.3. Specifically, we can approximate the margin $M(x; S)$ without training a model on the subset S.

Concretely, let x be a test example, D be a training dataset, and recall the subset size ratio $\alpha \in (0, 1)$. Then, there exist for example x datamodel weights $\mathbf{w}_x^\alpha$ and bias term $\mathbf{b}_x^\alpha$ such that

$$\mathbb{E}_S \left[\left(M(x; S) - (\mathbb{1}_S^\top \mathbf{w}_x^\alpha + \mathbf{b}_x^\alpha) \right)^2 \right] \leq \varepsilon \tag{2.4.3}$$

where S is a subset of D of size $m = \lfloor \alpha \cdot |D| \rfloor$ examples uniformly at random [IPE+22]. Alternatively, we can write $M(x; S) \approx \mathbb{1}_S^\top \mathbf{w}_x^\alpha + \mathbf{b}_x^\alpha$. Using this result, we can prove the following lemma (where we suppress the dependence on α from the notation).

Lemma 2.4.1. *For a subset $S \subset D$, let $\mathbf{h}_S = \frac{1}{|S|} \mathbb{1}_{S_f} - \frac{1}{n-|S|} \mathbb{1}_{S_f^c}$ and $t_f = \lfloor \alpha \cdot |S_f| \rfloor$ be the expected number of examples having the feature f for a given model used to compute the datamodel weights. Then, for all $x \in D$, we have*

$$d_f(x) := s_f(x, t_f, t_f + 1) \approx \mathbf{w}_x^\top \mathbf{h}_{S_f} \tag{2.4.4}$$

The proof can be found in section 2.9. In the above lemma, $d_f(x)$ represents the discrete derivative $m_f(x, \cdot)$ evaluated at t_f, the expected number of examples with feature f observed when training models for the estimation of datamodels. A large value of $d_f(x)$ indicates that adding more examples with the feature f has a strong effect on the model's prediction, while a small value indicates that adding more examples with that feature has little effect on the model's predictions. For backdoor attacks, we expect the derivative for the trigger $d_{f_{trigger}}(\cdot)$ to be large for examples

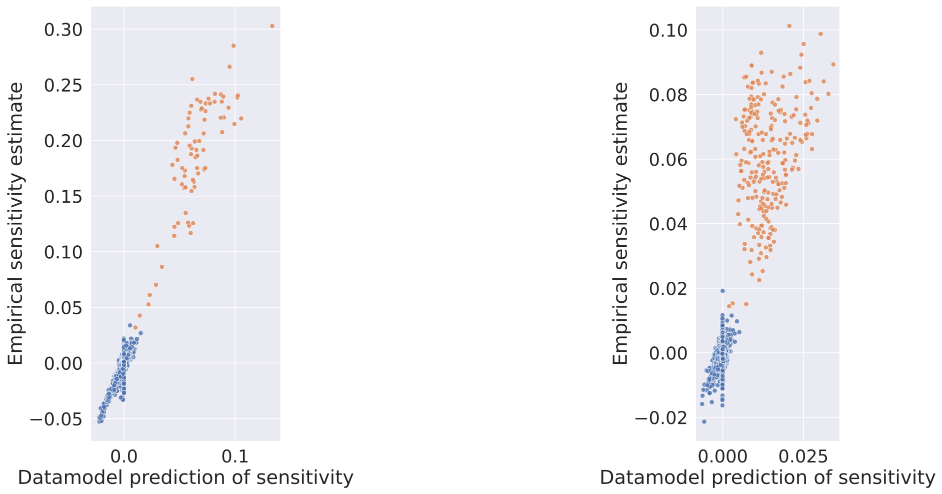


Figure 2.4: Comparing the sensitivity predicted by datamodel weights estimated by training 100,000 models (x axis) with the empirical estimates of sensitivity from the same 100,000 models for two of our poisoning scenarios (left and right).

with the trigger, and small for examples without the trigger (clean examples).

To verify the quality of our approximation with datamodels, we plot in figure 2.4 for all the examples in the dataset the empirical values of $d_f(\cdot)$ and the approximation using the datamodel weights (modulo a constant factor). Overall, we see a good correlation between the true and predicted values.

So far, we have seen how a highly poisonous feature f for an example x implies something about the datamodel vector $\mathbf{w}_x$ of x. However, what about the opposite direction? That is, what does $\mathbf{w}_x$ imply about the poisonous features for an example x?

2.4.3 The Highest-Sensitivity Feature for an Example

To answer the above question, we establish the following result, showing that the strength of the most poisonous feature for x is tightly controlled by the ℓ_∞ norm of $\mathbf{w}$ when it is shifted to have mean zero:

Lemma 2.4.2. *Let $\mathbf{w}_x$ be the datamodel weights for an example x. Consider the subset*

$S_x \subset D$ which maximizes $\mathbf{w}_x^\top \mathbf{h}_S$, i.e.

$$S_x = \mathrm{argmax}_S\, \mathbf{w}_x^\top \mathbf{h}_S.$$

Also, let $\overline{\mathbf{w}}_x = \mathbf{w}_x - \frac{1}{|D|}\sum_x \mathbf{w}_x$ be the centered datamodel weights for example x. Then, we have

$$\frac{n}{n-1}\left\|\overline{\mathbf{w}}_x\right\|_\infty \leq \mathbf{w}_x^\top \mathbf{h}_{S_x} \leq 2\left\|\overline{\mathbf{w}}_x\right\|_\infty$$

The proof can be found in section 2.9. In the next section, we will leverage these theoretical results as the core of a practical algorithm for detecting trigger-like features in the data.

2.5 Detecting Poisoned Samples Using Datamodels

How can we turn the theoretical connections between poisonous features and datamodels into a practical algorithm to detect backdoor triggers?

2.5.1 Motivation

As outlined above, we start from a natural set of assumptions about the backdoor feature $f_{backdoor}$:

- $f_{backdoor}$ is significantly more poisonous for examples x with the trigger than the other features in the data;
- $f_{backdoor}$ is not significantly poisonous for examples x without the trigger.

Given the results from the previous section, these assumptions have implications about the datamodel weights:

- $\mathbf{w}_x^\top \mathbf{h}_{S_f} \gg \mathbf{w}_z^\top \mathbf{h}_{S_f}$ for an example x with the trigger and an example z without the trigger. In particular, this implies that the poisoned examples are separated by a hyperplane of large margin from the clean examples, and suggests that the vectors $\mathbf{w}_x$ for poisoned x are similar to each other (have high dot product) since all of them are well-aligned with the same vector $\mathbf{h}_{S_f}$.

- $\|\mathbf{w}_x\|_\infty \gg \|\mathbf{w}_z\|_\infty$ for an example x with the trigger and an example z without the trigger.

Taken together, these statements suggest that the examples with the trigger should form a well-separated cluster on the periphery of the dataset in the space of datamodel vectors $\{\mathbf{w}_x\}_{x\in D}$.

2.5.2 Algorithm

Given the reasoning in the previous section, we propose the following algorithm:

- shift the datamodel vectors $\mathbf{w}_x$ to their mean-zero counterparts $\overline{\mathbf{w}}_x$ for all the examples $x \in D$;
- run a standard dimensionality reduction method (we chose PCA) on the vectors $\{\overline{\mathbf{w}}_x\}_{x\in D}$, followed by a standard clustering algorithm (we chose k-means);
- for each example x, calculate the sum of distances from the (dimensionality-reduced) vector corresponding to x to the cluster centers, and flag for removal examples for which this sum is high. Since this score favors points in clusters on the periphery of the data, we expect to be able to detect the poisoned examples with this approach.

2.6 Experiments and Results

We evaluate our framework against two common types of backdoor attacks: dirty-label attacks [GDGG17] and clean-label attacks [TTM19]. Furthermore, we consider multiple triggers and poison different proportions of our dataset. For every attack, we measure the effectiveness of the poison by comparing the accuracy on the clean

Exp.	Attack Type	Trigger	Target Class	Proportion	Clean Acc.	Poisoned Acc.
1	Dirty Label	1-pixel	automobile	1.5%	86.64	19.90
2	Dirty-Label	1-pixel	automobile	5%	86.67	12.92
3	Dirty-Label	m-way	automobile	1.5%	86.39	49.57
4	Dirty-Label	m-way	automobile	5%	86.23	10.67
5	Clean-Label	3x3	automobile	1.5%	86.89	75.58
6	Clean-Label	3x3	automobile	5%	87.11	41.89
7	Clean-Label (no adv.)	3x3	cat	5%	86.94	71.68
8	Clean-Label (no adv.)	3x3	cat	10%	87.02	52.08

Table 2.1: To evaluate the effectiveness of our framework, we run multiple experiments and vary the attack type, the trigger, the target class, and the proportion of poisoned samples. We report the clean accuracy and the poisoned accuracy of every attack when no defense is employed. A larger gap between clean and poisoned accuracy means the attack is more effective.

and backdoored validation sets. Table 2.1 shows a summary of the parameters of the experiments, along with the effectiveness of the attacks.

