Provable, structured, and efficient methods for robustness of deep networks to adversarial examples

Eric Wong

May 2020
CMU-ML-20-102

Machine Learning Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee

J. Zico Kolter (Chair)
Barnabás Póczos
Matt Fredrikson
Aleksander Mądry (MIT)

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This research was funded by Air Force Research Laboratory award FA87501720027, Defense Advanced Research Projects Agency award FA87501620207, Department of Energy award DEOE0000684, National Science Foundation award CCF1522054, Office of Naval Research award N660011714036, a fellowship from the Siebel Scholars Foundation, and Bosch contract 0087016732PCR.

Keywords: adversarial examples, provable defenses, adversarial training, deep networks

Dedicated to my parents

Abstract

While deep networks have contributed to major leaps in raw performance across various applications, they are also known to be quite brittle to targeted data perturbations. By adding a small amount of adversarial noise to the data, it is possible to drastically change the output of a deep network. The existence of these so-called *adversarial examples*, perturbed data points which fool the model, pose a serious risk for safety- and security-centric applications where reliability and robustness are critical. In this dissertation, we present and analyze a number of approaches for mitigating the effect of adversarial examples, also known as adversarial defenses. These defenses can offer varying degrees and types of robustness, and in this dissertation we study defenses which differ in the strength of the the robustness guarantee, the efficiency and simplicity of the defense, and the type of perturbation being defended against.

We start with the strongest type of guarantee called provable adversarial defenses, showing that is possible to compute duality-based certificates that guarantee no adversarial examples exist within an ℓ_p-bounded region, which are trainable and can be minimized to learn networks which are provably robust to adversarial attacks. The approach is agnostic to the specific architecture and is applicable to arbitrary computational graphs, scaling to medium sized convolutional networks with random projections.

We then switch gears to developing a deeper understanding of a more empirical defense known as adversarial training. Although adversarial training does not come with formal guarantees, it can learn networks more efficiently and with better empirical performance against attacks. We study the optimization process and reveal several intriguing properties of the robust learning problem, finding that a simple modification to one of the earliest adversarial attacks can be sufficient to learn networks robust to much stronger attacks, as well as finding that adversarial training as a general procedure is highly susceptible to overfitting. These discoveries have significant implications on both the efficiency of adversarial training as well as the state of the field: for example, virtually all recent algorithmic improvements in adversarial training can be matched by simply using early stopping.

The final component of this dissertation expands the realm of adversarial examples beyond ℓ_p-norm bounded perturbations, to enable more realistic threat models for applications beyond imperceptible noise. We define a threat model called the Wasserstein adversarial example, which captures semantically meaningful image transformations like translations and rotations previously uncaptured by existing threat models. We present an efficient algorithm for projecting onto Wasserstein balls, enabling both generation of and adversarial training against Wasserstein adversarial examples. Finally, we demonstrate how to generalize adversarial training to defend against multiple types of threats simultaneously, improving upon naive aggregations of adversarial attacks.

Acknowledgments

I give my deepest thanks to my advisor Zico Kolter, who has mentored me from when I was a fledgling undergraduate student that knew virtually nothing about machine learning but thought wind turbines were cool. I am exceedingly grateful to Zico for always putting my own development and interests first, patiently guiding me to be a better researcher and human being, and believing in my work every step of the way for over seven years. Many thanks as well to Barnabás Póczos, Matt Fredrikson, and Aleksander Mądry for serving on my committee. I am grateful for their time, advice, and encouragement for this dissertation.

I would also thank all of my collaborators throughout the years that have had an impact on both the work in this dissertation as well as my own growth as a researcher. I thank Alnur Ali, for showing me the ropes in my early years, and for your patience in explaining the answers to all of my statistical questions. I thank Frank J. Schmidt for directly guiding and watching out for me throughout my time at Bosch in Germany, providing valuable and entertaining cultural and research insights in a foreign country. I thank Leslie Rice and Ezra Winston for collaborating with me on multiple projects, trusting me to lead and direct the research and building my confidence as a senior researcher. Leslie's help was instrumental for the work in Chapter 4. I thank Pratyush Maini for his continual work and belief in my mentorship, and for his contribution to part of the work in Chapter 5.

Many thanks as well to both the current and past members of Locus Lab, who continue to foster such a welcoming and enjoyable group dynamic while exposing me to numerous, drastically different research fields. I thank Brandon Amos for showing me the many uses of generative modeling, generating images beyond my imagination. I thank Po-Wei Wang for sharing his endless optimization knowledge and motivational drive, and for thinking of us when abroad and bringing back food souvenirs. I thank Priya Donti with whom I have had numerous insightful discussions, about not just sustainability research but also about life in general. I thank Vaishnavh Nagarajan for teaching me about generalization and his incredible sense of humor. I thank Gaurav Manek for making all the treats around deadlines and am incredibly thankful for his enormous efforts in upgrading and maintaining the lab servers. I thank all the other lab members, with whom I've shared numerous discussions and interactions within our overlapping time in Locus Lab: Shaojie Bai, Jeremy Cohen, Rizal Fathony, Saurabh Garg, Chun Kai Ling, Filipe de Avila Belbute-Peres, Mel Roderick, Mingjie Sun, Asher Trockman, and Josh Williams, as well as past lab members including Matt Wytock and Xiao Zhang.

I am especially thankful to Diane Stidle, who's tireless efforts behind the scenes have cultivated such a welcoming department that truly cares for its students. I also thank Ann Stetser for helping me navigate numerous intricacies and saving me from an almost comically constant string of bad luck.

I am grateful to all of my friends and family who have supported me over the years. I thank my parents for their unconditional support and never-ending concern for my health and wellbeing. I thank my siblings for always welcoming me with

smiles and open arms during my brief visits back home. I thank my friends for being with me to go on eye-opening travels, entertain each other, and partake in delicious foods, all while offering encouragement and motivating me to pursue my goals. Last but not least, I thank my best friend and partner Stephanie Wang, for bringing me joy and always reminding me throughout these years that there is more to life than work.

Contents

Chapter 1

Introduction

While artificial intelligence continues to become more ubiquitous in everyday life, there are still fundamental properties of these systems which are still not understood. Of particular concern to systems which need robust and reliable behavior is the prevalence of adversarial examples in deep learning, which are inputs to the model which look indistinguishable from normal examples but can completely fool the model. These adversarial examples indicate that the deep architectures we use for artificial intelligence are exceptionally brittle and potentially highly exploitable. This makes it difficult for higher stakes applications such as safety and health to put deep learning models into production due to the lack of guarantees and potential for misuse.

To tackle this problem, in this dissertation we study methods for learning networks which are robust to this phenomenon, also known as defenses against adversarial examples. The techniques in this work can be broadly divided into two main types of approaches for achieving robustness, namely provable methods and adversarial training. Both of these defenses have their benefits and downsides, with neither being strictly better than the other, and one can be selected based on the requirements of the application. In general, both of these defenses change the standard optimization procedure of deep learning without changing the specific architectures or adding extra pre-processing steps, in order to learn a set of network weights which are less susceptible to adversarial examples than networks trained with standard techniques.

1.1 Contributions

In this initial chapter, we summarize the primary contributions of this dissertation, and discuss the significance of this work in the context of adversarial examples as well as from a broader perspective. Chapter 2 will follow up with a more detailed presentation on the background of adversarial examples, discussing how the field has evolved and the main research directions being studied. The following three chapters present the work done in this dissertation towards mitigating adversarial examples, which propose provable defenses, enhance our understanding of adversarial training, and advance threat models to situations beyond ℓ_p perturbations (Chapters 3, 4, and 5 respectively). Finally, we will end the dissertation with a reflection on the work in this thesis, and pose some open questions for the field of adversarial examples.

1.1.1 Formal guarantees for deep networks

Chapter 3 proposes and studies *provable defenses*. These are methods which can formally guarantee properties of neural networks, in this case the non-existence of adversarial examples. These methods do not rely on empirical means for evaluating robustness and thus provide the strongest kind of guarantee, but at the cost of relatively high computational complexity over standard training methods and high degrees of regularization which can affect standard performance metrics. In this dissertation, we present methods based on linear programming relaxations and duality to create provable defenses which are efficient and scalable up to medium sized networks, resulting in a fully modular framework for computing bounds for deep networks.

This work in provable defenses came at a time when adversarial defenses were having their first crisis: a large number of proposed heuristics to mitigate adversarial examples were found to be completely ineffective [Buckman et al., 2018, Guo et al., 2017, Papernot et al., 2016b, Song et al., 2017], and there was an apparent arms-race between "attackers" and "defenders" with the attackers being almost universally victorious [Athalye et al., 2018a, Carlini and Wagner, 2017b, Uesato et al., 2018]. In this setting, the work in Chapter 3 was one of the earliest approaches to propose a provable defense which could put an end to this arms race once and for all by relying on principled, formal guarantees, and was the first verified defense to scale beyond fully-connected two-layer networks. This represented a breakthrough in provable adversarial robustness, which made it possible to finally learn a convolutional MNIST classifier which was formally guaranteed to be robust [Wong and Kolter, 2017], and advancing the problems in provably robust deep learning to harder, more complicated settings such as CIFAR10.

The mere ability to perform meaningful, formal verification in the deep network setting may be quite suprising on its own, as deep networks are notoriously known for being highly complex and inexplainable. Despite having been extensively studied previously within contexts beyond adversarial examples, existing formal verification methods for deep networks could not scale due to combinatorial complexity [Carlini and Wagner, 2017b, Carlini et al., 2017, Cheng et al., 2017, Ehlers, 2017, Huang et al., 2017, Katz et al., 2017, Lomuscio and Maganti, 2017, Tjeng and Tedrake, 2017]. On the other hand, a convex relaxation of a deep network may be faster (taking polynomial time), but is likely to be extremely loose and provide vacuous results [Weng et al., 2018, Zhang et al., 2018] or not scale beyond small networks [Raghunathan et al., 2018a,b]. The provable defenses in this dissertation avoid both pitfalls: it is capable of performing verification with *linear* complexity while also providing meaningful guarantees via training.

The significance of this work goes beyond the narrow setting of image classification which is robust to imperceptible noise. As deep learning is applied in new fields and applications, our work makes it possible to learn deep networks with meaningful specifications and properties beyond test set generalization. For example, the provable defenses in this dissertation have been used to learn virtual sensors for fuel injection in vehicles to provide meaningful sensitivity specifications under sensor noise [Wong et al., 2020b]. By training for and certifying desired properties of deep networks, the work in this dissertation opens up, to some degree, the black box of deep networks to more higher-stakes applications such as health care or autonomous driving.

1.1.2 Uncovering properties of adversarial training

Adversarial training is a faster optimization procedure which typically achieves better empirical results and is more scalable than provable defenses, but is generally not proven to be formally robust. However, they have thus far withstood the test of time and remain empirically robust, and can serve as an intermediate stepping stone towards creating provable defenses. In Chapter 4, we identify several intriguing and surprising properties of adversarial training, discovering several overfitting properties specific to adversarial training which overturn longstanding views within this field.

We first present the finding that training against weak adversarial attacks can actually learn models which are robust to much stronger attacks. This overturned a long-standing belief that single-step attacks were insufficient for learning robust models [Tramèr et al., 2017]. To find out why previous attempts had failed, we uncover a phenomenon called "catastrophic overfitting" which results in a complete and rapid failure of single-step adversarial training, and propose a simple adjustment which allows single-step adversarial training to succeed.

This work has major implications on the computational requirements of adversarial training [Madry et al., 2017], which was previously orders of magnitude more expensive than standard training due to their reliance on using multi-step attacks. Although adversarial training can be less expensive than provable defenses, it has typically struggled to scale to typical, large-scale deep learning problems like ImageNet without using an enormous amount of resources [Xie et al., 2019]. Our adjusted single-step adversarial training approach achieves robust performance which is almost on par with multi-step adversarial training [Madry et al., 2017], while being significantly faster. We highlight the speed of the approach by leveraging fast techniques from standard training to accelerate robust, single-step adversarial training, demonstrating for the first time that adversarial training can be as computationally fast as standard training and opening the door for adversarially robust training to be applied to large-scale problems.

Even when successful, we identify another more general property of adversarial training, that overfitting is a dominant phenomenon in adversarial training. Crucially, we find that robust test error can be drastically harmed by training for too long, and so large gains in robust performance can be obtained by early stopping. This has unfortunate ramifications for the state of adversarially robust training, where due to inconsistencies in reporting and methodology in prior work, we find that early stopping the most basic form of multi-step adversarial training [Madry et al., 2017] outperforms all recent algorithmic improvements to adversarial training, suggesting that no algorithmic progress has been made in learning empirically robust deep networks since then. This re-establishes the effectiveness of the baseline adversarial training defense, and highlights the need for future work to follow best-practices in machine learning such as using held-out validation sets and reporting model-selection criteria.

Another key finding here is that existing methods and explanations for overfitting and generalization (both from classical and deep learning perspectives) fail to explain overfitting in the adversarially robust training setting, and raises further questions such as why does overfitting occur in the adversarial training setting, and how can we prevent it? In our search for an explanation, the only approach which could substantially improve upon early stopping was to use semi-supervised data augmentation, confirming to some degree the hypothesis that robust training requires more data [Schmidt et al., 2018]. However, from the work in this dissertation it is

clear that learning in the adversarial training setting behaves significantly differently from the standard training setting.

1.1.3 Advancing threat models beyond ℓ_p balls

Chapter 5, we take a step back from learning robust models and define new threat models for generating adversarial examples that are more general and structured. Key components for this work are that the proposed threat models are mathematically well-defined and use prior knowledge to leverage known structures. In particular, a well-defined threat model is critical for accurately measuring the progress of adversarial defenses against attacks in a meaningful way.

To bring adversarial training beyond the setting of norm-bounded perturbations, we first propose the Wasserstein adversarial example for images. The Wasserstein metric has been widely successful for images, and more naturally captures semantic image transformations such as rotations, translations, and distortions, encoding structure into the threat model which ℓ_p perturbations lack. We demonstrate how to generate Wasserstein adversarial examples, which result in semantically meaningful perturbations, and use adversarial training to train baseline models which are robust to Wasserstein attacks. The Wasserstein attack highlights a limitation of the provable defenses in this thesis: although we can tie it into our general framework for provable defenses, our reliance on using interval bounds for the activations of the network are fundamentally incompatiable with the Wasserstein attack.

In addition to providing a new threat model, this work provides an algorithmic contribution in the form of an efficient, approximate projection algorithm onto Wasserstein balls. Computing Wasserstein distances typically requires solving an optimization problem and is generally computationally expensive, and so in this dissertation we formulate an entropy-regularized Wasserstein projection inspired by the Sinkhorn iteration [Cuturi, 2013] and derive a fast, block coordinate descent algorithm in the dual space for solving it. We make further improvements in efficiency by leveraging local transport plans, which scales the approach to high dimensional problems such as RGB images and critically makes adversarial training against Wasserstein attacks computationally feasible.

Finally, we study the setting of defending against the union of multiple perturbations sets as a more general threat model. We analyze basic approaches in this space, and present a natural algorithm for improving the training procedure to improve the final robustness performance against the union of multiple adversaries. As new threat models defining different types of perturbation sets are defined, this can be seen as the next natural step towards a learning a more human-like classifier which is robust to all perturbation sets simultaneously, as it is well-known at this point that adversarially robust training does not necessarily generalize beyond the threat model for which it was trained against [Kang et al., 2019]. This work finds that combining multiple perturbation sets may not be so straightforward due to imbalances in strengths between adversaries, which our proposed algorithm improves upon.

1.2 Itemized summary of contributions and code repositories

- Chapter 3 presents a provable defense based on linear programming and duality, and discusses the various techniques used to make them tractable and applicable to modern deep architectures.
  - Section 3.1 discusses the linear programming formulation for getting bounds on adversarial examples, and how to compute this efficiently with dual feasible solutions, summarizing the work done in Wong and Kolter [2017].
  - Section 3.4 discusses how to scale the the approach with random Cauchy projections, reducing computational complexity to being linear instead of quadratic in the size of the network, while generalizing to arbitrary network architectures and honing robustness further with network cascades, covering the work done in Wong et al. [2018].
  - All code related to the work in these two sections is available at

    `https://github.com/locuslab/convex_adversarial`.
- Chapter 4 presents several unique and unexpected properties of training adversarially robust networks, which can greatly speed up robust learning and improve generalization.
  - Section 4.1 discusses how adversarial training can succeed with extremely weak adversaries, in contrast to the need for strong adversaries at evaluation time. Code for this section is available at

    `https://github.com/locuslab/fast_adversarial`.
  - Section 4.3 discusses the interactions of overfitting with adversarially robust training, drawing similarities and differences to the standard setting and exploring methods to mitigate overfitting. Code for this section is available at

    `https://github.com/locuslab/robust_overfitting`.
- Chapter 5 presents new threat models for adversarial robustness.
  - Section 5.1 discusses how to generate Wasserstein adversarial examples to generate semantically meaningful image perturbations for use in adversarial training. Code for this section is available at

    `https://github.com/locuslab/projected_sinkhorn`.
  - Section 5.3 discusses how to generalize adversarial training to multiple threat models. Code for this section is available at

    `https://github.com/locuslab/robust_union`.

Chapter 2

Background

The topic of adversarial examples for deep learning is relatively young, with their initial discovery posted on arXiv in December of 2013. In this chapter, we present an overview containing a more in-depth background of the research in this field, spanning a range of topics from threat models and adversarial attacks to adversarial defenses. In the process, we present a retrospective on how the field has developed over the years and the main research directions which have progressed our understanding of robust deep learning.

2.1 Adversarial examples: threats and attacks

Adversarial examples were originally introduced by Szegedy et al. [2014] as data points fed to a machine learning algorithm which are visually indistinguishable from "normal" examples, but which are specifically tuned so as to fool or mislead the machine learning system. These earliest forms of adversarial examples were framed as an intriguing property of neural networks, where even a single, small gradient step was sufficient to to harm the performance of deep learning classifiers [Goodfellow et al., 2015]. Nowadays, methods for generating these adversarial examples are significantly more sophisticated and powerful to the point where it is now expected for standard models to completely fail (e.g. achieve zero accuracy) when evaluated on adversarial examples.

In its most fundamental form, the adversarial example can be framed as a solution to a constrained optimization problem, where an adversary is trying to maximize a loss of a model within some constrained set around the input. Specifically, let x, y be a data point and its corresponding label, let f be some classifier (e.g. a deep network), and let $\Delta(x)$ represent a set of allowable perturbations from which the adversary is allowed to search over. Then, an adversarial example x' can be found by solving the following optimization problem

$$\arg\max_{x' \in \Delta(x)} \ell(f(x'), y) \tag{2.1}$$

using some loss function ℓ. The loss incurred by the adversarial example is called the *adversarial loss*. In other words, the "adversary" performing this maximization is trying to find some perturbed example within $\Delta(x)$ which incurs a high loss for the given classifier f in order to break the model and force misclassification.

2.1.1 Threat models

Perhaps the most characteristic component of an adversarial example is the set of allowable perturbations $\Delta(x)$, comonly referred to as the threat model. This controls what the adversarial example is allowed to manifest as, with implications on the strength and characterization of the adversarial example. A commonly used threat model is called the ℓ_p perturbation. This is an ℓ_p-norm bounded ball around an unperturbed input x for some radius $\epsilon > 0$, more formally described as

$$\Delta(x) = \{x' : \|x' - x\|_p \leq \epsilon\}. \tag{2.2}$$

It is common practice at this point to take ϵ to be small enough such that that the ℓ_p perturbation represents *imperceptible noise*. For example, this can manifest as an ℓ_∞ ball with radius $8/255$ on RGB images like CIFAR10, which is difficult to see with the human eye. More general distance metrics beyond those induced by the ℓ_p norm can be used as well, such as the Wasserstein metric [Wong et al., 2019], but also tend to be imperceptible. The notion that adversarial examples use imperceptible perturbations stems from their original discovery, when it was found that visually identical images could be classified completely differently by deep networks [Szegedy et al., 2014], which Goodfellow et al. [2015] adapted to an ℓ_∞ threat model with a single gradient step attack called the Fast Gradient Sign Method.

What is considered to be an adversarial example has since then expanded in scope beyond imperceptible changes, in particular those which manifest in the real world on real machine learning systems. A common thread amongst most adversarial attacks, including real-world attacks, is that the threat model consists of *changes to the data under which a reasonable human classifier would not change*. For images, this subsumes the previously mentioned imperceptible changes, as a human which cannot see a difference would not change their mind, but includes other image transformations such as spatial transformations like rotations, translations, or distortions [Engstrom et al., 2017, Xiao et al., 2018], which when done adversarially can vastly degrade image classifier performance. Adversarial glasses can be used to fool facial recognition software [Sharif et al., 2016], while physical 3D objects can be printed with adversarial textures [Athalye et al., 2018b] to be misclassified. Adversarial patches can be printed and added to virtually any scene to break a classifier [Brown et al., 2017], and stop signs can be adversarially corrupted with seemingly innocuous graffiti or stickers to break traffic sign classifiers [Eykholt et al., 2018]. Adversarial audio can trick speech recognition systems [Carlini and Wagner, 2018, Du et al., 2019] while semantically and syntactically similar texts can fool language models [Alzantot et al., 2018]. All of these examples are clearly "perceptible" by humans and yet humans are not affected by these changes, which demonstrates how the notion of an adversarial example has matured to learning human-like invariants encoded by the threat model into our deep learning models.

There has been some discussion in the community regarding the relevance of the ℓ_p adversarial example. Although initially motivated as an "imperceptible" perturbation (e.g. to the naked human eye), nearness according to ℓ_p norm is generally neither a sufficient nor necessary criterion for visual imperceptibility [Sharif et al., 2018]. Other work has noted the inability of ℓ_p robustness to generalize to more meaningful perturbations beyond the ℓ_p norm, and have proposed looking at other measures of robustness such as natural adversarial examples [Hendrycks et al., 2019] or sets of common corruptions [Hendrycks and Dietterich, 2019]. That being said,

corruptions of small ℓ_∞ norm do remain imperceptible to the human eye, and the more general case for studying ℓ_p robustness can be motivated two-fold as 1) obtaining a better understanding of the gap between deep networks and human classifiers, which are robust to ℓ_p perturbations and 2) a mathematically well-defined instantiation of learning deep networks with invariants, in this case stability of classification over small regions. While there may be some debate over the usefulness of ℓ_p robustness in real settings, it is a necessary step towards learning classifiers with human-level performance and remains a property that we would like deep networks to have.

2.1.2 Adversarial attacks

Given a threat model $\Delta(x)$ which defines the set of perturbations, the next component of an adversarial attack is to actually find an adversarial example, a specific perturbation which incurs a high loss for the classifier within the threat model, effectively solving to some degree the optimization problem from Equation (2.1). The maximization here is critical, as deep learning classifiers can often perform well against random perturbations. Attacks can be considered as targeted or untargeted, which characterizes whether an attack is trying to force a classifier to produce a particular label, or simply trying to make the classifier output any incorrect label. The notion of targeted or untargeted can be encoded in the loss function of the adversarial attack. For example, maximizing the standard cross-entropy loss with respect to the correct label corresponds to a untargeted attack, whereas minimizing the cross-entropy loss with respect to an incorrect label corresponds to a targeted attack.

Standard deep learning classifiers are now known to be notoriously susceptible to even weak adversarial attacks: one of the earliest methods for generating adversarial examples called the Fast Gradient Sign Method (FGSM) used only a single gradient step to significantly harm the performance of a classifier [Goodfellow et al., 2015], as seen in Equation (2.3) for a step size of ϵ and example x:

$$x' = x + \epsilon \cdot \text{sign}(\nabla_x \ell(f(x), y)) \tag{2.3}$$

This attack performs a fairly coarse first-order approximation of the adversarial attack for the ℓ_∞ threat model with radius ϵ, however similar variations can be performed for other ℓ_p threat models. In general, this can be viewed as a more general gradient step known in the optimization literature as the *direction of steepest ascent*, which finds the steepest direction which maximizes the objective with a first-order Taylor approximation. The steepest ascent generalization of the FGSM attack is shown in Equation (2.4)

$$x' = x + \underset{\|\nu\| \leq \epsilon}{\arg\max} \nabla_x \ell(f(x), y)^T \nu \tag{2.4}$$

where different choices in ℓ_p norm lead to different algorithms, with the ℓ_2 norm reducing to the familiar gradient descent setting.

While this may have worked for standard classifiers, the adversarial attack has had to evolve over time as new methods were proposed to mitigate the effect of adversarial examples. Adversarial attacks can be roughly categorized into two groups, depending on whether they leverage gradient information of the model being attacked. These roughly correspond to what is referred to as "white box" and "black box" attacks in the security setting, which characterize the amount

of information available to the attacker. The mainstay of adversarial attacks which leverage gradient information is a straightforward generalization of the single-step attack known as the Basic Iterative Method [Kurakin et al., 2017a], more commonly referred to as a Projected Gradient Descent (PGD) adversary [Madry et al., 2017]. Here, the adversary repeatedly takes smaller FGSM steps while projecting onto the original threat model to find a better approximate solution to the adversarial attack. For example, for the ℓ_∞ threat model, a PGD adversary repeats the following iteration:

$$x' = \text{clip}(x + \alpha \cdot \text{sign}(\nabla_x \ell(f(x'), y)), -\epsilon, \epsilon) \tag{2.5}$$

where $\alpha < \epsilon$ is the step size and the adversarial example x' is either initialized to the original example x or at a randomly perturbed initial point within the threat model. As with the FGSM attack, the PGD attack can be generalized to other norms by using the corresponding steepest ascent step as follows:

$$x' = \mathcal{P}_{\Delta(x)}(x' + \arg\max_{\|\nu\| \leq \epsilon} \nabla_x \ell(f(x'), y)^T \nu) \tag{2.6}$$

where $\mathcal{P}_{\Delta(x)}$ is the projection operator onto the threat model $\Delta(x)$.

The PGD adversary is perhaps the most widely used and studied adversary in the literature, and has become a standard and consistent benchmark when evaluating robustness in the ℓ_∞ setting. Numerous incremental improvements have been proposed for the PGD adversary, with varying degrees of success. Traditional optimization tricks such as momentum were incorporated into the PGD adversary, and can in some cases lead to a stronger attack [Dong et al., 2018]. Multiple restarts and more iterations can improve the effectiveness of the attack [Uesato et al., 2018]. Non-differentiable model components can be replaced with differentiable approximations and still result in effective attacks [Athalye et al., 2018a]. Models with built-in randomness can be attacked by averaging over the random components to compute expected gradients [Athalye and Sutskever, 2017]. Unfortunately, choosing poor hyperparameters (e.g. step size, number of iterations, number of restarts, initialization scheme) can result in sub-par performance and so a non-trivial amount of effort was needed to select reasonable hyperparameters for each setting. However, parameter-free versions of the PGD adversary have since been developed which significantly improves the power of the attack without needing to tune any hyperparameters [Croce and Hein, 2020]. Amongst the ℓ_p norms, the ℓ_∞ PGD attack has seen the most stable and consistent results.

There are several additional adversarial attacks beyond the PGD adversary which also utilize gradient information from the model, which are more specific to the threat model being attacked. A form of L-BFGS was used to construct the earliest known adversarial examples [Szegedy et al., 2014], although the method is no longer in use as it has been eclipsed by more efficient attacks. This was followed by the DeepFool attack, which was more efficient and uses a specialized technique based on linear hyperplanes optimized for ℓ_2 adversarial examples in the untargeted setting. The Jacobian-based Saliency Map Attack uses the gradient with respect to the input to select pixels in an image to completely saturate, resulting in an ℓ_0 attack [Papernot et al., 2016a]. The Elastic-Net attack produces ℓ_1 adversarial examples with ℓ_2 adversarial examples as a special subcase, and SPSA has also been explored as a viable adversarial attack [Uesato et al., 2018]. Of all the alternatives to the PGD adversary, the CW attack is perhaps the most frequently used

[Carlini and Wagner, 2017b]. Although it takes multiple gradient steps to increase a loss similar to the PGD adversary, it uses a Lagrangian penalty on the ℓ_p norm of the perturbation rather than explicitly constraining it to a specific radius. The CW attack is better suited for the ℓ_2 setting than the ℓ_∞ setting, and generally requires more iterations than a PGD adversary in order to gradually decay regularization hyperparameters to obtain comparable performance.

In the event that model gradients cannot be computed, it is still possible to generate adversarial examples with query access to a model, commonly referred to as a black box attack, of which one of the earliest is known as a transfer attack. This class of attacks leverages a separate, known surrogate model to generate adversarial examples with gradient-based attacks in the hopes that adversarial examples generated on a surrogate model transfer to the target unknown model [Papernot et al., 2017]. If prediction or confidence score outputs are available from the model, then the transfer attack can be further improved to use less queries with greater power [Guo et al., 2019].

Black box attacks which do not leverage external models are often referred to as *decision-based* or *score-based* attacks, since they only rely on the decision or score output of the model. For example, the boundary attack uses rejection sampling for finding adversarial examples with progressively smaller ℓ_2 difference [Brendel et al., 2017], which was later adapted to use gradient information to be faster and more effective in more general ℓ_p settings [Croce and Hein, 2019a]. Black box ℓ_∞ and ℓ_2 attacks can be computationally inefficient and require many queries to be effective [Li et al., 2019b], however approaches based on random search using score outputs have made improvements in this space to be reasonably efficient [Andriushchenko et al., 2019]. For the ℓ_0 setting, a single-pixel and small local groups of pxiels are perturbed using either greedy heuristics [Narodytska and Kasiviswanathan, 2016] or differential evolution [Su et al., 2019], as well as a multi-pixel attack called the pointwise attack which greedily minimizes the ℓ_0 norm [Schott et al., 2019].

Although the black box attack has less available information and is thus theoretically weaker than, for example, a PGD adversary, in certain situations the black box attack can sometimes outperform gradient-based methods. This is a property identified in the literature as *gradient masking*, where gradient-based attacks like the PGD adversary are led to poor local optima by the local gradients and fail to break the model [Athalye et al., 2018a], while at the same time, black box attacks can successfully attack the model. This can sometimes be the case in the ℓ_2 setting, and is quite frequently the case in the ℓ_1 and ℓ_0 setting, where gradient based approaches can fail quite easily [Maini et al., 2019]. As a result, it is often recommended in these settings to perform black box attacks in addition to white box attacks when evaluating adversarial robustness [Carlini et al., 2019]. Since well-tuned gradient-based attacks are not as prone to gradient masking in the ℓ_∞ and ℓ_2 settings, black box attacks are not nearly as widely-used in these settings when gradient information is available.

Adversarial attacks have also been shown to be effective in the real world, however tend to use threat models which are quite different from the usual ℓ_p setting. For example, adversarial glasses can be 3D printed to fool facial recognition software but needs to be constrained to look like normal glasses Sharif et al. [2016]. By increasing the magnitude of the perturbation, adversarial images can be printed and fed back into cameras while remaining adversarial [Kurakin et al., 2017b]. Modern attacks are now capable of attacking both physical and electronic real-world systems, for example by placing carefully crafted invisible sticker films on camera lenses [Li

et al., 2019a] or synthesizing adversarial audio for black-box speech systems such as the Google Speech Recognition API [Abdullah et al., 2019]. Since real world perturbations need to be visible by cameras and other sensors, the size and scope of adversarial examples in the real world tends to be quite different from adversarial examples studied on image datasets like CIFAR10 and ImageNet, and this gap has yet to be addressed.