Baselines. We compare our algorithm with multiple baselines: Inverse Self-Paced Learning[2] (ISPL) [JSR21], Spectral Signatures[3] (SS) [TLM18], SPECTRE[4] [HKSO21b] and Activation Clustering[5] (AC) [CCB+18]. To evaluate the algorithms, we use the CIFAR-10 dataset [Kri09] and the ResNet-9 model [HZRS15a][6].

Computing Datamodels. We adopt the framework presented in [IPE+22] to estimate the datamodels of each experiment. Specifically, we train 100k models on different subsets containing 50% of the training set chosen at random. Furthermore, we use the FFCV library for efficient data-loading [LIE+22]. One augmentation was used for dirty-label attacks (Cutout [DT17]) to improve the performance of the model on CIFAR10. Similar to [TTM19], we do not use any data augmentation when performing clean-label attacks.

[2]We thank the authors for sharing their implementation with us
[3]We re-implement the algorithm presented in the paper
[4]`https://github.com/SewoongLab/spectre-defense`
[5]We use the implementation provided in: `https://github.com/SewoongLab/spectre-defense`
[6]https://github.com/wbaek/torchskeleton/blob/master/bin/dawnbench/cifar10.py

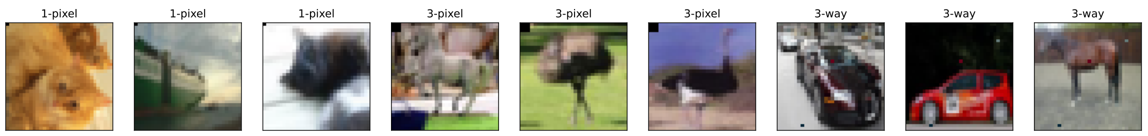


Figure 2.5: We execute the poisoning attacks with three types of triggers: (a) one black pixel on top left corner (first three images), (b)3x3 black square on top left corner (middle three images), and (c) 3-way triggers adapted from [XHCL20].

Trigger. We conduct our experiments with two types of triggers. The first type is a fixed pattern inserted in the top left corner of the image. The trigger is unchanged between train and test time. This type of trigger has been used in multiple works [GDGG17, TTM19]. The other type of trigger is an m-way trigger, with m=3 [XHCL20]. During training, one of three triggers is chosen at random for each image to be poisoned, and then the trigger is inserted into one of three locations in the image. At test time, all three triggers are inserted at the corresponding positions to reinforce the signal. We display in Figure 2.5 the triggers used to poison the dataset.

2.6.1 Poisoning Attacks

Dirty-Label Backdoor Attacks

The most prominent type of backdoor attacks is a dirty-label attack [GDGG17]. During a dirty-label attack, the adversary inserts a trigger into a subset of the training set, then sets the label of the poisoned samples to a particular target class y_b. Looking at Table 2.2 (experiments 1 through 4), we can see that our proposed defense is successful and results in a consistently high accuracy on the full poisoned validation set.

Clean-Label Backdoor Attacks

A more challenging attack is the clean-label attack [SHN$^+$18, TTM19][7] where the adversary avoids changing the label of the poisoned samples. To mount a successful

[7]We evaluate the clean-label attack presented in [TTM19]

clean-label attack, the adversary only poisons samples from the target class with the goal of creating a strong correlation between the target class and the trigger.

We perform two clean-label attacks. During the first attack, we perturb the image with an adversarial example before inserting the trigger in order to weaken the features already present in the images, and hence allowing the trigger to dominate other features [TTM19]. During the second attack, we avoid adding the adversarial example, however, we poison more samples to have an effective attack.

We show the results of the clean-label attacks in Table 2.1. We can see that the defense is successful, and the poisoned accuracy is very close to the clean accuracy.

2.6.2 Efficiency of the Proposed Defense Mechanism

We next analyze the efficiency of our proposed defense mechanism. Specifically, our method returns a numerical score for each training sample, and the poisoned samples exhibit a higher score than the clean samples. We assume that the portion of poisoned samples is small, and flag as “suspicious” the top 10% samples with the highest scores.

In Figure 2.6, we plot the actual proportion of removed poisoned samples when removing the samples with the highest scores. As we can see, by removing 10% of the samples with the highest score, we remove a large portion of the poisoned samples in most of our experiments. Even though this is not the case for experiments 3 and 5, the attack is ineffective after removing 10% of the samples. We think this could be happening since (1) these experiments contain fewer poisoned samples, (2) and removing the strongest of the poisoned samples is enough to nullify the attack. In all of the experiments, the remaining poisoned samples are not sufficient to mount an effective backdoor attack, as can be seen in Table 2.2. Specifically, after training a model on the curated dataset, which still contains some poisoned samples, the accuracy of the model doesn’t drop after inserting the trigger into test images.

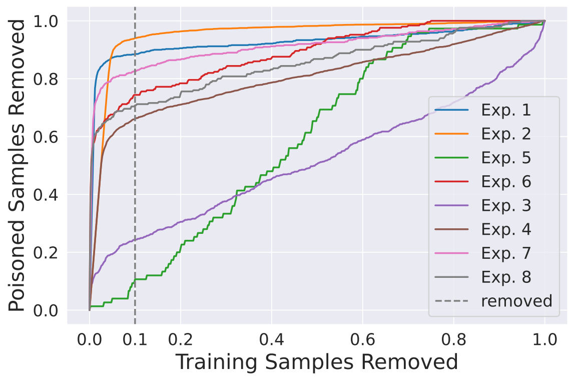


Figure 2.6: We plot how the portion of poisoned samples removed changes when removing different portions of the training set. For all our experiments, we assume a small portion of the dataset is poisoned, and we remove the top 10% samples having the highest score returned by our method. By removing 10% of the suspicious samples, we effectively remove a large portion of the poisoned samples, and the attacks become ineffective, as can be seen in Table 2.2.

2.6.3 Principal Components of the Datamodels Matrix

Our theoretical foundation from section 2.4 indicates that the norm of datamodels' weights for poisoned samples is larger than for clean samples. As such, it is possible to cluster the samples to separate poisoned samples from clean samples.

In this section, we inspect visually this property. Similar to [IPE+22], we compute the top 100 principal components of our datamodels matrices, and visualize the top and bottom images along these components. We can see in Figure 2.7 that many of the top images along these components are poisoned.

In addition, for each of our experiments, we project the corresponding datamodels matrices using PCA to a two-dimensional space, and plot the distribution of clean and poisoned samples in this space. We notice in Figure 2.8 that the clean samples are heavily concentrated near the centroid of the data, while the poisoned samples are further away. This suggests that the stronger features are further away from the centroid than the weaker features.

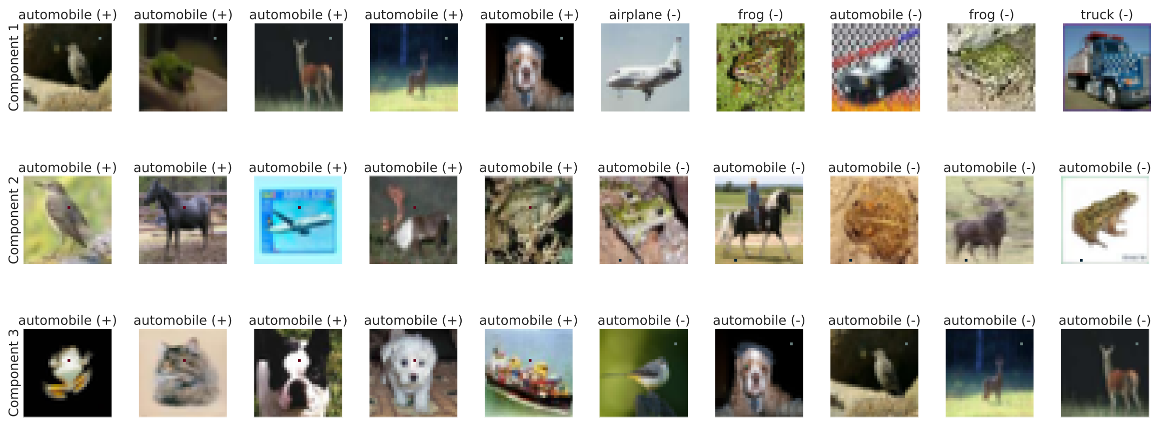


Figure 2.7: We compute the datamodels corresponding to Exp. 4 (m-way trigger, c.f. Table 2.1), and then compute the top 100 PCA components. For each component, we show above the images that have the highest projection on the component. In accordance with our theory, we observe that these images contain many poisoned samples. Interestingly, the train-time m-way trigger–which could be one of three options–is consistent across each direction of the PCA component. Specifically, all the images along the positive direction of the first PCA component share the top-right green trigger, while the ones corresponding to the second component share the centered red trigger, etc.