2.2 Robust optimization and adversarial defenses

In light of adversarial attacks and their ability to completely break deep classifiers, a great amount of work has looked towards mitigating or defending models against adversarial attacks, resulting in what is commonly referred to as adversarial defenses. This problem is fundamentally related to the field of robust optimization Ben-Tal et al. [2009], the task of solving an optimization problem where some of the problem data is unknown, but belong to a bounded set. Indeed, robust optimization techniques have been used in the context of linear machine learning models [Xu et al., 2009] to create classifiers that are robust to perturbations of the input.

To defend models from adversarial attacks, we want to learn a set of model weights which minimizes the worst case loss against an adversarial attack. Mathematically, this can be framed as the following robust optimization problem:

$$\min_{\theta} \max_{x' \in \Delta(x)} \ell(f_\theta(x'), y) \tag{2.7}$$

where we've simply taken the adversarial loss from the previous optimization problem of finding an adversarial example from Equation (2.1), and wrapped it within an outer minimization over the model parameters θ for a deep network f_θ. This connection from defending against adversarial examples to robust optimization was addressed in an early adversarial examples paper [Goodfellow et al., 2015], where it was noted that for linear models, robustness to adversarial examples can be achieved via an ℓ_1 norm penalty on the weights within the loss function.[1] Madry et al. [2017] revisited this connection to robust optimization, and noted that simply solving the (non-convex) min-max formulation of the robust optimization problem works very well in practice to find and then optimize against adversarial examples. The approach was motivated by the classical result known as Danskin's theorem [Danskin, 1966], which says that the gradient of a maximization problem is equal to the gradient of the objective evaluated at the optimum, though in this setting it may only be an approximate optimum.

Methods for solving this robust optimization problem can be categorized into one of two main categories: provable defenses, which minimize a guaranteed upper bound of the adversarial loss, and adversarial training, which minimizes a lower bound of the adversarial loss. After computing a bound, both of these categories of defenses then use standard backpropagation tools for deep learning to minimize the bound to learn networks robust to adversarial examples.

[1] This fact is well-known in robust optimization, and we merely mean that the original paper pointed out this connection.

2.2.1 Verification

The precursor to provable defenses was verification methods for deep networks, which tried to formally verify whether deep networks satisfy certain properties. There is a great deal of work using exact (combinatorial) solvers to verify properties of neural networks, including robustness to adversarial attacks. These typically employ either Satisfiability Modulo Theories (SMT) solvers [Carlini and Wagner, 2017b, Carlini et al., 2017, Ehlers, 2017, Huang et al., 2017, Katz et al., 2017] or integer programming approaches [Cheng et al., 2017, Lomuscio and Maganti, 2017, Tjeng and Tedrake, 2017]. The obvious advantage of these approaches is that they are able to reason *exactly* whether a property is satisfied or not. However, because they are fundamentally combinatorial in nature, they tend to be limited in practice to small, fully-connected networks with one or two layers, and struggle to verify even reasonably small convolutional networks, such as those used on the MNIST dataset [LeCun, 1998].

There is one notable exception: Tjeng et al. [2018] adapt some of the ideas presented in this dissertation to drastically prune the number of branches needed to solve a mixed integer linear program (MILP) for verifying adversarial robustness, which is able to verify some, but not all, small convolutional networks at a small radius. However, this scalability issue has thus far prevented these methods from effectively scaling to large models typically used in deep learning applications or being used within a training setting, as tying exact verification into a deep learning training loop is simply computationally infeasible at this point in time.

In order to scale beyond small networks, other work has looked to verify network properties using non-combinatorial methods by forgoing an exact certificate and instead certifying a looser bound instead, typically by overapproximation. For example, there is a line of work towards developing a suite of verification methods based upon abstract interpretations from programming languages, which can be broadly construed as relaxations of combinations of activations that are maintained as they pass through the network [Gehr et al., 2018]. This approach has been refined and scaled to larger, more general network architectures [Singh et al., 2018a], combined with MILP solvers to enhance the precision of the approximation [Singh et al., 2018b], and extended to geometric transformations [Balunovic et al., 2019] and generative models [Mirman et al., 2020].

Other optimization approaches can be leveraged to produce certified bounds. For example, Dvijotham et al. [2018b] solve an optimization problem resulting from dual functions of the activations to verify robustness to adversarial examples, which is most similar to the bounds presented in this dissertation. Semidefinite programming (SDP) relaxations can offer some of the tightest bounds which are solvable in polynomial time, but can only verify small fully connected networks [Raghunathan et al., 2018b]. The semidefinite programming approach can be further tightened by adding quadratic constraints [Fazlyab et al., 2019], trading increased complexity for a tighter upper bound (but still polynomial time). While these verification approaches trade off exact verification for computing a more scalable but looser bound which can be applid to larger networks, they are still too computationally expensive to be tied into the training procedure to be used to solve a robust optimization problem.

2.2.2 Provable defenses

A subset of verification methods are those which compute even looser but *tractable* bounds on properties of deep networks, typically to guarantee robustness of the network against an adversarial attack. The primary difference between these methods and the previous verification approaches is in the computation: these bounds can be typically computed in closed form without solving an optimization problem. As a result, these bounds can also be reasonably tied into the training procedure to learn a network which minimizes the bound to guarantee that no adversarial example exists, resulting in what we call a provable defense. Although they are efficient, the bounds are typically so loose that it is typical for the bound to be vacuous and not guarantee adversarial robustness at reasonably small thresholds, unless the network was specifically trained to minimize the bound.

One of the earliest defenses to guarantee robustness to adversarial examples was Parseval networks [Cisse et al., 2017], which regularize the ℓ_2 operator norm of the weight matrices in order to keep the Lipschitz constant of the network less than one. This guarantees that the network is non-expansive in the ℓ_2 norm resulting in a bound on the norm of the output, and can achieve some minor degree of adversarial robustness. Similar work showed how to limit the possible layerwise ℓ_2 norm expansions in a variety of different layer types in a modular fashion [Peck et al., 2017]. Although this work did not incorporate their bound into the training procedure, in hindsight it may have been more successful if it had been trained to minimize the bound. After all, later work which studied the ℓ_∞ analogue of this approach propagated interval bounds layer-by-layer through a deep network [Gowal et al., 2018]. Although the loose approximation is unsurprisingly vacuous on most networks, the interval bound can become reasonably tight when the bound is optimized, and achieve competitive levels of certified robustness when applied to large networks and tuned properly.

Other "layerwise" bounds have been developed which can be orders of magnitude tighter than the previously described bound propagation methods while still being tractable enough to be tied into training. One of the earliest works in this space provided an adversarial robustness guarantee for ℓ_2 perturbations in two-layer networks, and trained using a surrogate of the robust bound to get provable guarantees [Hein and Andriushchenko, 2017]. Later, the work of Raghunathan et al. [2018a] developed a dual SDP relaxation for ℓ_∞ robustness also in the two-layer network setting, which reduced the bound to an eigenvalue problem. Although these bounds were trainable, they were limited in scalability to the two-layer setting and fully connected networks.

Over time, however, a number of provable defenses were developed that were scalable and widely applicable to modern architectures. For example, the verification line of work based on abstract interpretations was adapted to faster, layer-wise abstractions which could then be used in training [Mirman et al., 2018]. The work in this dissertation took a different perspective, starting with a linear programming (LP) relaxation more similar to the SDP approach [Raghunathan et al., 2018a]. However, instead of solving the LP, we leveraged dual feasible solutions that could be constructed by propagating dual variables layer-wise forward and backward through the network [Wong and Kolter, 2017] to get a certified bound. By training on these dual feasible certificates as a provable defense, we were able to learn small convolutional networks that could be verified for the first time. The work in this dissertation also extends the dual LP to the general setting, making it applicable to arbitrary computational graphs and leveraging random projections to make the

bounds more tractable on medium-sized networks [Wong et al., 2018]. Later work found that the exact same verification algorithm obtained by dual feasible solutions of the linear program could be equivalently obtained by forward propagating bounds in a linearized version of the deep network [Weng et al., 2018, Zhang et al., 2018], where specific choices in dual feasible solutions for the linear program are equivalent to specific choices in linearizing the ReLU activations of the network. However, without training on the bound as done in this dissertation [Wong and Kolter, 2017, Wong et al., 2018], these bounds tend to produce vacuous, non-meaningful guarantees when used only as verifiers. Later analysis showed that the dual LP bound used in this dissertation, when trained as a provable defense, is tight when the LP is solved exactly [Salman et al., 2019b]. The bounds based on dual linear programs or linearized networks were later combined with interval bound propagation to slightly improve the final verified performance after training [Zhang et al., 2019c], and this remains one of the most competitive approaches for ℓ_∞ provable robustness.

Another distinct category of provable defenses are those which leverage randomized smoothing to generate probabilistic guarantees. Initially proposed from a differential privacy perspective [Lecuyer et al., 2019], randomized smoothing replaces the output of a classifier with its expected output under noise, in order to compute a probabilistic bound on the output of a network. These bounds were eventually tightened and combined with Gaussian data augmentation at high noise levels to produce state of the art certified results for robustness against ℓ_2 bounded noise [Cohen et al., 2019]. Later work further improved the approach by combining it with adversarial training methods [Salman et al., 2019a], and randomized smoothing can now be prepended to standard classifiers with no guarantees to add probabilistic guarantees to pretrained classifiers in a modular fashion [Salman et al., 2020]. While the approach has been generalized to other ℓ_p norms [Yang et al., 2020], theoretical analysis suggests that randomized smoothing may be unable to certify ℓ_∞ perturbations at a reasonably sized radius [Blum et al., 2020]. However, randomized smoothing tends to outperform other LP-based bounds in the ℓ_2 setting, and remains the most competitive approach for ℓ_2 provable robustness.

Several provable defenses do not fall into any of the previously described categories. There has been some work in studying distributional robustness, or minimizing the worst-case loss over the entire population [Sinha et al., 2018a]. Rather than constraining the ℓ_p norm of each example to generate adversarial examples, distributional robustness can be seen as constaining the total ℓ_p norm of perturbations for an entire population (or dataset) summed over each example. Sinha et al. [2018a] are able to produce a bound on the adversarial population loss and train to minimize the bound, which furthermore comes with generalization guarantees under proper assumptions. Another line of work has looked into analyzing the properties of robust networks, designing heuristics to encourage these properties, and formally verify their robustness using independent MILP solvers. These heuristics include encouraging weight sparsity and stability of ReLU activations [Xiao et al., 2019] as well as maximizing the linear regions of the network [Croce et al., 2018].

2.2.3 Adversarial training

While provable defenses provide strong guarantees on the performance of the network under adversarial perturbations, these guarantees come at a cost: it is common for provably robust

networks to achieve lower clean accuracy than their standard counterparts. At the same time, the bound may be too conservative, certifying a lower adversarial radius than is empirically possible. As a result, despite the advancement of provable defenses, a great deal of interest has looked at improving a more empirical defense known as adversarial training, which typically has better empirical performance (both clean and adversarial) but does not come with any formal guarantees.

At its core, adversarial training performs an adversarial attack to approximate the inner maximization to compute an adversarial loss, and performs backpropagation on the adversarial loss. In short, rather than minimizing an upper bound on the adversarial loss, adversarial training minimizes a lower bound bound in the form of an adversarial example. This was initially proposed for the FGSM adversary in the early days of adversarial examples as FGSM adversarial training [Goodfellow et al., 2015], but was found to be converging to a degenerate local minimum and was combined with an initial randomization step [Tramèr et al., 2017]. However even with this additional randomization, models trained with FGSM adversarial training at the time did not produce robustness to strong PGD attacks. The effectiveness of adversarial training was not recognized until it was combined with a PGD adversary [Madry et al., 2017], and FGSM adversarial training was dismissed as simply being too weak and a poor approximation of the adversarial loss. However, part of the work in this dissertation presents a surprising discovery that goes against what was previously believed: FGSM adversarial training with a better random intialization can in fact learn a robust network, and with proper tuning, can achieve results comparable to PGD adversarial training [Wong et al., 2020a].

Further incremental improvements to both the PGD adversary and the adversarial training procedure include incorporating momentum into the adversary [Dong et al., 2018], leveraging matrix estimation [Yang et al., 2019], logit pairing [Mosbach et al., 2018], and feature denoising [Xie et al., 2019]. However, all of these approaches rely on using adversarial training as the core defense, and are either not as effective or completely fail when used on their own. Zhang et al. [2019b] proposed a method called TRADES for adversarial training that performs adversarial training but balances between standard and robust errors, and for a long while achieved state-of-the-art robust performance on standard benchmarks in adversarial examples, improving upon the standard PGD adversarial training approach. However, in this dissertation we find that the gains in adversarial robustness from newer methods like TRADES are a product of early stopping and not algorithmic improvement due to the prevalance of overfitting in adversarial training [Rice et al., 2020]. Unfortunately, this suggests that there has been no algorithmic improvement in adversarially robust deep learning since PGD adversarial training, which has been confirmed by improved adversarial attacks [Croce and Hein, 2020].

On the other hand, there has been a growing body of evidence suggesting that adversarially robust training needs more data [Schmidt et al., 2018], and that adversarial training can hurt generalization [Raghunathan et al., 2019]. This matches empirical observations, where current datasets have larger generalization gaps when trained robustly with adversarial training. Additional data can greatly improve adversarial robustness when used with self-supervised learning techniques [Alayrac et al., 2019, Carmon et al., 2019, Zhai et al., 2019], which was one of the most significant improvements in adversarial robustness that actually improved upon vanilla PGD adversarial training.

Because PGD adversarial training is significantly more time consuming than standard train-

ing, several works have focused on improving the efficiency of adversarial training. For example, one can reduce the computational complexity of calculating gradients by caching gradients that remain the same across PGD iterations [Zhang et al., 2019a]. Other work has looked at reducing the number of attack iterations [Wang, 2018], proposing methods such as free adversarial training [Shafahi et al., 2019], and speeding up FGSM adversarial training with methods from fast standard training [Wong et al., 2020a], the last of which is discussed in this dissertation.

Separate works have also expanded the general PGD adversarial training algorithm to different threat models beyond the ℓ_p ball which capture perturbations beyond unstructured noise. These include various image transformations such as rotations and translations [Engstrom et al., 2017] or spatial flows [Xiao et al., 2018]. The work in this dissertation proposes a different threat model based on the Wasserstein distance for images [Wong et al., 2019] to leverage prior knowledge about the pixels in an image and more accurately capture small image transformations. Other work has looked at how PGD adversarial training can generalize to multiple threat models [Maini et al., 2019, Tramèr and Boneh, 2019], which is also discussed in this dissertation.

2.2.4 Other defenses

The previous sections focused on provable defenses and adversarial training, as these have been the most successful methods for mitigating adversarial examples that haven't been broken by stronger attacks. However, there is a long and complicated history of numerous other heuristic defenses which were proposed to provide robustness to adversarial examples, but were proven to be ultimately ineffective when evaluated against a stronger adversary.

For example, one of the earliest proposed methods for mitigating adversarial examples was called defensive distillation, which uses a temperature variable to control the magnitude of the network softmax values, and was initially though to be effective at preventing early adversarial attacks [Papernot et al., 2016b], until stronger versions of these attacks were able to break networks with defensive distillation [Carlini and Wagner, 2017b]. Other work argued that, under "realistic" settings of rotation and scaling, adversarial examples were nothing to worry about [Lu et al., 2017] until adversarial examples were crafted to also be robust to these sorts of transformations [Athalye and Sutskever, 2017]. Rather than defending a specific network, other work tried to instead simply detect whether an example was adversarial or not using small detector networks [Metzen et al., 2017], identifying adversarial artifacts with Bayesian uncertainty estimates [Feinman et al., 2017], or leveraging interpretable attributes to identify adversarial examples [Tao et al., 2018]. However, all of these detection methods (and many more) were ultimately shown to be ineffective and bypassable [Carlini, 2019, Carlini and Wagner, 2017a]. Indeed, defenses with optimistic evaluations like thermometer encoding [Buckman et al., 2018], data purifiers to remove adversarial perturbations [Song et al., 2017], and input transformations to destroy adversarial perturbations [Guo et al., 2017] were being released so rapidly that papers started to break multiple models en masse [Athalye et al., 2018a, Uesato et al., 2018], and contests at conferences like the NIPS 2017 adversarial examples challenge were organized to pit attackers against defenders to identify the real progress made on developing empirically robust deep classifiers [Kurakin et al., 2018].