2.7 Related Work

2.7.1 Related Threat Models for Machine Learning

This work focuses on backdoor data poisoning attacks; here we briefly contrast them with other popular attacks on ML models. A closely related, but different, threat model is that of *targeted data poisoning attacks*, where an adversary wants a pre-defined test example to be misclassified and is only allowed to modify training examples. In this model, the attacker does not need access during inference time [KL17, SHN+18]. In contrast, backdoor attacks can be applied to any image at inference time by inserting the backdoor trigger. Another closely related threat model is that of *availability attacks* [MGBD+17, LKY22]. Here the goal of the attacker is to bring down the test performance as much as possible by perturbing a small part of the training set. In contrast, backdoor attacks are designed to only affect images once the attacker's trigger is inserted.

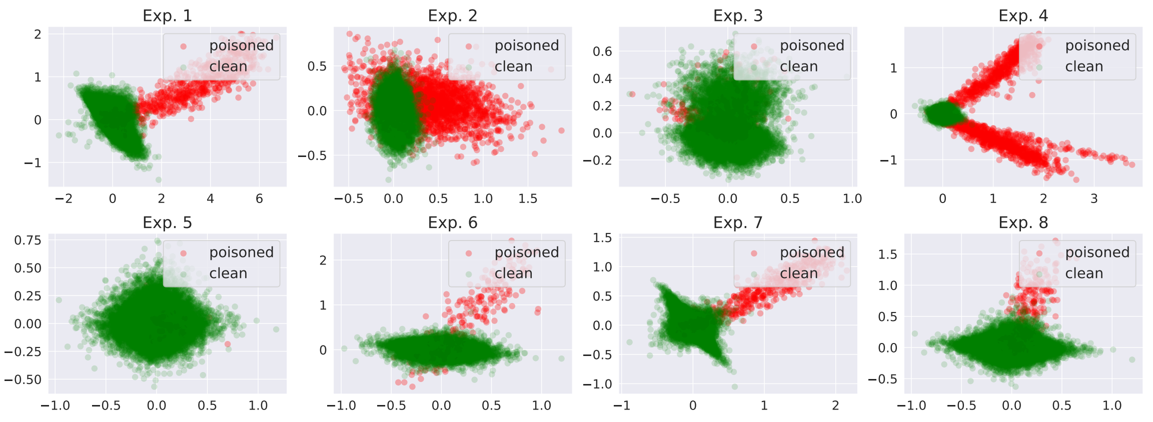


Figure 2.8: For each of our experiments, we compute the corresponding datamodels, and then project them to a 2D space using PCA. We plot this projection above, and observe that the poisoned samples are separable from the clean samples in this 2D space.

Adversarial examples are another well-studied threat model [SZS+14]. There, the adversary seeks to make the model misclassify a given example by introducing imperceptible changes to the images at inference time. Unlike in backdoor attacks, the adversary does not have the power to corrupt training examples. As a result, crafting adversarial examples is more complex than inserting backdoor triggers at inference time, and often requires either white-box access to the model or a significant number of model queries.

Finally, recent work has explored the inevitability of a certain kind of backdoor attacks [GKVZ22]. However, in the setup of this work, the assumption is that the training process is delegated to the adversary, and the client then has a limited computational budget to verify that no backdoor exists. This threat model is much more powerful for the adversary compared to the backdoor triggers considered in the current work, and correspondingly the undetectability results do not carry over to our setting.

2.7.2 Overview of Backdoor Attacks and Defenses

Developing backdoor attacks and defenses in the context of deep learning is an increasingly active area of research. The reader is directed to [GTX+22, LJLX22] for a survey of the field. Here, we briefly overview the most relevant attacks and defenses. [GDGG17] mislabel images where the trigger is inserted in order to make the attack realizable with a small budget of train example corruptions. This, however, can be easily detected by training a backdoor classifier on a small sample of human-annotated examples [TTM19].

This motivates clean-label attacks, which create an effective backdoor attack without inserting easy to detect mislabeled images [TTM19, SSP20]. On the defense side, a popular line of work applies outlier detection in the latent space of neural networks [TLM18, CCB$^+$18, HKSO21a]; another direction targets the fact that triggers have small norm to detect them [WYS$^+$19]. Both approaches have been shown to be successful in dirty-label and clean-label settings. However, if the adversary knows what defense is going to be deployed, they can create adaptive attacks that bypass all the above defenses [S$^+$20, QXMM22].

Closest to our work is that of [JSR21]. The authors consider a defense based on the classifications the model makes, rather than properties of its latent space. This avoids latent-space-based defenses' vulnerability to adaptive attacks.

2.7.3 Techniques for Influence Estimation

Finally, a number of prior works explore the applicability of influence-based methods as defenses against different attacks in deep learning [KL17, FGL20]. To the best of our knowledge, only one concurrent work discusses using such methods for *backdoor* attacks [LZL$^+$22]. While their methodology is similar to ours, they only provide a proof of concept for detecting backdoors. In particular, the sparsity parameters they use for the defense depend on the attack, which is not practical since the attack is not known a priori to the defender.

2.8 Discussion and Conclusion

In this chapter, we proposed a new perspective of data poisoning. Essentially, we argued that backdoor triggers are fundamentally indistinguishable from features to which a model is highly sensitive, and consequently, detecting poisoned samples in a dataset is equivalent to detecting such features. Given this new perspective, we showed through two illustrative examples that one can leverage "natural" features to mount a backdoor attack.

Since the predictions of ML models can be greatly influenced by such features, an ML practitioner might want to remove the model's strong dependence on these strong features – regardless of whether they constitute a malicious backdoor attack or not – while retaining high accuracy. To this end, we presented a new theoretical framework to quantify model sensitivity to a feature. Furthermore, by leveraging the datamodeling framework, we derived theoretically a new algorithm that flags strong features in a given dataset. Through a wide range of backdoor poisoning setups, we demonstrated that by removing $\approx 10\%$ of a given dataset, our framework can lead to a model that is robust to variations of the strong features in a given dataset, all while retaining high accuracy.

2.9 Deferred proofs

2.9.1 Proof of Lemma 1.

Throughout this section, we let $p = |S_f|$ be the total number of examples with the feature in the training set D, and we write $w_x = w_x^\alpha, b_x = b_x^\alpha$, hiding the dependence on the fixed value of α for brevity. We assume throughout that the datamodel weights generalize well: namely, we assume that for every example $x \in D$ we have

$$\mathbb{E}_S\left[\left(M(x;S) - (\mathbb{1}_S^\top w_x + b_x)\right)^2\right] \leq \varepsilon$$

First, we will show that $m_f(x, t_f + 1) - m_f(x, t_f)$ can be well-approximated by the corresponding datamodel quantities:

Lemma 2.9.1. *Suppose $\alpha \le 1 - c$ for some constant c, e.g. $c = \frac{1}{10}$. Then, for all large enough values of n, m, p there exists a constant $C > 0$ such that for all $x \in D$ we have*

$$\left| m_f(x, t_f + 1) - \mathbb{E}_{S \sim \mathcal{D}_S}\left[\mathbf{w}_x^T \mathbb{1}_S \middle| |S_f \cap S| = t_f + 1\right]\right| \le \sqrt{C\varepsilon\sqrt{n}}$$

and

$$\left| m_f(x, t_f) - \mathbb{E}_{S \sim \mathcal{D}_S}\left[\mathbf{w}_x^T \mathbb{1}_S \middle| |S_f \cap S| = t_f\right]\right| \le \sqrt{C\varepsilon\sqrt{n}}$$