This back-and-forth where heuristic defenses are constantly being defeated by stronger attacks highlights the imbalance in difficulty between adversarial attacks and defenses. For an

adversary to "win", it does not have to find the optimal adversarial example which incurs the maximum loss. Instead, it is sufficient for the adversary to just find some example within the threat model which is incorrectly classified, whereas a defender must ensure that all points within the threat model are correctly classified. Unfortunately, the incentives are also misaligned: adversarial defenses are not motivated to perform proper evaluations with strong adversaries, as stronger adversaries will make the defense appear less effective. This has resulted in the establishment of a set of community guidelines for properly evaluating adversarial defenses [Carlini et al., 2019], which includes peforming an adaptive attack against proposed adversarial defenses. While this has improved the situation to some degree, many heuristic defenses still only use incomplete adaptive attacks, as a significant number of heuristic defenses published at top machine learning conferences continue to be circumvented with improved adaptive attacks [Tramer et al., 2020].

Chapter 3

Provable defenses

One way to truly harden classifiers against adversarial attacks is to design classifiers that are *guaranteed* to be robust to adversarial perturbations, even if the attacker is given full knowledge of the classifier. This has the advantage of not relying on "security through obscurity" and will be robust regardless of the strength of the adversary.

In this chapter, we present a method for training *provably robust* deep ReLU classifiers, classifiers that are guaranteed to be robust against any norm-bounded adversarial perturbations on the training set. The approach also provides a provable method for detecting any previously unseen adversarial example, with zero false negatives (i.e., the system will flag any adversarial example in the test set, though it may also mistakenly flag some non-adversarial examples). The crux of our approach is to construct a *convex outer bound* on the so-called "adversarial polytope", the set of all final-layer activations that can be achieved by applying a norm-bounded perturbation to the input; if we can guarantee that the class prediction of an example does not change within this outer bound, we have a proof that the example could not be adversarial (because the nature of an adversarial example is such that a small perturbation changed the class label). The convex outer bound we use in this work leverages the linear ReLU relaxations employed by the PLANET solver [Ehlers, 2017], which used a similar type of relaxation in a larger combinatorial solver.

We show how we can efficiently compute and optimize over the "worst case loss" within this convex outer bound, even in the case of deep networks that include relatively large (for verified networks) convolutional layers, and thus learn classifiers that are provably robust to such perturbations. From a technical standpoint, the outer bounds we consider involve a large linear program, but we show how to bound these optimization problems using a formulation that computes a feasible dual solution to this linear program using just a single backward pass through the network (and avoiding any actual linear programming solvers). On the one hand, the method overcomes the combinatorial computational barrier for exact verification methods for adversarial examples based on SMT [Carlini and Wagner, 2017b, Carlini et al., 2017, Ehlers, 2017, Huang et al., 2017, Katz et al., 2017] or MILP solvers [Cheng et al., 2017, Lomuscio and Maganti, 2017, Tjeng and Tedrake, 2017], as well as the polynomial computational barrier from SDP solvers [Raghunathan et al., 2018a], all of which cannot scale to even the medium-sized networks that we study here, let alone be tied into the training procedure. On the other hand, the bounds produced by the method are significantly tighter than other layer-wise bounds [Cisse et al., 2017, Peck et al., 2017], often by many orders of magnitude.

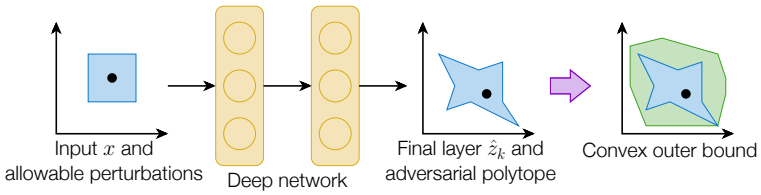


Figure 3.1: Conceptual illustration of the (non-convex) adversarial polytope, and an outer convex bound.

In a later section, we make substantial progress towards scaling this approach for learning provably robust networks to realistic sizes in three key ways. First, we extend the framework to deal with abstract computational graphs, including residual/skip connections (a hallmark of modern deep network architectures) and arbitrary activation functions. Second, note that the original approach scales *quadratically* in the number of hidden units in the network, making it impractical for larger networks. To scale, we use a nonlinear random projection technique to estimate the bound in a manner that scales only linearly in the size of the hidden units (i.e., only a constant multiple times the cost of traditional training), and which empirically can be used to train the networks with no degradation in performance from the previous work. Third, we show how to further improve robust performance of these methods, though at the expense of worse non-robust error, using multi-stage cascade models. Through these extensions, we are able to improve substantially upon the verified robust errors.

3.1 Training provably robust classifiers

This section contains the main methodological contribution towards provable defenses: a method for training deep ReLU networks that are provably robust to norm-bounded perturbations. Our derivation roughly follows three steps: first, we define the adversarial polytope for deep ReLU networks, and present our convex outer bound; second, we show how we can efficiently optimize over this bound by considering the *dual problem* of the associated linear program, and illustrate how to find solutions to this dual problem using a single modified backward pass in the original network; third, we show how to incrementally compute the necessary elementwise upper and lower activation bounds, using this dual approach. After presenting this algorithm, we then summarize how the method is applied to train provably robust classifiers, and how it can be used to detect potential adversarial attacks on previously unseen examples.

3.1.1 Outer bounds on the adversarial polytope

In this section, we consider a k layer feedforward ReLU-based neural network, $f_\theta : \mathbb{R}^{|x|} \rightarrow \mathbb{R}^{|y|}$ given by the equations

$$\begin{aligned} \hat{z}_{i+1} &= W_i z_i + b_i, \quad \text{for } i = 1, \ldots, k-1 \\ z_i &= \max\{\hat{z}_i, 0\}, \quad \text{for } i = 2, \ldots, k-1 \end{aligned} \tag{3.1}$$

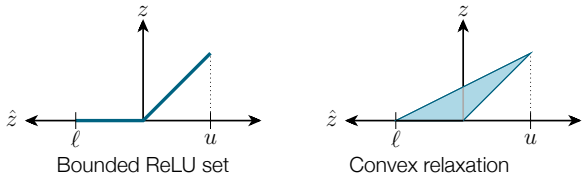


Figure 3.2: Illustration of the convex ReLU relaxation over the bounded set $[\ell, u]$.

with $z_1 \equiv x$ and $f_\theta(x) \equiv \hat{z}_k$ (the logits input to the classifier). We use $\theta = \{W_i, b_i\}_{i=1,\dots,k}$ to denote the set of all parameters of the network, where W_i represents a linear operator such as matrix multiply or convolution.

We use the set $\mathcal{Z}_\epsilon(x)$ to denote the adversarial polytope, or the set of all final-layer activations attainable by perturbing x by some δ with ℓ_∞ norm bounded by ϵ:[1]

$$\mathcal{Z}_\epsilon(x) = \{f_\theta(x+\delta) : \|\delta\|_\infty \leq \epsilon\}. \tag{3.2}$$

For multi-layer networks, $\mathcal{Z}_\epsilon(x)$ is a non-convex set (it can be represented exactly via an integer program as in [Lomuscio and Maganti, 2017] or via SMT constraints [Katz et al., 2017]), so cannot easily be optimized over. Then, we can rewrite the problem of finding an adversarial example as a maximization of some loss ℓ over the adversarial polytope:

$$\begin{aligned} &\underset{z}{\text{maximize}} \;\; \ell(z, y) \\ &\text{subject to} \;\; z \in \mathcal{Z}_\epsilon(x) \end{aligned} \tag{3.3}$$

The foundation of our approach will be to construct a *convex outer bound* on this adversarial polytope, as illustrated in Figure 3.1. If no point within this outer approximation exists that will change the class prediction of an example, then we are also guaranteed that no point within the true adversarial polytope can change its prediction either, i.e., the point is robust to adversarial attacks. Our eventual approach will be to train a network to optimize the *worst* case loss over this convex outer bound, effectively applying robust optimization techniques despite non-linearity of the classifier.

The starting point of our convex outer bound is a linear relaxation of the ReLU activations. Specifically, given known lower and upper bounds ℓ, u for the pre-ReLU activations, we can replace the ReLU equalities $z = \max\{0, \hat{z}\}$ from (3.1) with their upper convex envelopes,

$$z \geq 0, \;\; z \geq \hat{z}, \;\; -u\hat{z} + (u-\ell)z \leq -u\ell. \tag{3.4}$$

The procedure is illustrated in Figure 3.2, and we note that if ℓ and u are both positive or both negative, the relaxation is exact. The same relaxation at the activation level was used in Ehlers [2017], however as a sub-step for exact (combinatorial) verification of networks, and the method for actually computing the crucial bounds ℓ and u is different. We denote this outer bound on the adversarial polytope from replacing the ReLU constraints described in Equation (3.1) with the three linear constraints from Equation (3.4) as $\tilde{\mathcal{Z}}_\epsilon(x)$.

[1]For the sake of concreteness, we will focus on the ℓ_∞ bound during this exposition, but the method does extend to other norm balls, which we will highlight shortly.

Robustness guarantees via the convex outer adversarial polytope. We can use this outer bound to provide provable guarantees on the adversarial robustness of a classifier. Given a sample x with known label $y^\star$, we can find the point in $\tilde{\mathcal{Z}}_\epsilon(x)$ that minimizes this class and maximizes some alternative target y^{targ}, by solving the optimization problem

$$\begin{aligned} &\underset{\hat{z}_k}{\text{minimize}} \;\; (\hat{z}_k)_{y^\star} - (\hat{z}_k)_{y^{\text{targ}}} \equiv c^T \hat{z}_k \\ &\text{subject to} \;\; \hat{z}_k \in \tilde{\mathcal{Z}}_\epsilon(x) \end{aligned} \tag{3.5}$$

where $c \equiv e_{y^\star} - e_{y^{\text{targ}}}$ instantiates the loss for a targeted adversarial attack. Importantly, this is a *linear program* (LP): the objective is linear in the decision variables, and our convex outer approximation consists of just linear equalities and inequalities, which is more obvious after expanding $\tilde{\mathcal{Z}}_\epsilon(x)$ as seen in Equation (3.6).

$$\begin{aligned} &\underset{\hat{z}_k}{\text{minimize}}\, c^T \hat{z}_k, \;\; \text{subject to} \\ &\hat{z}_{i+1} = W_i z_i + b_i, \; i = 1, \ldots, k-1 \\ &z_1 \leq x + \epsilon \\ &z_1 \geq x - \epsilon \\ &z_{i,j} = 0, \; i = 2, \ldots, k-1, j \in \mathcal{I}_i^- \\ &z_{i,j} = \hat{z}_{i,j}, \; i = 2, \ldots, k-1, j \in \mathcal{I}_i^+ \\ &\left.\begin{aligned} &z_{i,j} \geq 0, \\ &z_{i,j} \geq \hat{z}_{i,j}, \\ &(u_{i,j} - \ell_{i,j}) z_{i,j} - u_{i,j} \hat{z}_{i,j} \leq -u_{i,j} \ell_{i,j} \end{aligned}\right\} \; i = 2, \ldots, k-1, j \in \mathcal{I}_i \end{aligned} \tag{3.6}$$

Crucially, *if we solve this LP for all target classes $y^{\text{targ}} \neq y^\star$ and find that the objective value in all cases is* positive *(i.e., we cannot make the true class activation lower than the target even in the outer polytope), then we know that* no *norm-bounded adversarial perturbation of the input could misclassify the example.*

We can conduct similar analysis on test examples as well. If the network predicts some class $\hat{y}$ on an example x, then we can use the same procedure as above to test whether the network will output any *different* class for a norm-bounded perturbation. If not, then the example *cannot* be adversarial, because no input within the norm ball takes on a different class (although of course, the network could still be predicting the wrong class). Although this procedure may incorrectly "flag" some non-adversarial examples, it will have zero false negatives, e.g., there may be a normal example that can still be classified differently due to a norm-bounded perturbation, but all norm-bounded adversarial examples will be detected.

Of course, two major issues remain: 1) although the LP formulation can be solved "efficiently", actually solving an LP via traditional methods for each example, for each target class, is not tractable; 2) we need a way of computing the crucial ℓ and u bounds for the linear relaxation. We address these in the following two sections.

3.1.2 Efficient optimization via the dual network

Because solving an LP with a number of variables equal to the number of activations in the deep network via standard approaches is not practically feasible, the key aspect of our approach lies in our method for very efficiently bounding these solutions. Specifically, we consider the *dual problem* of the LP above; recall that any feasible dual solution provides a guaranteed lower bound on the solution of the primal. *Crucially, we show that the feasible set of the dual problem can itself be expressed as a deep network, and one that is very similar to the standard backprop network*. This means that providing a provable lower bound on the primal LP (and hence also a provable bound on the adversarial error), can be done with *only a single backward pass through a slightly modified network* (assuming for the time being, that we still have known upper and lower bounds for each activation). This is expressed in the following theorem

Theorem 1. *The dual of* (3.5) *is of the form*

$$\begin{aligned} \underset{\alpha}{\text{maximize}} \;\; & J_\epsilon(x, g_\theta(c, \alpha)) \\ \text{subject to} \;\; & \alpha_{i,j} \in [0,1], \; \forall i, j \end{aligned} \tag{3.7}$$

where $J_\epsilon(x, \nu)$ *is equal to*

$$-\sum_{i=1}^{k-1} \nu_{i+1}^T b_i - x^T \hat{\nu}_1 - \epsilon \|\hat{\nu}_1\|_1 + \sum_{i=2}^{k-1} \sum_{j \in \mathcal{I}_i} \ell_{i,j} [\nu_{i,j}]_+ \tag{3.8}$$

and $g_\theta(c, \alpha)$ *is a* k *layer feedforward neural network given by the equations*

$$\begin{aligned} \nu_k &= -c \\ \hat{\nu}_i &= W_i^T \nu_{i+1}, \quad \text{for } i = k-1, \ldots, 1 \\ \nu_{i,j} &= \begin{cases} 0 & j \in \mathcal{I}_i^- \\ \hat{\nu}_{i,j} & j \in \mathcal{I}_i^+ \\ \frac{u_{i,j}}{u_{i,j} - \ell_{i,j}} [\hat{\nu}_{i,j}]_+ - \alpha_{i,j} [\hat{\nu}_{i,j}]_- & j \in \mathcal{I}_i, \end{cases} \\ & \qquad\qquad \text{for } i = k-1, \ldots, 2 \end{aligned} \tag{3.9}$$

where ν *is shorthand for* $(\nu_i, \hat{\nu}_i)$ *for all* i *(needed because the objective* J *depends on* all ν *terms, not just the first), and where* $\mathcal{I}_i^-$*,* $\mathcal{I}_i^+$*, and* $\mathcal{I}_i$ *denote the sets of activations in layer* i *where the lower and upper bounds are both negative, both positive, or span zero respectively.*

Proof. In detail, we associate the following dual variables with each of the constraints

$$\begin{aligned} \hat{z}_{i+1} = W_i z_i + b_i &\Rightarrow \nu_{i+1} \in \mathbb{R}^{|\hat{z}_{i+1}|} \\ z_1 \leq x + \epsilon &\Rightarrow \xi^+ \in \mathbb{R}^{|x|} \\ -z_1 \leq -x + \epsilon &\Rightarrow \xi^- \in \mathbb{R}^{|x|} \\ -z_{i,j} \leq 0 &\Rightarrow \mu_{i,j} \in \mathbb{R} \\ \hat{z}_{i,j} - z_{i,j} \leq 0 &\Rightarrow \tau_{i,j} \in \mathbb{R} \\ -u_{i,j} \hat{z}_{i,j} + (u_{i,j} - \ell_{i,j}) z_{i,j} \leq -u_{i,j} \ell_{i,j} &\Rightarrow \lambda_{i,j} \in \mathbb{R} \end{aligned} \tag{3.10}$$

where we note that can easily eliminate the dual variables corresponding to the $z_{i,j} = 0$ and $z_{i,j} = \hat{z}_{i,j}$ from the optimization problem, so we don't define explicit dual variables for these; we also note that $\mu_{i,j}$, $\tau_{i,j}$, and $\lambda_{i,j}$ are only defined for i, j such that $j \in \mathcal{I}_i$, but we keep the notation as above for simplicity. With these definitions, the dual problem becomes

$$\begin{aligned}
\text{maximize} \quad & -(x+\epsilon)^T\xi^+ + (x-\epsilon)^T\xi^- - \sum_{i=1}^{k-1} \nu_{i+1}^T b_i + \sum_{i=2}^{k-1} \lambda_i^T (u_i \ell_i) \\
\text{subject to} \quad & \\
& \nu_k = -c \\
& \nu_{i,j} = 0, \quad \text{for } j \in \mathcal{I}_i^-, \quad i = 2, \ldots, k-1 \\
& \nu_{i,j} = (W_i^T \nu_{i+1})_j \quad \text{for } j \in \mathcal{I}_i^+, \quad i = 2, \ldots, k-1 \\
& (u_{i,j} - \ell_{i,j})\lambda_{i,j} - \mu_{i,j} - \tau_{i,j} = (W_i^T \nu_{i+1})_j \quad \text{for } j \in \mathcal{I}_i, \quad i = 2, \ldots, k-1 \\
& \nu_{i,j} = u_{i,j}\lambda_{i,j} - \mu_i \quad \text{for } j \in \mathcal{I}_i, \quad i = 2, \ldots, k-1 \\
& W_1^T \nu_2 = \xi^+ - \xi^- \\
& \lambda, \tau, \mu, \xi^+, \xi^- \geq 0
\end{aligned} \tag{3.11}$$

The key insight we highlight here is that *the dual problem can also be written in the form of a deep network*, which provides a trivial way to find feasible solutions to the dual problem, which can then be optimized over. Specifically, consider the constraints

$$\begin{aligned}
(u_{i,j} - \ell_{i,j})\lambda_{i,j} - \mu_{i,j} - \tau_{i,j} &= (W_i^T \nu_{i+1})_j \\
\nu_{i,j} &= u_{i,j}\lambda_{i,j} - \mu_i.
\end{aligned} \tag{3.12}$$

Note that the dual variable λ corresponds to the upper bounds in the convex ReLU relaxation, while μ and τ correspond to the lower bounds $z \geq 0$ and $z \geq \hat{z}$ respectively; by the complementarity property, we know that at the optimal solution, these variables will be zero if the ReLU constraint is non-tight, or non-zero if the ReLU constraint is tight. Because we cannot have the upper and lower bounds be simultaneously tight (this would imply that the ReLU input $\hat{z}$ would exceed its upper or lower bound otherwise), we know that either λ or $\mu + \tau$ must be zero. This means that at the optimal solution to the dual problem

$$\begin{aligned}
(u_{i,j} - \ell_{i,j})\lambda_{i,j} &= [(W_i^T \nu_{i+1})_j]_+ \\
\tau_{i,j} + \mu_{i,j} &= [(W_i^T \nu_{i+1})_j]_-
\end{aligned} \tag{3.13}$$

i.e., the dual variables capture the positive and negative portions of $(W_i^T \nu_{i+1})_j$ respectively. Combining this with the constraint that

$$\nu_{i,j} = u_{i,j}\lambda_{i,j} - \mu_i \tag{3.14}$$

means that

$$\nu_{i,j} = \frac{u_{i,j}}{u_{i,j} - \ell_{i,j}} [(W_i^T \nu_{i+1})_j]_+ - \alpha [(W_i^T \nu_{i+1})_j]_- \tag{3.15}$$

for $j \in \mathcal{I}_i$ and for some $0 \leq \alpha \leq 1$ (this accounts for the fact that we can either put the "weight" of $[(W_i^T \nu_{i+1})_j]_-$ into μ or τ, which will or will not be passed to the next ν_i). This is exactly a type of leaky ReLU operation, with a slope in the positive portion of $u_{i,j}/(u_{i,j} - \ell_{i,j})$ (a term between 0 and 1), and a negative slope anywhere between 0 and 1. Similarly, and more simply, note that ξ^+ and ξ^- denote the positive and negative portions of $W_1^T \nu_2$, so we can replace these terms with an absolute value in the objective. Finally, we note that although it is possible to have $\mu_{i,j} > 0$ and $\tau_{i,j} > 0$ simultaneously, this corresponds to an activation that is identically zero pre-ReLU (both constraints being tight), and so is expected to be relatively rare. Putting this all together, and using $\hat{\nu}$ to denote "pre-activation" variables in the dual network, we can write the dual problem in terms of the network

$$\begin{aligned} \nu_k &= -c \\ \hat{\nu}_i &= W_i^T \nu_{i+1}, i = k-1, \ldots, 1 \\ \nu_{i,j} &= \begin{cases} 0 & j \in \mathcal{I}_i^- \\ \hat{\nu}_{i,j} & j \in \mathcal{I}_i^+ \\ \frac{u_{i,j}}{u_{i,j} - \ell_{i,j}} [\hat{\nu}_{i,j}]_+ - \alpha_{i,j} [\hat{\nu}_{i,j}]_- & j \in \mathcal{I}_i, \end{cases} \\ & \qquad \text{for } i = k-1, \ldots, 2 \end{aligned} \tag{3.16}$$

which we will abbreviate as $\nu = g_\theta(c, \alpha)$ to emphasize the fact that $-c$ acts as the "input" to the network and α are per-layer inputs we can also specify (for only those activations in $\mathcal{I}_i$), where ν in this case is shorthand for all the ν_i and $\hat{\nu}_i$ activations.

The final objective we are seeking to optimize can also be written

$$\begin{aligned} J_\epsilon(x, \nu) &= -\sum_{i=1}^{k-1} \nu_{i+1}^T b_i - (x + \epsilon)^T [\hat{\nu}_1]_+ + (x - \epsilon)^T [\hat{\nu}_1]_- \\ &\quad + \sum_{i=2}^{k-1} \sum_{j \in \mathcal{I}_i} \frac{u_{i,j} \ell_{i,j}}{u_{i,j} - \ell_{i,j}} [\hat{\nu}_{i,j}]_+ \\ &= -\sum_{i=1}^{k-1} \nu_{i+1}^T b_i - x^T \hat{\nu}_1 - \epsilon \|\hat{\nu}_1\|_1 \\ &\quad + \sum_{i=2}^{k-1} \sum_{j \in \mathcal{I}_i} \ell_{i,j} [\nu_{i,j}]_+ \end{aligned} \tag{3.17}$$

□

The "dual network" from (3.9) in fact is almost identical to the backpropagation network, except that for nodes j in $\mathcal{I}_i$ there is the additional free variable $\alpha_{i,j}$ that we can optimize over to improve the objective. In practice, rather than optimizing explicitly over α, we choose the fixed, dual feasible solution

$$\alpha_{i,j} = \frac{u_{i,j}}{u_{i,j} - \ell_{i,j}}. \tag{3.18}$$

This makes the entire backward pass a *linear* function, and is additionally justified by considerations regarding the conjugate set of the ReLU relaxation as the largest choice of α which does

Algorithm 1 Computing Activation Bounds

input: Network parameters $\{W_i, b_i\}_{i=1}^{k-1}$, data point x, ball size ϵ
// initialization
$\hat{\nu}_1 := W_1^T$
$\gamma_1 := b_1^T$
$\ell_2 := x^T W_1^T + b_1^T - \epsilon \|W_1^T\|_{1,:}$
$u_2 := x^T W_1^T + b_1^T + \epsilon \|W_1^T\|_{1,:}$
// $\|\cdot\|_{1,:}$ for a matrix here denotes ℓ_1 norm of all columns
for $i = 2, \ldots, k-1$ **do**
  form $\mathcal{I}_i^-, \mathcal{I}_i^+, \mathcal{I}_i$; form D_i as in (3.20)
  // initialize new terms
  $\nu_{i,\mathcal{I}_i} := (D_i)_{\mathcal{I}_i} W_i^T$
  $\gamma_i := b_i^T$
  // propagate existing terms
  $\nu_{j,\mathcal{I}_j} := \nu_{j,\mathcal{I}_j} D_i W_i^T, \quad j = 2, \ldots, i-1$
  $\gamma_j := \gamma_j D_i W_i^T, \quad j = 1, \ldots, i-1$
  $\hat{\nu}_1 := \hat{\nu}_1 D_i W_i^T$
  // compute bounds
  $\psi_i := x^T \hat{\nu}_1 + \sum_{j=1}^{i} \gamma_j$
  $\ell_{i+1} := \psi_i - \epsilon \|\hat{\nu}_1\|_{1,:} + \sum_{j=2}^{i} \sum_{i' \in \mathcal{I}_i} \ell_{j,i'} [-\nu_{j,i'}]_+$
  $u_{i+1} := \psi_i + \epsilon \|\hat{\nu}_1\|_{1,:} - \sum_{j=2}^{i} \sum_{i' \in \mathcal{I}_i} \ell_{j,i'} [\nu_{j,i'}]_+$
end for
output: bounds $\{\ell_i, u_i\}_{i=2}^{k}$

not increase the bound, which is discussed later in Section 3.4.4. Because *any* solution α is still dual feasible, this still provides a lower bound on the primal objective, and one that is reasonably tight in practice.[2] Thus, in the remainder of this work we simply refer to the dual objective as $J(x, g_\theta(c))$, implicitly using the above-defined α terms.

We also note that norm bounds other than the ℓ_∞ norm are also possible in this framework: if the input perturbation is bounded within some convex ℓ_p norm, then the only difference in the dual formulation is that the ℓ_1 norm on $\|\hat{\nu}\|_1$ changes to $\|\hat{\nu}\|_q$ where q is the dual norm of p. This becomes more clear in the later section, when we use duality to bound arbitrary computational graphs.

3.1.3 Computing activation bounds

Thus far, we have ignored the (critical) issue of how we actually obtain the elementwise lower and upper bounds on the pre-ReLU activations, ℓ and u. Intuitively, if these bounds are too loose, then the adversary has too much "freedom" in crafting adversarial activations in the later layers that don't correspond to any actual input. However, because the dual function $J_\epsilon(x, g_\theta(c))$ provides a bound on *any* linear function $c^T \hat{z}_k$ of the final-layer coefficients, we can compute J

[2] The tightness of the bound is examined in Section 3.3.1.

for $c = I$ and $c = -I$ to obtain lower and upper bounds on these coefficients. For $c = I$, the backward pass variables (where $\hat{\nu}_i$ is now a matrix) are given by

$$\begin{aligned} \hat{\nu}_i &= -W_i^T D_{i+1} W_{i+1}^T \dots D_n W_n^T \\ \nu_i &= D_i \hat{\nu}_i \end{aligned} \tag{3.19}$$

where D_i is a diagonal matrix with entries

$$(D_i)_{jj} = \begin{cases} 0 & j \in \mathcal{I}_i^- \\ 1 & j \in \mathcal{I}_i^+ \\ \frac{u_{i,j}}{u_{i,j} - \ell_{i,j}} & j \in \mathcal{I}_i \end{cases}. \tag{3.20}$$

We can compute $(\nu_i, \hat{\nu}_i)$ and the corresponding upper bound $J_\epsilon(x, \nu)$ (which is now a vector) in a layer-by-layer fashion, first generating bounds on $\hat{z}_2$, then using these to generate bounds on $\hat{z}_3$, etc.

The resulting algorithm, which uses these backward pass variables in matrix form to incrementally build the bounds, is described in Algorithm 1. From here on, the computation of J will implicitly assume that we also compute the bounds. Because the full algorithm is somewhat involved, we highlight that there are two dominating costs to the full bound computation: 1) computing a forward pass through the network on an "identity matrix" (i.e., a basis vector e_i for each dimension i of the input); and 2) computing a forward pass starting at an intermediate layer, once for each activation in the set $\mathcal{I}_i$ (i.e., for each activation where the upper and lower bounds span zero). Direct computation of the bounds requires computing these forward passes explicitly, since they ultimately factor into the nonlinear terms in the J objective, and this is admittedly the poorest-scaling aspect of our approach. In a later section, we show how to use random Cauchy projections to estimate these terms, allowing us to scale the method to even larger models. However, even without improving scalability, the technique already can be applied to much larger networks than typical networks which are verifiable by exact solvers.

3.1.4 Efficient robust optimization

Using the lower bounds developed in the previous sections, we can develop an efficient optimization approach to training provably robust deep networks. Given a data set $(x_i, y_i)_{i=1,\dots,N}$, instead of minimizing the loss at these data points, we minimize (our bound on) the *worst* location (i.e. with the highest loss) in an ϵ ball around each x_i, i.e.,

$$\underset{\theta}{\text{minimize}} \quad \sum_{i=1}^{N} \max_{\|\Delta\|_\infty \leq \epsilon} L(f_\theta(x_i + \Delta), y_i). \tag{3.21}$$

This is a standard robust optimization objective, but prior to this work it was not known how to train these classifiers when f is a deep nonlinear network.

We also require that a multi-class loss function have the following property (all of cross-entropy, hinge loss, and zero-one loss have this property):

Property 1. *A multi-class loss function $L : \mathbb{R}^{|y|} \times \mathbb{R}^{|y|} \to \mathbb{R}$ is translationally invariant if for all $a \in \mathbb{R}$,*

$$L(y, y^\star) = L(y - a1, y^\star). \tag{3.22}$$

Under this assumption, we can upper bound the loss of the robust optimization problem with the loss evaluated on our bound from the dual problem as described in Theorem 2.

Theorem 2. *Let L be a monotonic loss function that satisfies Property 1. For any data point (x, y), and $\epsilon > 0$, the worst case adversarial loss from (3.21) can be upper bounded by*

$$\max_{\|\Delta\|_\infty \leq \epsilon} L(f_\theta(x+\Delta), y) \leq L(-J_\epsilon(x, g_\theta(e_y 1^T - I)), y), \tag{3.23}$$

where J_ϵ is vector valued and as defined in (3.8) for a given ϵ, and g_θ is as defined in (3.9) for the given model parameters θ.