Proof. We have

$$\Pr_{S \sim \mathcal{D}_S}\left[|S_f \cap S| = t_f\right] = \frac{\binom{p}{t_f}\binom{n-p}{m-t_f}}{\binom{n}{m}}$$

and

$$\begin{aligned}\Pr_S\left[|S \cap S_f| = t_f + 1\right] &= \frac{\binom{p}{t_f+1}\binom{n-p}{m-t_f-1}}{\binom{n}{m}} \\ &= \frac{p - t_f}{t_f + 1}\frac{m - t_f}{n - p - m + t_f + 1}\Pr_S\left[|S \cap S_f| = t_f + 1\right]\end{aligned}$$

Our first goal is to show that both these probabilities are bounded from below by something reasonably large. First we will show that the ratio of the two probabilities is within a constant. We have (using that $m = \alpha n, t_f = \alpha p$)

$$\begin{aligned}\frac{p - t_f}{t_f + 1}\frac{m - t_f}{n - p - m + t_f + 1} &= \frac{1-\alpha}{\alpha + \frac{1}{p}}\frac{\alpha}{1 - \alpha + \frac{1}{n-p}} \\ &= \frac{\alpha}{\alpha + \frac{1}{p}}\frac{1-\alpha}{1 - \alpha + \frac{1}{n-p}} \\ &\ge \frac{\alpha}{\alpha + \alpha}\frac{1-\alpha}{2-\alpha} \\ &\ge \frac{1}{2}\frac{c}{1+c}\end{aligned}$$

where we used that $\frac{1}{p} \leq \frac{t_f}{p} = \alpha$ and $\alpha \leq 1 - c$.

So we have

$$\begin{aligned}
&\Pr_S \left[|S \cap S_f| = t_f + 1 \right] \\
&= \frac{\binom{p}{t_f}\binom{n-p}{m-t_f}}{\binom{n}{m}} \\
&\geq \frac{c}{2(1+c)} \Pr_S \left[|S \cap S_f| = t_f \right]
\end{aligned}$$

Now we will estimate the latter probability. Using Stirling's approximation, we have

$$\Pr_S \left[|S \cap S_f| = t_f \right] \asymp \frac{\sqrt{\frac{p}{2\pi t_f (p-t_f)}} \frac{p^p}{t_f^{t_f} (p-t_f)^{p-t_f}} \sqrt{\frac{n-p}{2\pi(m-t_f)(n-p-m+t_f)}} \frac{(n-p)^{n-p}}{(m-t_f)^{m-t_f} (n-p-m+t_f)^{n-p-m+t_f}}}{\sqrt{\frac{n}{2\pi m(n-m)}} \frac{n^n}{m^m (n-m)^{n-m}}}$$

$$= \sqrt{\frac{n}{p(n-p)} \frac{1}{\alpha(1-\alpha)}} \geq \frac{C}{\sqrt{n}}$$

for some constant C. Now from the triangle inequality, Jensen's inequality and Markov's inequality we have

$$\begin{aligned}
&\left| m_f(x, t_f) - \mathbb{E}_{S \sim \mathcal{D}_S} \left[\mathbf{w}_x^T \mathbb{1}_S + b_x \,\middle|\, |S \cap S_f| = t_f \right] \right| \\
&= \left| \mathbb{E}_{S \sim \mathcal{D}_S} \left[M(x; S) \,\middle|\, |S \cap S_f| = t_f \right] - \mathbb{E}_{S \sim \mathcal{D}_S} \left[\mathbf{w}_x^T \mathbb{1}_S + b_x \,\middle|\, |S \cap S_f| = t_f \right] \right| \\
&\leq \mathbb{E}_{S \sim \mathcal{D}_S} \left[\left| M(x; S) - (\mathbf{w}_x^T \mathbb{1}_S + b_x) \right| \,\middle|\, |S \cap S_f| = t_f \right] \\
&\leq \sqrt{\mathbb{E}_{S \sim \mathcal{D}_S} \left[(M(x; S) - (\mathbf{w}_x^T \mathbb{1}_S + b_x))^2 \,\middle|\, |S \cap S_f| = t_f \right]} \\
&\leq \sqrt{C\sqrt{n}\varepsilon}
\end{aligned}$$

and similarly for the case with $m_f(x, t_f + 1)$ (except with a somewhat worse constant). □

Lemma 2.9.2. *We have for every $x \in D$ that*

$$\mathbb{E}_{S\sim\mathcal{D}_S}\left[w_x^T \mathbb{1}_S + b_x \middle| \left|S \cap S_f\right| = t_f + 1\right] - \mathbb{E}_{S\sim\mathcal{D}_S}\left[w_x^T \mathbb{1}_S + b_x \middle| \left|S \cap S_f\right| = t_f\right] = w_x^T \mathbf{h}_f$$

where

$$\mathbf{h} = \frac{1}{p}\mathbb{1}_{S_f} - \frac{1}{n-p}\mathbb{1}_{S_f^c}$$

Proof. First observe that the bias b_x will cancel from both expressions, so we can ignore it. Note that we can write

$$\begin{aligned}
&\mathbb{E}_{S\sim\mathcal{D}_S}\left[\mathbf{w}_x^T \mathbb{1}_S \middle| \left|S \cap S_f\right| = t_f\right] \\
&= \mathbb{E}_{S\sim\mathcal{D}_S}\left[\sum_{z\in D} \mathbb{1}_{z\in S}\mathbf{w}_{xz} \middle| \left|S \cap S_f\right| = t_f\right] \\
&= \sum_{z\in D} \Pr_{S\sim\mathcal{D}_S}\left[z \in S \middle| \left|S \cap S_f\right| = t_f\right]\mathbf{w}_{xz}
\end{aligned}$$

There are a total of

$$\binom{p}{t_f}\binom{n-p}{m-t_f}$$

sets satisfying $\left|S \cap S_f\right| = t_f$, and each is sampled with the same probability. Among these, given $z \in S_f$ there are

$$\binom{p-1}{t_f-1}\binom{n-p}{m-t_f}$$

sets containing z, so for all $z \in S_f$ we have

$$\Pr\left[z \in S \middle| \left|S \cap S_f\right| = t_f\right] = \frac{\binom{p-1}{t_f-1}\binom{n-p}{m-t_f}}{\binom{p}{t_f}\binom{n-p}{m-t_f}} = \frac{t_f}{p}$$

Similarly, we have for all $z \in S_f^c$ that

$$\Pr\left[z \in S \middle| |S \cap S_f| = t_f\right] = \frac{\binom{p}{t_f}\binom{n-p-1}{m-t_f-1}}{\binom{p}{t_f}\binom{n-p}{m-t_f}} = \frac{m-t_f}{n-p}$$

So we end up with

$$\mathbb{E}_{S\sim\mathcal{D}_S}\left[\mathbf{w}_x^T \mathbb{1}_S \middle| |S \cap S_f| = t_f\right]$$
$$= \frac{t}{p}\mathbf{w}_x^T \mathbb{1}_{S_f} + \frac{m-t}{n-p}\mathbf{w}_x^T \mathbb{1}_{S_f^c}$$

Analogously,

$$\mathbb{E}_{S\sim\mathcal{D}_S}\left[\mathbf{w}_x^T \mathbb{1}_S \middle| |S \cap S_f| = t_f + 1\right]$$
$$= \frac{t+1}{p}\mathbf{w}_x^T \mathbb{1}_{S_f} + \frac{m-t-1}{n-p}\mathbf{w}_x^T \mathbb{1}_{S_f^c}$$

and the lemma follows when we subtract the two. □

The proof of Lemma 1 follows by combining the results of Lemma 2.9.1 and Lemma 2.9.2.