Proof. First, we rewrite the problem using the adversarial polytope $\mathcal{Z}_\epsilon(x)$.

$$\max_{\|\Delta\|_\infty \leq \epsilon} L(f_\theta(x+\Delta), y) = \max_{\hat{z}_k \in \mathcal{Z}_\epsilon(x)} L(\hat{z}_k, y)$$

Since $L(x, y) \leq L(x - a1, y)$ for all a, we have

$$\begin{aligned} \max_{\hat{z}_k \in \mathcal{Z}_\epsilon(x)} L(\hat{z}_k, y) &\leq \max_{\hat{z}_k \in \mathcal{Z}_\epsilon(x)} L(\hat{z}_k - (\hat{z}_k)_y 1, y) \\ &= \max_{\hat{z}_k \in \mathcal{Z}_\epsilon(x)} L((I - \mathbf{e}_y 1^T)\hat{z}_k, y) \\ &= \max_{\hat{z}_k \in \mathcal{Z}_\epsilon(x)} L(C\hat{z}_k, y) \end{aligned} \tag{3.24}$$

where $C = (I - \mathbf{e}_y 1^T)$. Since L is a monotone loss function, we can upper bound this further by using the element-wise maximum over $[C\hat{z}_k]_i$ for $i \neq y$, and elementwise-minimum for $i = y$ (note, however, that for $i = y$, $[C\hat{z}_k]_i = 0$). Specifically, we bound it as

$$\max_{\hat{z}_k \in \mathcal{Z}_\epsilon(x)} L(C\hat{z}_k, y) \leq L(h(\hat{z}_k))$$

where, if C_i is the ith row of C, $h(z_k)$ is defined element-wise as

$$h(z_k)_i = \max_{\hat{z}_k \in \mathcal{Z}_\epsilon(x)} C_i \hat{z}_k$$

This is exactly the adversarial problem from (3.2) (in its maximization form instead of a minimization). Recall that J from (3.8) is a lower bound on (3.2) (using $c = -C_i$).

$$J_\epsilon(x, g_\theta(-C_i)) \leq \min_{\hat{z}_k \in \mathcal{Z}_\epsilon(x)} -C_i^T \hat{z}_k \tag{3.25}$$

Multiplying both sides by -1 gives us the following upper bound

$$-J_\epsilon(x, g_\theta(-C_i)) \geq \max_{\hat{z}_k \in \mathcal{Z}_\epsilon(x)} C_i^T \hat{z}_k$$

Applying this upper bound to $h(z_k)_i$, we conclude

$$h(z_k)_i \leq -J_\epsilon(x, g_\theta(-C_i))$$

Applying this to all elements of h gives the final upper bound on the adversarial loss.

$$\max_{\|\Delta\|_\infty \leq \epsilon} L(f_\theta(x+\Delta), y) \leq L(-J_\epsilon(x, g_\theta(\mathbf{e}_y 1^T - I)), y)$$

□

We denote the upper bound from Theorem 2 as the robust loss. Replacing the summand of (3.21) with the robust loss results in the following minimization problem

$$\underset{\theta}{\text{minimize}} \quad \sum_{i=1}^{N} L(-J_\epsilon(x_i, g_\theta(e_{y_i} 1^T - I)), y_i). \tag{3.26}$$

All the network terms, including the upper and lower bound computation, are differentiable, so the whole optimization can be solved with any standard stochastic gradient variant and autodiff toolkit, and the result is a network that (if we achieve low loss) is guaranteed to be robust to adversarial examples.

3.1.5 Adversarial guarantees

Although we previously described, informally, the guarantees provided by our bound, we now state them formally. The bound for the robust optimization procedure gives rise to several *provable* metrics measuring robustness and detection of adversarial attacks, which can be computed for any ReLU based neural network independently from how the network was trained; however, not surprisingly, the bounds are by far the tightest and the most useful in cases where the network was trained explicitly to minimize a robust loss.

Robust error bounds The upper bound from Theorem 2 functions as a certificate that guarantees robustness around an example (if classified correctly), as described in Corollary 1.

Corollary 1. *For a data point x, label $y^\star$ and $\epsilon > 0$, if*

$$\min_{y \neq f(x)} [J_\epsilon(x, g_\theta(\mathbf{e}_{f(x)} 1^T - I, \alpha))]_y \geq 0 \tag{3.27}$$

then the model is guaranteed to be robust around this data point. Specifically, there does not exist an adversarial example $\tilde{x}$ such that $\|\tilde{x} - x\|_\infty \leq \epsilon$ and $f_\theta(\tilde{x}) \neq y^\star$.

Proof. Recall that J from (3.8) is a lower bound on (3.2). Combining this fact with the certificate in (3.27), we get that for all $y \neq f(x)$,

$$\min_{\hat{z}_k \in \mathcal{Z}_\epsilon(x)} (\hat{z}_k)_{f(x)} - (\hat{z}_k)_y \geq 0$$

Crucially, this means that for every point in the adversarial polytope and for any alternative label y, $(\hat{z}_k)_{f(x)} \geq (\hat{z}_k)_y$, so the classifier cannot change its output within the adversarial polytope and is robust around x. □

We denote the fraction of examples that do not have this certificate as the robust error. Since adversaries can only hope to attack examples without this certificate, the robust error is a provable upper bound on the achievable error by *any* adversarial attack.

Detecting adversarial examples at test time The certificate from Theorem 1 can also be modified trivially to detect adversarial examples at test time. Specifically, we replace the bound based upon the true class $y^\star$ to a bound based upon just the predicted class $\hat{y} = \max_y f_\theta(x)_y$. In this case we have the following simple corollary.

Corollary 2. *For a data point x, model prediction $\hat{y} = \max_y f_\theta(x)_y$ and $\epsilon > 0$, if*

$$J_\epsilon(x, g_\theta(e_{\hat{y}} 1^T - I)) \geq 0 \tag{3.28}$$

then x cannot be an adversarial example. Specifically, x cannot be a perturbation of a "true" example $x^\star$ with $\|x - x^\star\|_\infty \leq \epsilon$, such that the model would correctly classify $x^\star$, but incorrectly classify x.

This corollary follows immediately from the fact that the robust bound guarantees no example with ℓ_∞ norm within ϵ of x is classified differently from x. This approach may classify non-adversarial inputs as potentially adversarial, but it has zero false negatives, in that it will never fail to flag an adversarial example. Given the challenge in even defining adversarial examples in general, this seems to be as strong a guarantee as is currently possible.

ϵ-distances to decision boundary Finally, for each example x on a fixed network, we can compute the largest value of ϵ for which a certificate of robustness exists, i.e., such that the output $f_\theta(x)$ provably cannot be flipped within the ϵ ball. Such an epsilon gives a lower bound on the ℓ_∞ distance from the example to the decision boundary (note that the classifier may or may not actually be correct). Specifically, if we find ϵ to solve the optimization problem

$$\begin{aligned} &\underset{\epsilon}{\text{maximize}} \;\; \epsilon \\ &\text{subject to} \;\; J_\epsilon(x, g_\theta(e_{f_\theta(x)} 1^T - I))_y \geq 0, \end{aligned} \tag{3.29}$$

then we know that x must be at least ϵ away from the decision boundary in ℓ_∞ distance, and that this is the largest ϵ for which we have a certificate of robustness. The certificate is monotone in ϵ, and the problem can be solved using Newton's method.

3.2 Experiments in 2D space

We consider training a robust binary classifier on a 2D input space with randomly generated spread out data points. Note that there is no notion of generalization here; we are just visualizing and evaluating the ability of the learning approach to fit a classification function robustly, in this case a 2-100-100-100-100-2 fully connected network trained with the Adam optimizer [Kingma and Ba, 2015] (over the entire batch of samples) with a learning rate of 0.001. We also plot the true adversarial polytope and our bound of the polytope for both randomly initialized networks and robustly trained networks to visualize the tightness of the bound before and after training.

3.2.1 Visualization of robust classification

We incrementally randomly sample 12 points within the $[0, 1]$ xy-plane, at each point waiting until we find a sample that is at least 0.16 away from other points via ℓ_∞ distance, and assign

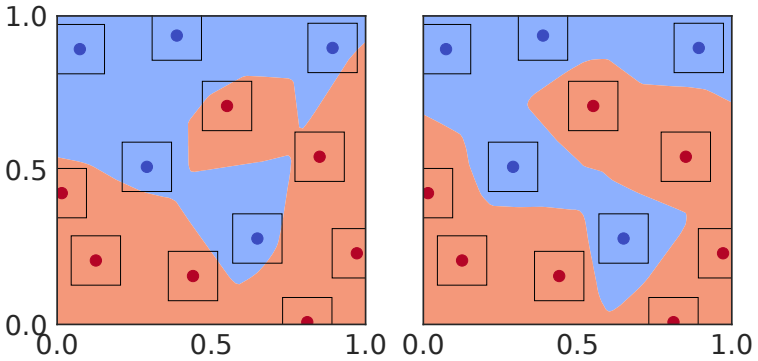


Figure 3.3: Illustration of classification boundaries resulting from standard training (left) and robust training (right) with ℓ_∞ balls of size $\epsilon = 0.08$ (shown in figure).

each point a random label. We then train a robust classifier that will correctly classify all points with an ℓ_∞ ball of $\epsilon = 0.08$ by minimizing our bound on the adversarial polytope, as well as a standard classifier for comparison.

Figure 3.3 shows the resulting classifiers produced by standard training (left) and robust training via our method (right). As expected, the standard training approach results in points that are classified differently somewhere within their ℓ_∞ ball of radius $\epsilon = 0.08$ (this is exactly an adversarial example for the training set). In contrast, the robust training method is able to attain zero robust error and provides a classifier that is guaranteed to classify all points within the balls correctly. Note that we did not need to check the visualization to know this: the robust training procedure had zero robust loss on the datapoints, and so the certificates from our bound guaranteed robustness around each point.

3.2.2 Visualization of the convex outer adversarial polytope

We consider some simple cases of visualizing the outer approximation to the adversarial polytope for random networks in Figure 3.4. Because the output space is two-dimensional we can easily visualize the polytopes in the output layer, and because the input space is two dimensional, we can easily cover the entire input space densely to enumerate the true adversarial polytope. In this experiment, we initialized the weights of the all layers to be normal $\mathcal{N}(0, 1/\sqrt{n_{\text{in}}})$ and biases normal $\mathcal{N}(0, 1)$ (due to scaling, the actual absolute value of weights is not particularly important except as it relates to ϵ). Although obviously not too much should be read into these experiments with random networks, the main takeaways are that 1) for "small" ϵ, the outer bound is an extremely good approximation to the adversarial polytope; 2) as ϵ increases, the bound gets substantially weaker. This is to be expected: for small ϵ, the number of elements in $\mathcal{I}$ will also be relatively small, and thus additional terms that make the bound lose are expected to be relatively small (in the extreme, when no activation can change, the bound will be exact, and the adversarial polytope will be a convex set). However, as ϵ gets larger, more activations enter the set $\mathcal{I}$, and the available freedom in the convex relaxation of each ReLU increases substantially, making the bound looser. Naturally, the question of interest is how tight this bound is for networks that are

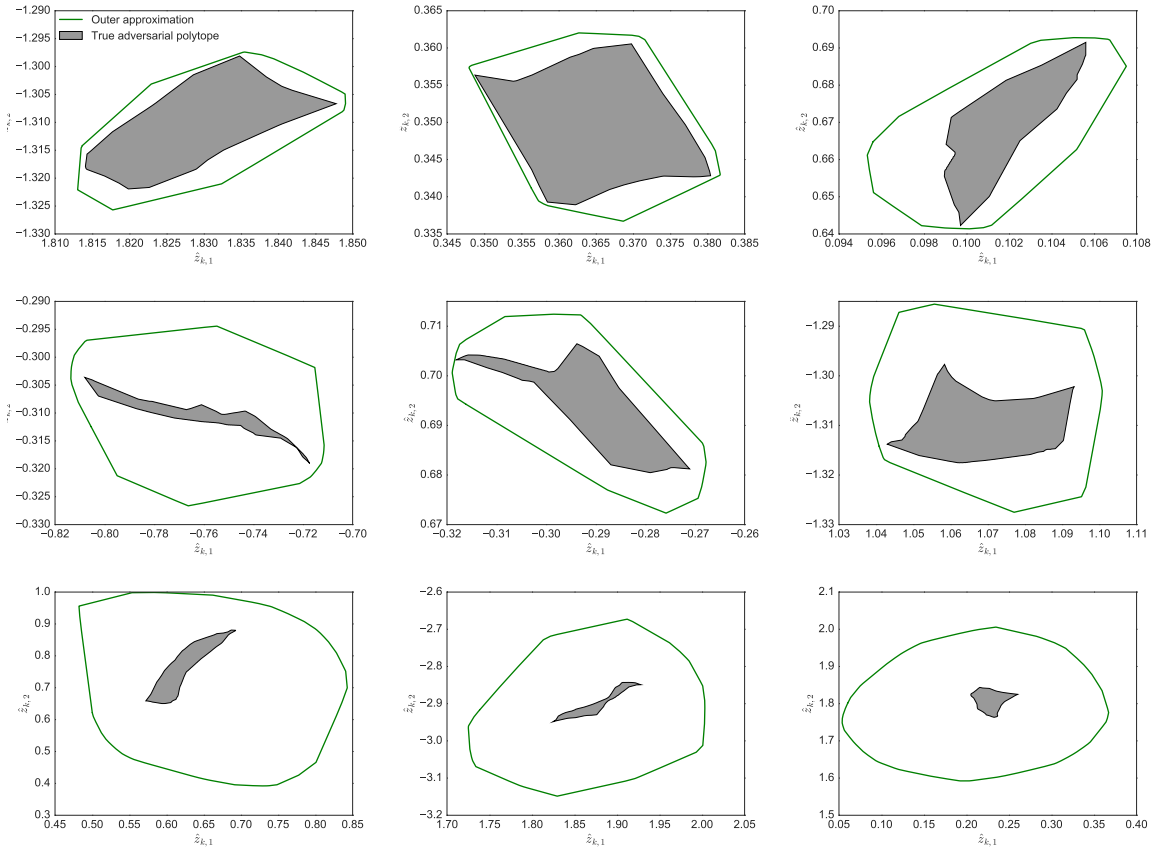


Figure 3.4: Illustrations of the true adversarial polytope (gray) and our convex outer approximation (green) for a random 2-100-100-100-100-2 network with $\mathcal{N}(0, 1/\sqrt{n})$ weight initialization. Polytopes are shown for $\epsilon = 0.05$ (top row), $\epsilon = 0.1$ (middle row), and $\epsilon = 0.25$ (bottom row).

actually trained to minimize the robust loss, which we will look at shortly.

Outer Bound after Training It is of some interest to see what the true adversarial polytope for the examples in this data set looks like versus the convex approximation, evaluated at the solution of the robust optimization problem. Figure 3.5 shows one of these figures, highlighting the fact that for the final network weights and choice of epsilon, the outer bound is empirically quite tight in this case.

3.2.3 Comparison to naive layerwise bounds

One additional point is worth making in regards to the bounds we propose. It would also be possible to achieve a naive "layerwise" bound by iteratively determining absolute allowable ranges for each activation in a network (via a simple norm bound), then for future layers, assuming each activation can vary arbitrarily within this range. This provides a simple iterative formula for computing layer-by-layer absolute bounds on the coefficients, and similar techniques have been used e.g. in Parseval Networks [Cisse et al., 2017] to produce more robust classifiers (albeit there

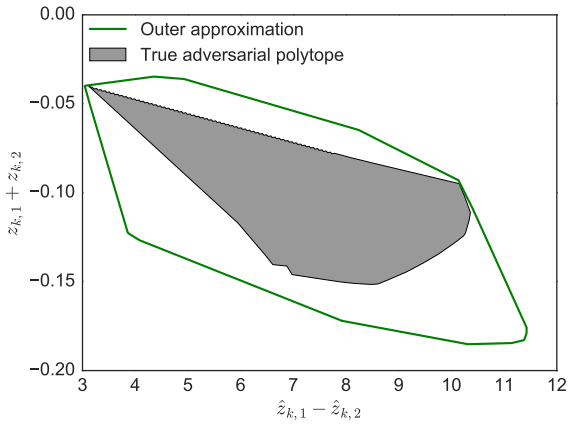


Figure 3.5: Illustration of the actual adversarial polytope and the convex outer approximation for one of the training points after the robust optimization procedure.

considering ℓ_2 perturbations instead of ℓ_∞ perturbations, which likely are better suited for such an approach). Unfortunately, these naive bounds are extremely loose for multi-layer networks (in the first hidden layer, they naturally match our bounds exactly). For instance, for the adversarial polytope shown in Figure 3.4 (top left), the actual adversarial polytope is contained within the range

$$\hat{z}_{k,1} \in [1.81, 1.85], \quad \hat{z}_{k,2} \in [-1.33, -1.29] \tag{3.30}$$

with the convex outer approximation mirroring it rather closely. In contrast, the layerwise bounds produce the bound:

$$\hat{z}_{k,1} \in [-11.68, 13.47], \quad \hat{z}_{k,2} \in [-16.36, 11.48]. \tag{3.31}$$

Such bounds are essentially vacuous in our case, which makes sense intuitively. The naive bound has no way to exploit the "tightness" of activations that lie entirely in the positive space, and effectively replaces the convex ReLU approximation with a (larger) box covering the entire space.

3.3 Experiments on real datasets

Here we demonstrate the approach on small and medium-scale problems. Although the bound in its exact form does not scale to ImageNet-sized classifiers, we do demonstrate the approach on a simple convolutional network applied to several image classification problems, illustrating that the method can apply to approaches beyond very small fully-connected networks. Code for these experiments is available at `http://github.com/locuslab/convex_adversarial`.

A summary of all the experiments is in Table 3.1. For all experiments, we report the clean test error, the error achieved by the fast gradient sign method [Goodfellow et al., 2015], the error achieved by the projected gradient descent approach [Madry et al., 2017], and the robust error bound. In all cases, the robust error bound for the robust model is significantly lower than the

Table 3.1: Test error rates for various problems and attacks, and our robust bound for baseline and robust models.

PROBLEM	ROBUST	ϵ	CLEAN	FGSM	PGD	ROBUST BOUND
MNIST	$\times$	0.1	1.07%	50.01%	81.68%	100%
MNIST	$\surd$	0.1	1.80%	3.93%	4.11%	5.82%
FASHION-MNIST	$\times$	0.1	9.36%	77.98%	81.85%	100%
FASHION-MNIST	$\surd$	0.1	21.73%	31.25%	31.63%	34.53%
HAR	$\times$	0.05	4.95%	60.57%	63.82%	81.56%
HAR	$\surd$	0.05	7.80%	21.49%	21.52%	21.90%
SVHN	$\times$	0.01	16.01%	62.21%	83.43%	100%
SVHN	$\surd$	0.01	20.38%	33.28%	33.74%	40.67%

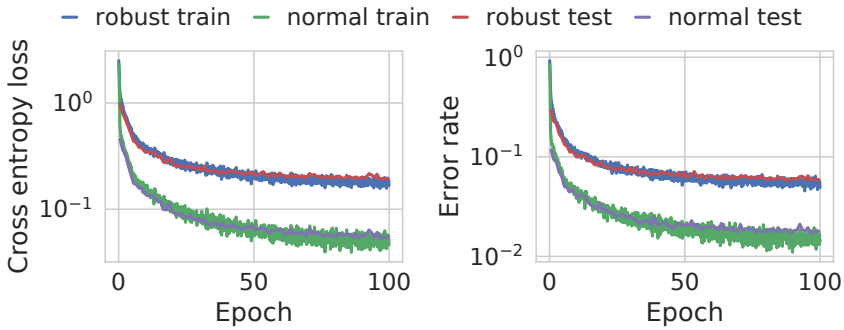


Figure 3.6: Loss (left) and error rate (right) when training a robust convolutional network on the MNIST dataset.

achievable error rates by PGD under standard training. All experiments in this section were run on a single Titan X GPU.

3.3.1 Training a provably robust MNIST classifier

We present results on a provably robust classifier on the MNIST data set. Specifically, we consider a ConvNet architecture that includes two convolutional layers, with 16 and 32 channels (each with a stride of two, to decrease the resolution by half without requiring max pooling layers), and two fully connected layers stepping down to 100 and then 10 (the output dimension) hidden units, with ReLUs following each layer except the last. We use the Adam optimizer [Kingma and Ba, 2015] with a learning rate of 0.001 (the default option) with no additional hyperparameter selection. We use minibatches of size 50 and train for 100 epochs.

ϵ scheduling Depending on the random weight initialization of the network, the optimization process for training a robust MNIST classifier may get stuck and not converge. To improve

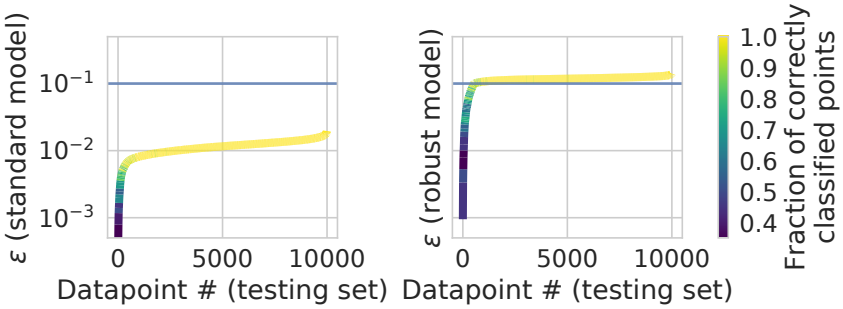


Figure 3.7: Maximum ϵ distances to the decision boundary of each data point in increasing ϵ order for standard and robust models (trained with $\epsilon = 0.1$). The color encodes the fraction of points which were correctly classified.

Table 3.2: Error rates for our LP approach in comparison to the SDP approach considered by [Raghunathan et al., 2018a] for a single hidden layer neural network with 500 units on MNIST at $\epsilon = 0.1$.

PROBLEM	PGD ERROR	LP BOUND	SDP BOUND
LP-NN	22%	26%	93%
SDP-NN	15%	99%	35%

convergence, it is helpful to start with a smaller value of ϵ and slowly increment it over epochs. For MNIST, all random seeds that we observed to not converge for $\epsilon = 0.1$ were able to converge when started with $\epsilon = 0.05$ and taking uniform steps to $\epsilon = 0.1$ in the first half of all epochs (so in this case, 50 epochs). Figure 3.6 shows the training progress using our procedure with a robust softmax loss function and $\epsilon = 0.1$. As described in Section 3.1.4, any norm-bounded adversarial technique will be unable to achieve loss or error higher than the robust bound. The final classifier after 100 epochs reaches a test error of 1.80% with a robust test error of 5.82%. For a traditionally-trained classifier (with 1.07% test error) the FGSM approach results in 50.01% error, while PGD results in 81.68% error. On the classifier trained with our method, however, FGSM and PGD only achieve errors of 3.93% and 4.11% respectively (both, naturally, below our bound of 5.82%). These results are summarized in Table 3.1.

Maximum ϵ-distances Using Newton's method with backtracking line search, for each example, we can compute in 5-6 Newton steps the maximum ϵ that is robust as described in (3.29) for both a standard classifier and the robust classifier. Figure 3.7 shows the maximum ϵ values calculated for each testing data point under standard training and robust training. Under standard training, the correctly classified examples have a lower bound of around 0.007 away from the decision boundary. However, with robust training this value is pushed to 0.1, which is expected since that is the robustness level used to train the model. We also observe that the incorrectly classified examples all tend to be relatively closer to the decision boundary.

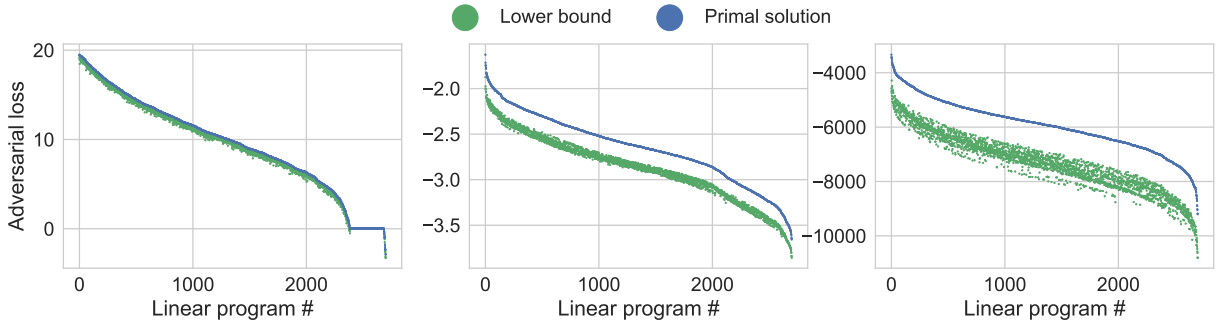


Figure 3.8: Plots of the exact solution of the primal linear program and the corresponding lower bound from the dual problem for a (left) robustly trained model, (middle) randomly intialized model, and (right) model with standard training.

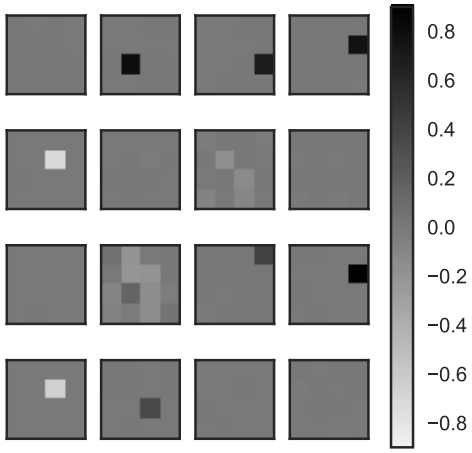


Figure 3.9: Learned convolutional filters for MNIST of the first layer of a trained robust convolutional network, which are quite sparse due to the ℓ_1 term in (3.8).

Tightness of bound We empirically evaluate the tightness of the bound by exactly computing the primal LP and comparing it to the lower bound computed from the dual problem via our method. We find that the bounds, when computed on the robustly trained classifier, are extremely tight, especially when compared to bounds computed for random networks and networks that have been trained under standard training, as can be seen in Figure 3.8.

Comparison to SDP-based provable defenses We compare our defense to the work proposed by [Raghunathan et al., 2018a], which has provable bounds and a certificate of robustness for neural networks with one hidden layer, but uses an SDP relaxation instead of an LP. In order to fairly compare with the SDP approach (which we refer to as SDP-NN), we trained a new robust MNIST classifier using our LP-based approach (which we refer to as LP-NN) with the same architecture used by the SDP approach (single hidden layer with 500 units), and compared their

Figure 3.10: Learned convolutional filters for MNIST of the second layer of a trained robust convolutional network, which are quite sparse due to the ℓ_1 term in (3.8).

robust bounds and susceptibility to PGD attacks in Table 3.2.

First, we observe that the SDP approach is unable to provide a reasonable bound to LP-NN, and the LP bound is unable to provide a reasonable bound for SDP-NN. Second, while LP-NN achieves a lower robust bound, it is less effective at defending against PGD. These numbers suggest that the two approaches are fundamentally different in how they achieve robustness.

3.3.2 Analysis of robust convolutional filters and activation patterns for MNIST

Random filters from the two convolutional layers of the MNIST classifier after robust training are plotted in Figure 3.10. We see a similar story in both layers: they are highly sparse, and some filters have all zero weights, indicating the network has been highly regularized.

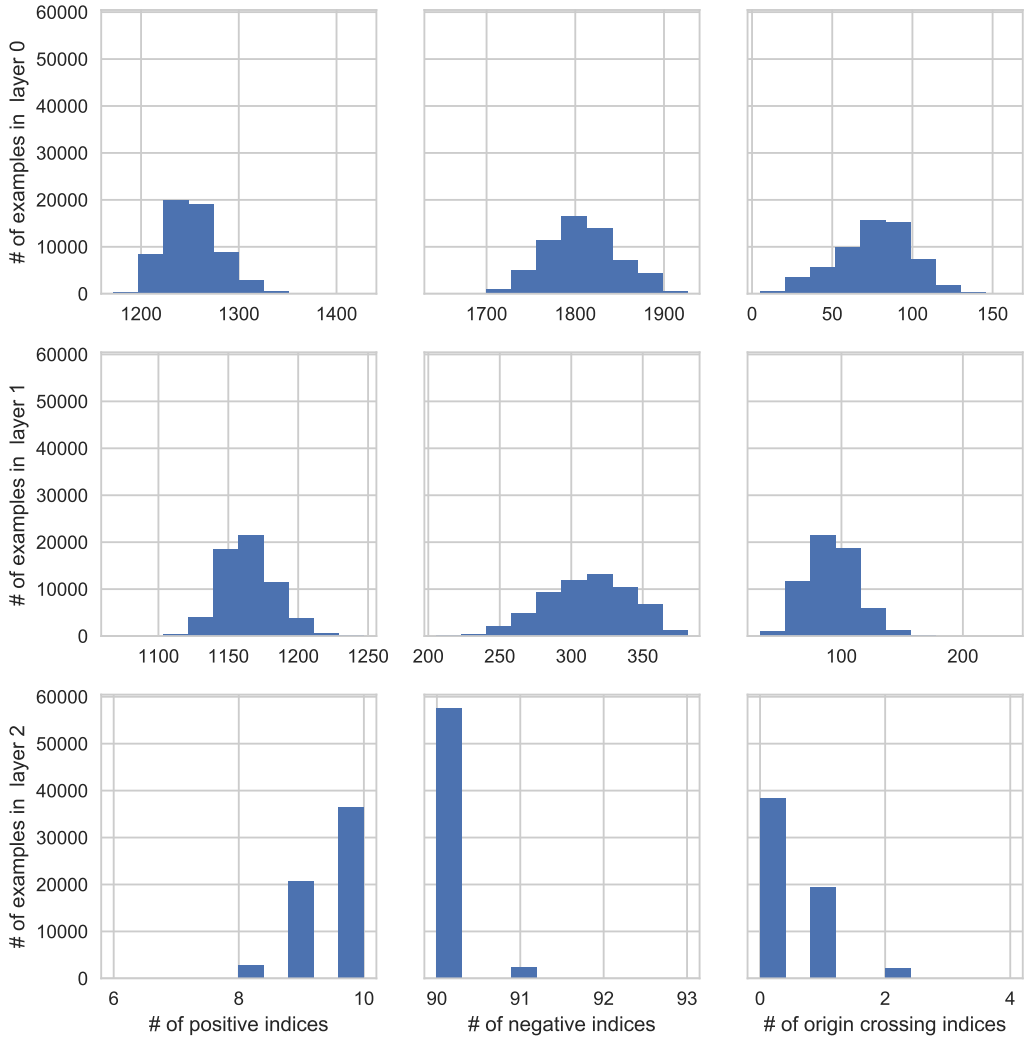


Figure 3.11: Histograms of the portion of each type of index set (as defined in 3.20 when passing training examples through the network.

We additionally plot histograms to visualize the distributions of pre-activation bounds over examples in Figure 3.11. We see that in the first layer, examples have on average more than half of all their activations in the $\mathcal{I}_1^-$ set, with a relatively small number of activations in the $\mathcal{I}_1$ set. The second layer has significantly more values in the $\mathcal{I}_2^+$ set than in the $\mathcal{I}_2^-$ set, with a comparably small number of activations in the $\mathcal{I}_2$ set. The third layer has extremely few activations in the $\mathcal{I}_3$ set, with 90% all of the activations in the $\mathcal{I}_3^-$ set. Crucially, we see that in all three layers, the number of activations in the $\mathcal{I}_i$ set is small, which benefits the method in two ways: a) it makes the bound tighter (since the bound is tight for activations through the $\mathcal{I}_i^+$ and $\mathcal{I}_i^-$ sets) and b) it makes the bound more computationally efficient to compute (since the last term of (3.8) is only summed over activations in the $\mathcal{I}_i$ set).

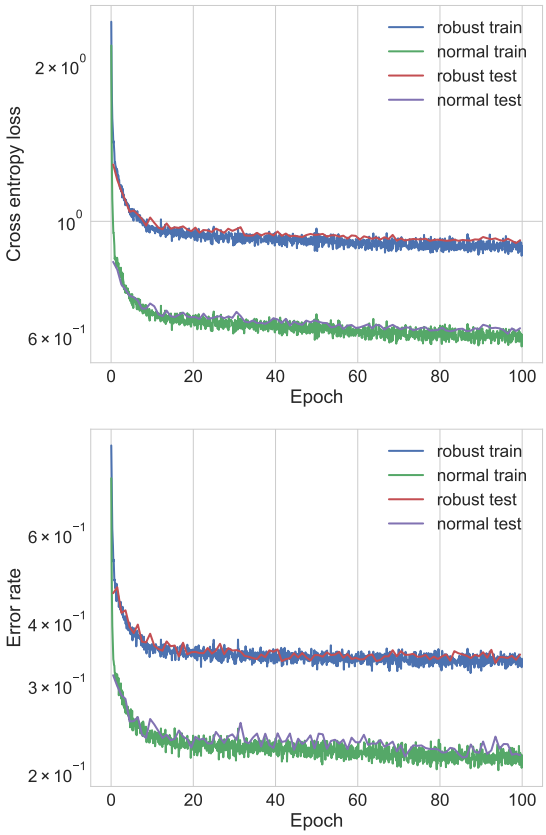


Figure 3.12: Loss (top) and error rate (bottom) when training a robust convolutional network on the Fashion-MNIST dataset.