2.9.2 Proof of Lemma 2.4.2

We want to solve the problem

$$\max_S \mathbf{w}^\top \mathbf{h}_S.$$

Note that $\mathbf{h}_S \perp \mathbb{1}$, so $\mathbf{w}^\top \mathbf{h}_S = \overline{\mathbf{w}}^\top \mathbf{h}_S$, and the problem is equivalent to $\max_S \overline{\mathbf{w}}^\top \mathbf{h}_S$. We have

$$\begin{aligned}\overline{\mathbf{w}}^\top \mathbf{h}_S &= \frac{1}{|S|}\sum_{i\in S}\overline{w}_i - \frac{1}{n-|S|}\sum_{i\notin S}\overline{w}_i \\ &= \frac{1}{|S|}\sum_{i\in S}\overline{w}_i + \frac{1}{n-|S|}\sum_{i\in S}\overline{w}_i \\ &= \frac{n}{|S|\,(n-|S|)}\sum_{i\in S}\overline{w}_i,\end{aligned}$$

and analogously

$$\overline{\mathbf{w}}^\top \mathbf{h}_S = -\frac{n}{|S|\,(n-|S|)}\sum_{i\notin S}\overline{w}_i,$$

where we used that $\sum_i \overline{w}_i = 0$. From this expression it follows that for any maximizer S of $\overline{\mathbf{w}}^\top \mathbf{h}_S$, the elements of S must be the $|S|$ largest entries of $\overline{\mathbf{w}}$.

To prove the lemma, observe that if $w_i = \|\overline{\mathbf{w}}\|_\infty$ for some i, we can take $S = \{i\}$ and see that

$$\max_S \overline{\mathbf{w}}^\top \mathbf{h}_S \geq \frac{n}{n-1}\|\overline{\mathbf{w}}\|_\infty$$

and otherwise if $w_i = -\|\overline{\mathbf{w}}\|_\infty$ for some i, we can take $S^c = \{i\}$ and again

$$\max_S \overline{\mathbf{w}}^\top \mathbf{h}_S \geq \frac{n}{n-1}\|\overline{\mathbf{w}}\|_\infty$$

For the other side of the inequality, we have

$$\max_S \overline{\mathbf{w}}^\top \mathbf{h}_S \leq \frac{1}{|S|}\sum_{i\in S}\overline{w}_i - \frac{1}{n-|S|}\sum_{i\notin S}\overline{w}_i \leq \|\overline{\mathbf{w}}\|_\infty + \|\overline{\mathbf{w}}\|_\infty = 2\|\overline{\mathbf{w}}\|_\infty$$

Exp.	No Defense		AC		ISPL		SPECTRE		SS		Ours	
	Clean	*Poisoned*	*Clean*	*Poisoned*	*Clean*	*Poisoned*	*Clean*	*Poisoned*	*Clean*	*Poisoned*	*Clean*	*Poisoned*
1	86.64	19.90	86.76	19.68	86.13	86.15	86.71	20.17	85.52	30.99	86.51	86.49
2	86.67	12.92	85.41	12.93	85.88	85.82	-	-	85.33	13.63	86.44	86.43
3	86.39	49.57	86.25	48.85	86.32	85.57	86.28	45.32	85.22	78.22	86.23	80.08
4	86.23	10.67	84.75	10.82	85.86	85.18	-	-	84.85	13.33	85.62	84.60
5	86.89	75.58	86.73	82.83	86.04	85.89	86.82	80.65	85.67	85.41	86.48	80.08
6	87.11	41.89	86.85	51.05	86.18	86.11	86.97	51.18	85.68	85.60	86.73	86.47
7	87.02	71.68	86.90	73.28	86.50	82.31	86.72	76.97	85.70	82.70	86.69	86.32
8	86.94	52.08	86.81	56.78	86.04	71.27	86.63	52.27	85.87	71.93	86.51	86.46

Table 2.2: We compare our method against multiple baselines in a wide range of experiments. We can see that we can achieve a consistently high accuracy on the fully poisoned validation set when using our method for detecting and removing poisoned samples. Refer to Table 2.1 for the full experiments parameters.

Chapter 3

Beyond Image Classification and Neural Networks

In this chapter, we present preliminary results suggesting that the datamodeling framework [IPE+22] may be broadly useful beyond the modality of images and the use of neural networks as ML models. Specifically, we will describe the results of a study on the MIMIC-III dataset of Electronic Health Record data [JPS+16] that investigated the use of datamodeling for two applications: identifying small, *worst-case* subsets of the training data whose removal causes misclassification of a given test example x; and detecting (simple) data poisoning attacks.

3.1 Preliminaries

In this section, we give background on the dataset and the class of learning algorithms used for this study. We also present a concept – *empirical influence estimates* – that is closely related to datamodeling, and indeed equivalent to a certain regime of datamodeling. We use this concept instead of datamodeling for this application, since with gradient-boosted decision trees we have no access to a quantity analogous to the margin of correct prediction in neural networks (we have prediction probabilities instead).

3.1.1 The MIMIC-III Dataset and MIMIC-Extract

The MIMIC-III EHR (electronic health record) dataset [JPS+16] contains data associated with 53,423 distinct hospital admissions for adult patients admitted to critical care units between 2001 and 2012 in the Beth Israel Deaconess Medical Center in Boston, Massachusetts. It records many different kinds of data, including approximately-hourly nurse-verified measurements of vital signs, diagnosis at admission, demographic details, interventions, lab measurements, and free-text notes. The data is represented in a relational database with tens of tables and millions of rows in total.

We use a particular projection of the MIMIC-III database called MIMIC-Extract [WMC+20]. MIMIC-Extract is an open-source[1] data extraction and processing pipeline that transforms raw data into a table of post-processed time-series features over a 24h period of a number of admissions to intensive care units, together with labels for 4 clinically meaningful tasks, and demographics data of the patients. This data format makes it directly usable in standard machine learning algorithms.

MIMIC-Extract provides a parametrized extraction pipeline, as well as a ready-made preprocessed dataset corresponding to an execution of the pipeline with default settings; we use the latter in our experiments. We further split the entire dataset into a train and test partition using a 80%-20% split. Finally, of the four benchmark binary classification tasks[2], we choose in-ICU mortality. Some properties of the dataset thus obtained are given in table 3.1. In particular, note that the task is severely imbalanced.

3.1.2 Gradient-boosted Decision Trees

The model class we focus on in this study is gradient-boosted decision trees (GBDTs from now on) [Fri01]. Conceptually, GBDTs are ensembles of decision trees trained in sequence, where each next tree tries to fix the mistakes of the ensemble so

[1] `https://github.com/MLforHealth/MIMIC_Extract`

[2] These being in-ICU mortality, in-hospital mortality, LOS (length of stay) $>$ 3 days prediction, and LOS $>$ 7 days prediction

Property	Value
Number of training examples	19,155
Number of test examples	4,789
Total features	7,488
Time-series features	104
Measurements per time-series feature	24
Fraction of examples in class 1/0	7% / 93%
Number of training examples in class 1/0	1,377 / 17,778
Number of test examples in class 1/0	340 / 4,449

Table 3.1: Some descriptive properties of the particular version of MIMIC-III we use

far by using gradient information of the loss function. Importantly to this work, GBDTs are comparable to other tree ensembles (such as random forests), can be trained efficiently on GPU, and provide probability estimates of their predictions. In particular, GBDTs achieve AUROC scores on par with the best methods described in the MIMIC-Extract paper [WMC+20] for the task we have chosen.

Applying the datamodeling framework to this problem required us to train several million models on subsets of the data. The particular parameters of the dataset motivate the choice of the `catboost` library ([DEG18], `https://github.com/catboost/catboost`), which allows efficient GPU training for datasets with 1,000s of examples *and* features.

3.1.3 Empirical Influence Estimation

Finally, we describe a concept closely related to datamodeling. A recent line of work aims to use the *empirical influence* [HRRS11] of a training example x_i on model predictions on example x_j:

$$\begin{aligned}\text{Infl}\left[x_i \to x_j\right] :=& \mathbb{P}\left(\text{model trained on } S \text{ is correct on } x_j\right) \\ &- \mathbb{P}\left(\text{model trained on } S\setminus\{x_i\} \text{ is correct on } x_j\right)\end{aligned}$$

where randomness is over the training algorithm. A specific approximation to this quantity is developed by Feldman and Zhang [FZ20] with the purpose of estimating

Infl $\left[x_i \to x_j\right]$ for many pairs $\left(x_i, x_j\right)$ at once while re-using trained models. Assuming we have an empirical distribution $\mathcal{D}_S$ over subsets S of some training dataset D, their estimator takes the form

$$\begin{aligned}\widehat{\text{Infl}}\left[x_i \to x_j\right] =& \mathbb{P}_{S\sim\mathcal{D}_S}\left(\text{model trained on } S \text{ is correct on } x_j \middle| x_i \in S\right) \\ & - \mathbb{P}_{S\sim\mathcal{D}_S}\left(\text{model trained on } S \text{ is correct on } x_j \middle| x_i \notin S\right)\end{aligned}$$

As it turns out (see Lemma 1 in subsection 6.1. of [IPE+22]), the above estimator is equivalent to a rescaled datamodel in the infinite-sample limit. This is the quantity we use in our experiments in this chapter.

3.2 Application: Decision Support

In this section, we use empirical influence estimates to estimate data counterfactuals on the MIMIC-III dataset, focusing on the *decision support* of model predictions:

> *what is the smallest amount of training points that, when removed from the training set, lead to a misprediction on a test example x?*

This question was also considered for image classification and neural networks in [IPE+22].

3.2.1 Motivation

Any useful classification algorithm must be sensitive to *some* change in the data. However, what if we find that the label (in this case in-ICU mortality prediction) of some test example x can flip when we remove only *as few as 10 examples* from the training set?

Such behavior can indeed arise in modern data analyses ([BGM21] exhibits an econometric analysis where removing a *single* example changes the outcome), and is concerning from the point of view of data privacy [DKM+06]. However, it is particularly concerning in medical applications, where measurements can be missing not-at-random [GNS+20]. If models indeed depend on small data subpopulations to drive much of their prediction, there is a risk for some of these subpopulations to be entirely missing, or severely underrepresented, in the training data.

3.2.2 Methodology

To bring influence estimates to bear on this question, we compute a matrix $\mathbf{W}$ where $\mathbf{W}_{ij} = \widehat{\text{Infl}}\left[x_i \to x_j\right]$ where i ranges over the training set and j ranges over the test set, from a set of 100,000 models each trained on a random subset of the training dataset of relative size $\alpha \in (0, 1)$.

Then, given a test example x_j for which we want to estimate the decision support, we sort the training indices i in order of decreasing value of $\mathbf{W}_{ij}$, and remove increasingly large prefixes of this order (doubling, starting from 10 examples to be removed).

3.2.3 Baselines

In this subsection, we describe the baselines we compare influence estimates with for this application, as well as the motivation for their choice. Each baseline provides a different way to define a similarity metric between a given test example x and the training examples. Data counterfactuals are then computed by removing training examples in decreasing order of similarity.

Tree Ensemble Embedding

This baseline trains a GBDT with the same hyperparameters as the one we use in our influence estimation – but on the *entire* dataset, and uses the probability assigned to a given example x by each of the (one hundred) trees in the ensemble as a *model-aware feature representation* $\varphi(x) \in \mathbb{R}^{100}$ of x. Similarity between x and a training example x_i is calculated by Euclidean distance in this space.

This baseline was chosen because it is a simple sanity check, efficiently computable and intuitive, yet very different from the approach taken in this work. To obtain the embeddings we use subroutines of the code from [BL20] (see 3.2.3 below).

Data Shapley

Data Shapley [GZ19] is a data valuation methodology based on the game-theoretic concept of Shapley values [Sha51]. Below, we describe how the data Shapley framework can be adapted to the setting of the current work.

Given a test example x, the goal is to assign a value $\phi_i \in \mathbb{R}$ to training example $x_i \in S$ that intuitively reflects the contribution of x_i to $f_{\mathcal{A}}(x; S')$ as S' ranges over *all* subsets of S. The main observation of [GZ19] is that, if certain common-sense properties are required from the values ϕ_i, these values are in fact *forced* to satisfy

$$\phi_i = \sum_{S' \subset S - \{i\}} \frac{f_{\mathcal{A}}(x, S' \cup \{i\}) - f_{\mathcal{A}}(x, S')}{\binom{n-1}{|S'|}}$$

up to a constant scaling factor. To estimate these intractable expressions, the authors of [GZ19] rewrite it as an expectation

$$\phi_i = \mathbb{E}_{\pi \sim \Pi}\left[f_{\mathcal{A}}\left(x; S^i_\pi \cup \{i\}\right) - f_{\mathcal{A}}\left(S^i_\pi\right)\right]$$

where Π is the uniform distribution over all permutations of the training set, and S^i_π denotes the set of data points that appear before the example x_i in the permutation π. An approximation of ϕ_i can be obtained by sampling many random permutations π and averaging the results.

We implemented our own version of the data Shapley algorithm from [GZ19]. Namely, we sample 300 random permutations of the dataset, and we truncate the estimation process (as described in [GZ19]) at a prefix of size 2,000 examples. This relatively early truncation is motivated by the observation that performance quickly saturates on this dataset with increasing training set size.

This baseline was chosen because it is conceptually similar to the datamodeling approach (as it compares the behavior of many models trained on different subsets of the data), yet different enough in implementation to warrant a comparison.

T-REX

This baseline [BL20] is a *representer-point explanation method* [YKYR18] for ensembles of decision trees. It trains a simpler *surrogate model* to predict the predictions of a tree ensemble; the surrogate model is a linear method, and it is fit and validated on embeddings derived from a trained tree ensemble to predict the probabilities assigned by the ensemble. See [BL20] for more details.

In our implementation, we used the code accompanying the paper that defines T-REX [BL20]. We use the kernel logistic regression (KLR) surrogate model, and did a cross-validated hyperparameter search over the grid where C takes values $10^{-5}, 10^{-4}, \ldots, 10^{5}$ and the tree kernel ranges over all the kernels implemented for the `catboost` ensemble (`'feature_path'`, `'leaf_path'`, `'leaf_output'`, `'tree_output'`). This resulted in $C = 1.0$ and `tree_kernel='tree_output'` as the best settings, achieving TODO. See [BL20] for definitions of the hyperparameters.

This baseline was chosen because it builds a surrogate model with the explicit goal of approximating the real model's behavior. Furthermore, the surrogate model (being linear) admits a 'closed form' data valuation solution. In this sense, T-REX can be seen as a way to approximate datamodeling by replacing a non-linear class of models with a linear one.

3.2.4 Results

The results from this study are shown in figure 3.1. As we can see, the empirical influence estimates greatly outperform the baselines. In particular, with $\alpha = 0.5$, we see that for each of roughly $\frac{1}{3}$ of the test set, there exist some ~ 100 examples in the training set whose removal causes misclassification on this particular test example.

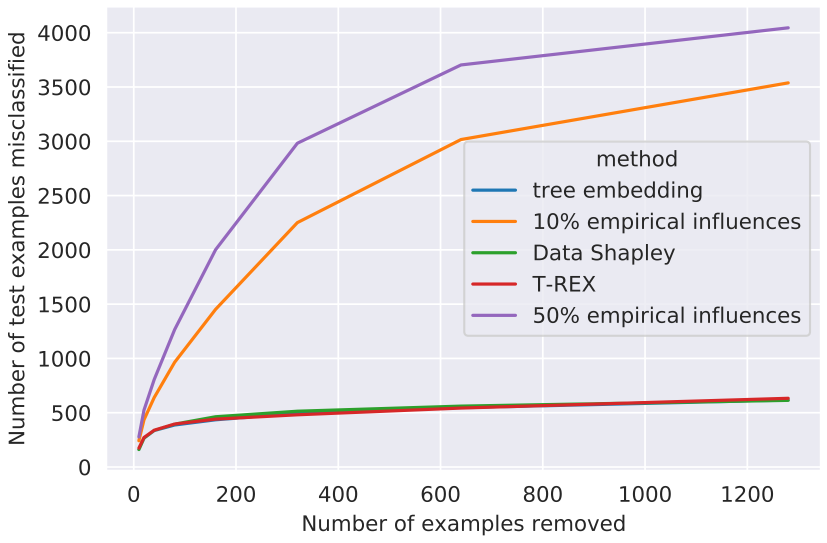


Figure 3.1: Results of the decision support experiment on the in-ICU mortality prediction task for the MIMIC-Extract variant of the MIMIC-III dataset. The x-axis shows the number of examples we remove from the training set according to the scores produced by the methods for a given test example x. The y-axis shows the cumulative number of test examples x that are misclassified by the removal of the given number (or fewer) training examples. Here we use influence estimates for 100,000 trained models in two regimes: the orange line shows the results with $\alpha = 0.1$, and the purple line with $\alpha = 0.5$. We see that $\alpha = 0.5$ leads to better results.

3.3 Application: Detecting Backdoor Attacks

In analogy with the previous chapter 2, we can also ask whether this toolkit can surface backdoors in the data. To this end, we provide preliminary evidence that this is indeed possible. Specifically, we deploy a very naive backdoor attack on the dataset – for 1% of the training dataset, we alter one of the 104 features in the data by replacing all its values with 0.0. We note that constructing backdoor attacks that are meaningful for medical data is a more complicated problem [JMSH21]; here we are only presenting a simple proof-of-concept.

As evidence for the useful information contained in the empirical influence estimates, we show in Figure 3.2 a certain two-dimensional visualization of the influence estimates for the entire dataset. This figure indicates that the influence estimates contain useful signal about which datapoints contain the backdoor. It is an interesting question for future work whether a more extensive evaluation (along the lines of 2, and using more realistic modes of data poisoning specific to the medical setting) can leverage this signal to develop a detection algorithm.

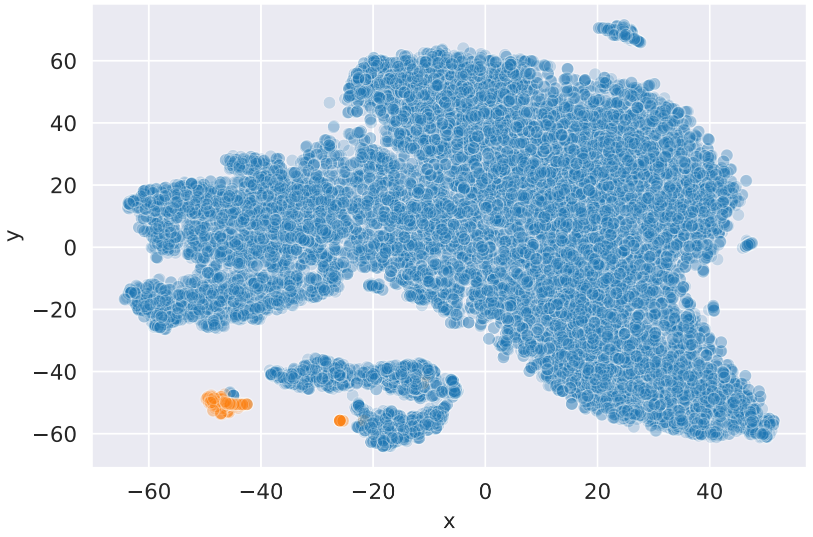


Figure 3.2: We poison 1% of the MIMIC-III dataset, and generate empirical influence estimates by training 100,000 models on random subsets of relative size 10% of the training set. We then compute the matrix of their pairwise dot products (which serves as a similarity score between training examples), and visualize this matrix in two dimensions using t-SNE. The poisoned examples (orange) are largely well-separated from the clean examples (blue).

Bibliography

[ABC$^+$18] Yossi Adi, Carsten Baum, Moustapha Cisse, Benny Pinkas, and Joseph Keshet. Turning your weakness into a strength: Watermarking deep neural networks by backdooring. *{USENIX} Security Symposium*, 2018.

[ABC$^+$20] OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

[AEIK18] Anish Athalye, Logan Engstrom, Andrew Ilyas, and Kevin Kwok. Synthesizing robust adversarial examples. In *International Conference on Machine Learning (ICML)*, 2018.

[AG17] Mahdieh Abbasi and Christian Gagné. Robustness to adversarial examples through an ensemble of specialists. *arXiv preprint arXiv:1702.06856*, 2017.

[BCM$^+$13] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. In *Joint European conference on machine learning and knowledge discovery in databases (ECML-KDD)*, 2013.

[BGM21] Tamara Broderick, Ryan Giordano, and Rachael Meager. An automatic finite-sample robustness metric: Can dropping a little data change conclusions? In *Arxiv preprint arXiv:2011.14999*, 2021.

[BL20] Jonathan Brophy and Daniel Lowd. Trex: Tree-ensemble representer-point explanations. *arXiv preprint arXiv:2009.05530*, 2020.

[BMR$^+$20] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[BR18] Battista Biggio and Fabio Roli. Wild patterns: Ten years after the rise of adversarial machine learning. 2018.

[BRB17] Wieland Brendel, Jonas Rauber, and Matthias Bethge. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. In *International Conference on Learning Representations (ICLR)*, 2017.

[BTEGN09] Aharon Ben-Tal, Laurent El Ghaoui, and Arkadi Nemirovski. *Robust optimization*. Princeton University Press, 2009.

[CCB$^+$18] Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin Edwards, Taesung Lee, Ian Molloy, and Biplav Srivastava. Detecting backdoor attacks on deep neural networks by activation clustering. *arXiv preprint arXiv:1811.03728*, 2018.

[CLL$^+$17] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526*, 2017.

[CW82] R Dennis Cook and Sanford Weisberg. *Residuals and influence in regression*. New York: Chapman and Hall, 1982.

[CW08] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *International Conference on Machine Learning (ICML)*, pages 160–167, 2008.

[CW17a] Nicholas Carlini and David Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In *Workshop on Artificial Intelligence and Security (AISec)*, 2017.

[CW17b] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *Symposium on Security and Privacy (SP)*, 2017.

[DDS$^+$09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition (CVPR)*, 2009.

[DDSV04] Nilesh Dalvi, Pedro Domingos, Sumit Sanghai, and Deepak Verma. Adversarial classification. In *International Conference on Knowledge Discovery and Data Mining*, 2004.

[DEG18] Anna Veronika Dorogush, Vasily Ershov, and Andrey Gulin. Catboost: gradient boosting with categorical features support. *arXiv preprint arXiv:1810.11363*, 2018.

[DKM$^+$06] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our data, ourselves: Privacy via distributed noise generation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2006.

[DT17] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. In *arXiv preprint arXiv:1708.04552*, 2017.

[FFF18] Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Analysis of classifiers' robustness to adversarial perturbations. *Machine Learning*, 107(3):481–508, 2018.

[FGL20] Minghong Fang, Neil Zhenqiang Gong, and Jia Liu. Influence function based data poisoning attacks to top-n recommender systems. In *Proceedings of The Web Conference 2020*, pages 3019–3025, 2020.

[Fri01] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[FZ20] Vitaly Feldman and Chiyuan Zhang. What neural networks memorize and why: Discovering the long tail via influence estimation. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 2881–2891, 2020.

[GDGG17] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733*, 2017.

[GKVZ22] Shafi Goldwasser, Michael P Kim, Vinod Vaikuntanathan, and Or Zamir. Planting undetectable backdoors in machine learning models. *arXiv preprint arXiv:2204.06974*, 2022.

[GMP$^+$17] Kathrin Grosse, Praveen Manoharan, Nicolas Papernot, Michael Backes, and Patrick McDaniel. On the (statistical) detection of adversarial examples. *arXiv preprint arXiv:1702.06280*, 2017.

[GNS$^+$20] Marzyeh Ghassemi, Tristan Naumann, Peter Schulam, Andrew L Beam, Irene Y Chen, and Rajesh Ranganath. A review of challenges and opportunities in machine learning for health. *AMIA Summits on Translational Science Proceedings*, 2020:191, 2020.

[GR06] Amir Globerson and Sam Roweis. Nightmare at test time: robust learning by feature deletion. In *Proceedings of the 23rd international conference on Machine learning*, 2006.

[GSS15] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations (ICLR)*, 2015.

[GTX$^+$22] Micah Goldblum, Dimitris Tsipras, Chulin Xie, Xinyun Chen, Avi Schwarzschild, Dawn Song, Aleksander Madry, Bo Li, and Tom Goldstein. Dataset security for machine learning: Data poisoning, backdoor attacks, and defenses. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[GZ19] Amirata Ghorbani and James Zou. Data shapley: Equitable valuation of data for machine learning. In *International Conference on Machine Learning (ICML)*, 2019.

[HKSO21a] Jonathan Hayase, Weihao Kong, Raghav Somani, and Sewoong Oh. Spectre: defending against backdoor attacks using robust statistics. *arXiv preprint arXiv:2104.11315*, 2021.

[HKSO21b] Jonathan Hayase, Weihao Kong, Raghav Somani, and Sewoong Oh. Spectre: Defending against backdoor attacks using robust statistics, 2021.

[HRRS11] Frank R Hampel, Elvezio M Ronchetti, Peter J Rousseeuw, and Werner A Stahel. *Robust statistics: the approach based on influence functions*, volume 196. John Wiley & Sons, 2011.

[HWC$^+$17] Warren He, James Wei, Xinyun Chen, Nicholas Carlini, and Dawn Song. Adversarial example defense: Ensembles of weak defenses are not strong. In *USENIX Workshop on Offensive Technologies (WOOT)*, 2017.

[HXSS15] Ruitong Huang, Bing Xu, Dale Schuurmans, and Csaba Szepesvari. Learning with a strong adversary. *arXiv preprint arXiv:1511.03034*, 2015.

[HZRS15a] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[HZRS15b] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *international conference on computer vision (ICCV)*, 2015.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[IPE$^+$22] Andrew Ilyas, Sung Min Park, Logan Engstrom, Guillaume Leclerc, and Aleksander Madry. Datamodels: Predicting predictions from training data. In *International Conference on Machine Learning (ICML)*, 2022.

[IST$^+$19] Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Madry. Adversarial examples are not bugs, they are features. In *Neural Information Processing Systems (NeurIPS)*, 2019.

[JCG21] Jinyuan Jia, Xiaoyu Cao, and Neil Zhenqiang Gong. Intrinsic certified robustness of bagging against data poisoning attacks. In *AAAI*, 2021.

[JMSH21] Byunggill Joe, Akshay Mehra, Insik Shin, and Jihun Hamm. Machine learning with electronic health records is vulnerable to backdoor trigger attacks. *arXiv preprint arXiv:2106.07925*, 2021.

[JPS$^+$16] Alistair EW Johnson, Tom J Pollard, Lu Shen, H Lehman Li-Wei, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter

Szolovits, Leo Anthony Celi, and Roger G Mark. Mimic-iii, a freely accessible critical care database. *Scientific data*, 2016.

[JSR21] Charles Jin, Melinda Sun, and Martin Rinard. Provable guarantees against data poisoning using self-expansion and compatibility. 2021.

[KDI$^+$22] Bojan Karlaš, David Dao, Matteo Interlandi, Bo Li, Sebastian Schelter, Wentao Wu, and Ce Zhang. Data debugging with shapley importance over end-to-end machine learning pipelines. *arXiv preprint arXiv:2204.11131*, 2022.

[KGB17] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial machine learning at scale. In *International Conference on Learning Representations (ICLR)*, 2017.

[KL17] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *International Conference on Machine Learning*, 2017.

[Kri09] Alex Krizhevsky. Learning multiple layers of features from tiny images. In *Technical report*, 2009.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2012.

[KZ21] Yongchan Kwon and James Zou. Beta shapley: a unified and noise-reduced data valuation framework for machine learning. *arXiv preprint arXiv:2110.14049*, 2021.

[LCWC18] Bai Li, Changyou Chen, Wenlin Wang, and Lawrence Carin. Second-order adversarial attack and certifiable robustness. *arXiv preprint arXiv:1809.03113*, 2018.

[LeC98] Yann LeCun. The mnist database of handwritten digits. In *Technical report*, 1998.

[LF21] Alexander Levine and Soheil Feizi. Deep partition aggregation: Provable defenses against general poisoning attacks. In *International Conference on Learning Representations*, 2021.

[LIE$^+$22] Guillaume Leclerc, Andrew Ilyas, Logan Engstrom, Sung Min Park, Hadi Salman, and Aleksander Madry. ffcv. https://github.com/libffcv/ffcv/, 2022.

[LJLX22] Yiming Li, Yong Jiang, Zhifeng Li, and Shu-Tao Xia. Backdoor learning: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[LKY22] Yiwei Lu, Gautam Kamath, and Yaoliang Yu. Indiscriminate data poisoning attacks on neural networks. *arXiv preprint arXiv:2204.09092*, 2022.

[LM19] Gábor Lugosi and Shahar Mendelson. Sub-gaussian estimators of the mean of a random vector. *The annals of statistics*, 47(2):783–794, 2019.

[LSI$^+$21] Guillaume Leclerc, Hadi Salman, Andrew Ilyas, Sai Vemprala, Logan Engstrom, Vibhav Vineet, Kai Xiao, Pengchuan Zhang, Shibani Santurkar, Greg Yang, et al. 3db: A framework for debugging computer vision models. In *arXiv preprint arXiv:2106.03805*, 2021.

[LZL$^+$22] Jinkun Lin, Anqi Zhang, Mathias Lecuyer, Jinyang Li, Aurojit Panda, and Siddhartha Sen. Measuring the effect of training data on deep learning predictions via randomized experiments. *arXiv preprint arXiv:2206.10013*, 2022.

[MFF16] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Computer Vision and Pattern Recognition (CVPR)*, 2016.

[MGBD$^+$17] Luis Muñoz-González, Battista Biggio, Ambra Demontis, Andrea Paudice, Vasin Wongrassamee, Emil C Lupu, and Fabio Roli. Towards poisoning of deep learning algorithms with back-gradient optimization. In *Proceedings of the 10th ACM workshop on artificial intelligence and security*, pages 27–38, 2017.

[MMS$^+$18] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations (ICLR)*, 2018.

[NYC15] Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Conference on computer vision and pattern recognition (CVPR)*, 2015.

[PMG16] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. In *ArXiv preprint arXiv:1605.07277*, 2016.

[PMW$^+$16] Nicolas Papernot, Patrick D. McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *Symposium on Security and Privacy (SP)*, 2016.

[QXMM22] Xiangyu Qi, Tinghao Xie, Saeed Mahloujifar, and Prateek Mittal. Circumventing backdoor defenses that are based on latent separability. *arXiv preprint arXiv:2205.13613*, 2022.

[S$^+$20] Reza Shokri et al. Bypassing backdoor detection algorithms in deep learning. In *2020 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 175–183. IEEE, 2020.

[SGSR17] Jure Sokolić, Raja Giryes, Guillermo Sapiro, and Miguel RD Rodrigues. Robust large margin deep neural networks. In *Transactions on Signal Processing*, 2017.

[Sha51] LS Shapley. Notes on the n-person game—ii: The value of an n-person game, the rand corporation, the rand corporation. In *Research Memorandum*, 1951.

[SHN$^+$18] Ali Shafahi, W Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. Poison frogs! targeted clean-label poisoning attacks on neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[SHS$^+$18] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[SRBB19] Lukas Schott, Jonas Rauber, Matthias Bethge, and Wieland Brendel. Towards the first adversarially robust neural network model on MNIST. In *International Conference on Learning Representations (ICLR)*, 2019.

[SSP20] Aniruddha Saha, Akshayvarun Subramanya, and Hamed Pirsiavash. Hidden trigger backdoor attacks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 11957–11965, 2020.

[SYN18] Uri Shaham, Yutaro Yamada, and Sahand Negahban. Understanding adversarial training: Increasing local stability of supervised models through robust optimization. *Neurocomputing*, 307:195–204, 2018.

[SZS$^+$14] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations (ICLR)*, 2014.

[TLM18] Brandon Tran, Jerry Li, and Aleksander Mądry. Spectral signatures in backdoor attacks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[TPGDB17] Florian Tramer, Nicolas Papernot, Ian Goodfellow, and Patrick McDaniel Dan Boneh. The space of transferable adversarial examples. In *ArXiv preprint arXiv:1704.03453*, 2017.

[TTM19] Alexander Turner, Dimitris Tsipras, and Aleksander Madry. Label-consistent backdoor attacks. 2019.

[Wal45] Abraham Wald. Statistical decision functions which minimize the maximum risk. In *Annals of Mathematics*, 1945.

[WMC$^+$20] Shirly Wang, Matthew BA McDermott, Geeticka Chauhan, Marzyeh Ghassemi, Michael C Hughes, and Tristan Naumann. Mimic-extract: A data extraction, preprocessing, and representation pipeline for mimic-iii. In *Proceedings of the ACM conference on health, inference, and learning*, pages 222–235, 2020.

[WYS$^+$19] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y Zhao. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *Proceedings of 40th IEEE Symposium on Security and Privacy*, 2019.

[XEQ18] Weilin Xu, David Evans, and Yanjun Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. In *Network and Distributed Systems Security Symposium (NDSS)*, 2018.

[XHCL20] Chulin Xie, Keli Huang, Pin-Yu Chen, and Bo Li. Dba: Distributed backdoor attacks against federated learning. In *International Conference on Learning Representations*, 2020.

[YKYR18] Chih-Kuan Yeh, Joon Sik Kim, Ian E. H. Yen, and Pradeep Ravikumar. Representer point selection for explaining deep neural networks. In *Neural Information Processing Systems (NeurIPS)*, 2018.