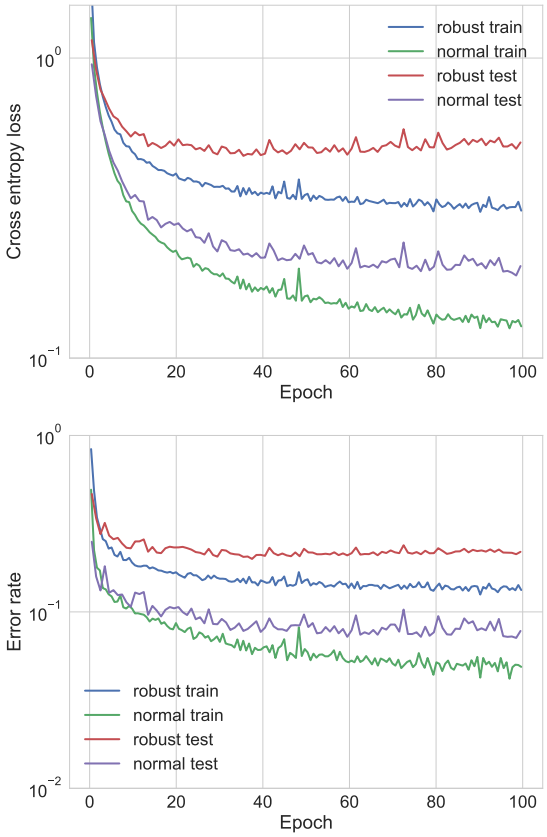


Figure 3.13: Loss (top) and error rate (bottom) when training a robust fully connected network on the HAR dataset with one hidden layer of 500 units.

3.3.3 Experiments on Fashion-MNIST, HAR, and SVHN

Fashion-MNIST We present the results of our robust classifier on the Fashion-MNIST dataset [Xiao et al., 2017], a harder dataset with the same size (in dimension and number of examples) as MNIST (for which input binarization is a reasonable defense). We use exactly the same parameters as for MNIST: Adam optimizer with the default learning rate 0.001, minibatches of size 50, and trained for 100 epochs. Using the same architecture as in MNIST, for $\epsilon = 0.1$, we achieve a robust error of 34.53%, which is fairly close to the PGD error rate of 31.63% (Table 3.1). Figure 3.12 plots the error and loss curves (and their robust variants) of the model over epochs. We observe no overfitting, and suspect that the performance on this problem is limited by model capacity.

HAR We present results on a human activity recognition dataset [Anguita et al., 2013]. Specifically, we consider a fully connected network with one layer of 500 hidden units and $\epsilon = 0.05$. We use the Adam optimizer with a learning rate 0.0001, minibatches of size 50, and trained for 100 epochs, achieving 21.90% robust error.

Table 3.3: Tightness of the bound on a single layer neural network with 500 hidden units after training on the HAR dataset with various values of ϵ. We observe that regardless of how large ϵ is, after training, the bound matches the error achievable by FGSM, implying that in this case the robust bound is tight.

ϵ	TEST ERROR	FGSM ERROR	ROBUST BOUND
0.05	9.20%	22.20%	22.80%
0.1	15.74%	36.62%	37.09%
0.25	47.66%	64.24%	64.47%
0.5	47.08%	67.32%	67.86%
1	81.80%	81.80%	81.80%

Figure 3.13 plots the error and loss curves (and their robust variants) of the model over epochs. The bottleneck here is likely due to the simplicity of the problem and the difficulty level implied by the value of ϵ, as we observed that scaling to more more layers in this setting did not help.

Tightness of bound with increasing ϵ Earlier, we observed that on random networks, the bound gets progressively looser with increasing ϵ in Figure 3.4. In contrast, we find that even if we vary the value of ϵ, after robust training on the HAR dataset with a single hidden layer, the bound *still* stays quite tight, as seen in Table 3.3. As expected, training a robust model with larger ϵ results in a less accurate model since the adversarial problem is more difficult (and potentially impossible to solve for some data points), however the key point is that the robust bounds are extremely close to the achievable error rate by FGSM, implying that in this case, the bound is tight.

SVHN Finally, we present results on SVHN. The goal here is not to achieve state of the art performance on SVHN, but to create a deep convolutional classifier for real world images with provable guarantees. We use the same architecture as in MNIST,the Adam optimizer with the default learning rate 0.001, minibatches of size 20, and trained for 100 epochs. We used an ϵ schedule which took uniform steps from $\epsilon = 0.001$ to $\epsilon = 0.01$ over the first 50 epochs. For $\epsilon = 0.01$ we achieve a robust error bound of 42.09%, with PGD achieving 34.52% error.

Note that the robust testing curve is the only curve calculated with $\epsilon = 0.01$ throughout all 100 epochs. The robust training curve was computed with the scheduled value of ϵ at each epoch. We see that all metrics calculated with the scheduled ϵ value steadily increase after the first few epochs until the desired ϵ is reached. On the other hand, the robust testing metrics for $\epsilon = 0.01$ steadily decrease until the desired ϵ is reached. Since the error rate here increases with ϵ, it suggests that for the given model capacity, the robust training cannot achieve better performance on SVHN, and a larger model is needed.

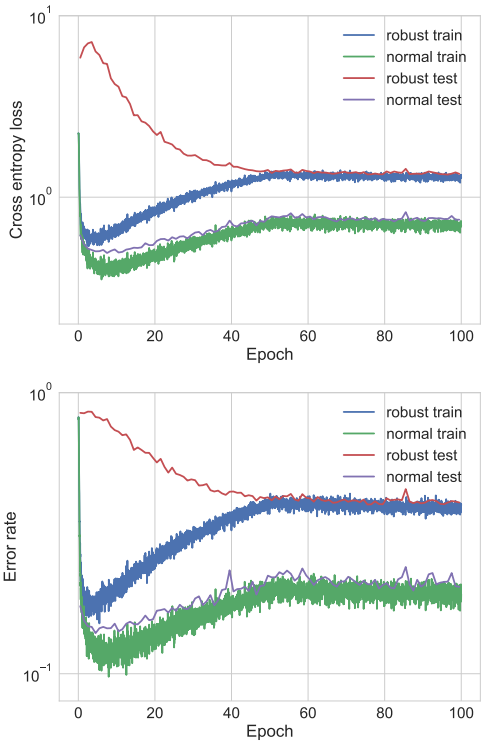


Figure 3.14: Loss (top) and error rate (bottom) when training a robust convolutional network on the SVHN dataset. The robust test curve is the only curve calculated with $\epsilon = 0.01$ throughout; the other curves are calculated with the scheduled ϵ value.

3.4 Scaling provable defenses

In this section, we extend the proposed methodology to training general network architectures in a scalable, modular manner, and makes substantial progress towards scaling these methods to realistic settings. While we cannot yet reach e.g. ImageNet scales, even in this current work, we show that it *is* possible to overcome the main hurdles to scalability of past approaches. The resulting method: 1) extends to general networks with skip connections, residual layers, and activations besides the ReLU; we do so by using a general formulation based on the Fenchel conjugate function of activations; 2) scales *linearly* in the dimensionality of the input and number of hidden units in the network, using techniques from nonlinear random projections, all while suffering minimal degradation in accuracy; and 3) further improves the quality of the bound with model cascades.

3.4.1 Robust bounds for general networks via modular dual functions

This section presents an architecture for constructing provably robust bounds for general deep network architectures, using Fenchel duality. Importantly, we derive the dual of each network operation in a fully modular fashion, simplifying the problem of deriving robust bounds of a

network to bounding the dual of individual functions. By building up a toolkit of dual operations, we can automatically construct the dual of any network architecture by iterating through the layers of the original network, without needing to go through the linear programming relaxation.

The adversarial problem for general networks We consider a generalized k "layer" neural network $f_\theta : \mathbb{R}^{|x|} \to \mathbb{R}^{|y|}$ given by the equations

$$z_i = \sum_{j=1}^{i-1} f_{ij}(z_j), \quad \text{for } i = 2, \ldots, k \tag{3.32}$$

where $z_1 = x$, $f_\theta(x) \equiv z_k$ (i.e., the output of the network) and $f_{ij} : \mathbb{R}^{|z_j|} \to \mathbb{R}^{|z_i|}$ is some function from layer j to layer i. Importantly, this differs from prior work in two key ways. First, unlike the conjugate forms found in Dvijotham et al. [2018b], Wong and Kolter [2017], we no longer assume that the network consists of linear operations followed by activation functions, and instead opt to work with an arbitrary sequence of k functions. This simplifies the analysis of sequential non-linear activations commonly found in modern architectures, e.g. max pooling or a normalization strategy followed by a ReLU,[3] by analyzing each activation independently, whereas previous work would need to analyze the entire sequence as a single, joint activation. Second, we allow layers to depend not just on the previous layer, but also on all layers before it. This generalization applies to networks with any kind of skip connections, e.g. residual networks and dense networks, and greatly expands the set of possible architectures.

Let $\mathcal{B}(x) \subset \mathbb{R}^{|x|}$, represent some input constraint for the adversary. For this section we will focus on an arbitrary norm ball $\mathcal{B}(x) = \{x + \Delta : \|\Delta\| \leq \epsilon\}$. This is the constraint set considered for norm-bounded adversarial perturbations, however other constraint sets can certainly be considered. Then, given an input example x, a known label y^*, and a target label y^{targ}, the problem of finding the most adversarial example within $\mathcal{B}$ (i.e., a so-called *targeted* adversarial attack) can be written as

$$\underset{z_k}{\text{minimize}} \quad c^T z_k, \quad \text{subject to} \quad z_i = \sum_{j=1}^{i-1} f_{ij}(z_j), \quad \text{for } i = 2, \ldots, k, \quad z_1 \in \mathcal{B}(x) \tag{3.33}$$

where $c = e_{y^\star} - e_{y^{\text{targ}}}$.

Dual networks via compositions of modular dual functions To bound the adversarial problem, we look to its dual optimization problem using the machinery of Fenchel conjugate functions [Fenchel, 1949], described in Definition 1.

Definition 1. *The conjugate of a function f is another function f^* defined by*

$$f^*(y) = \max_x x^T y - f(x) \tag{3.34}$$

[3] Batch normalization [Ioffe and Szegedy, 2015], since it depends on entire minibatches, is formally not covered by the approach, but it can be approximated by considering the scaling and shifting to be generic parameters, as is done at test time.

Specifically, we can lift the constraint $z_{i+1} = \sum_{j=1}^{i} f_{ij}(z_j)$ from Equation 3.33 into the objective with an indicator function, and use conjugate functions to obtain a lower bound. For brevity, we will use the subscript notation $(\cdot)_{1:i} = ((\cdot)_1, \ldots, (\cdot)_i)$, e.g. $z_{1:i} = (z_1, \ldots, z_i)$. Due to the skip connections, the indicator functions are not independent, so we cannot directly conjugate each individual indicator function. We can, however, still form its dual using the conjugate of a different indicator function corresponding to the backwards direction, as shown in Lemma 1.

Lemma 1. *Let the indicator function for the ith constraint be*

$$\chi_i(z_{1:i}) = \begin{cases} 0 & \text{if } z_i = \sum_{j=1}^{i-1} f_{ij}(z_j) \\ \infty & \text{otherwise,} \end{cases} \tag{3.35}$$

for $i = 2, \ldots, k$, and consider the joint indicator function $\sum_{i=2}^{k} \chi_i(z_{1:i})$. Then, the joint indicator is lower bounded by $\max_{\nu_{1:k}} \nu_k^T z_k - \nu_1^T z_1 - \sum_{i=1}^{k-1} \chi_i^*(-\nu_i, \nu_{i+1:k})$, where

$$\chi_i^*(\nu_{i:k}) = \max_{z_i} \nu_i^T z_i + \sum_{j=i+1}^{k} \nu_j^T f_{ji}(z_i) \tag{3.36}$$

for $i = 1, \ldots, k-1$. Note that $\chi_i^*(\nu_{i:k})$ is the exact conjugate of the indicator for the set $\{x_{i:k} : x_j = f_{ji}(x_i) \ \ \forall j > i\}$, which is different from the set indicated by χ_i. However, when there are no skip connections (i.e. z_i only depends on z_{i-1}), χ_i^* is exactly the conjugate of χ_i.

Proof. It is mathematically convenient to introduce additional variables $\hat{z}_{1:k}$ such that $\hat{z}_i = z_i$ for all $i = 1, \ldots, k$, and rephrase it as the equivalent constrained optimization problem.

$$\begin{aligned} &\min_{z_{1:k-1}, \hat{z}_{2:k}} \quad 0 \\ &\text{subject to } \hat{z}_i = \sum_{j=1}^{i-1} f_{ij}(z_j) \quad \text{for} \quad i = 2, \ldots, k \\ &\qquad\qquad z_i = \hat{z}_i \quad \text{for } i = 1, \ldots, k \end{aligned} \tag{3.37}$$

We introduce Lagrangian variables $\nu_{1:k}, \hat{\nu}_{2:k}$ to get the following Lagrangian:

$$L(z_{1:k}, \hat{z}_{1:k}, \nu_{1:k}, \hat{\nu}_{2:k}) = \sum_{i=2}^{k} \hat{\nu}_i^T \left(\hat{z}_i - \sum_{j=1}^{i-1} f_{ij}(z_j) \right) + \sum_{i=1}^{k} \nu_i^T (z_i - \hat{z}_i) \tag{3.38}$$

Grouping up terms by $z_i, \hat{z}_i$ and rearranging the double sum results in the following expression:

$$L(z_{1:k}, \hat{z}_{1:k}, \nu_{1:k}, \hat{\nu}_{2:k}) = -\nu_1^T \hat{z}_1 + \sum_{i=2}^{k} (\hat{\nu}_i - \nu_i)^T \hat{z}_i + \sum_{i=1}^{k} \left(\nu_i^T z_i - \sum_{j=i+1}^{k} \hat{\nu}_j^T f_{ji}(z_i) \right) \tag{3.39}$$

From the KKT stationarity conditions for the derivative with respect to $\hat{z}_i$, we know that $\hat{\nu}_i = \nu_i$. Also note that in the summand, the last term for $i = k$ has no double summand, so we move it out for clarity.

$$L(z_{1:k}, \nu_{1:k}) = -\nu_1^T \hat{z}_1 + \nu_k^T z_k + \sum_{i=1}^{k-1} \left(\nu_i^T z_i - \sum_{j=i+1}^{k} \nu_j^T f_{ji}(z_i) \right) \tag{3.40}$$

Finally, we minimize over z_i for $i = 2, \ldots, k-1$ to get the conjugate form for the lower bound via weak duality.

$$\begin{aligned} L(z_{1:k}, \nu_{1:k}) &\geq -\nu_1^T \hat{z}_1 + \nu_k^T z_k + \sum_{i=1}^{k-1} \min_{z_i} \left(\nu_i^T z_i - \sum_{j=i+1}^{k} \nu_j^T f_{ji}(z_i) \right) \\ &= -\nu_1^T \hat{z}_1 + \nu_k^T z_k - \sum_{i=1}^{k-1} \max_{z_i} \left(-\nu_i^T z_i + \sum_{j=i+1}^{k} \nu_j^T f_{ji}(z_i) \right) \\ &= -\nu_1^T z_1 + \nu_k^T z_k - \sum_{i=1}^{k-1} \chi_i^*(-\nu_i, \nu_{i+1:k}) \end{aligned} \tag{3.41}$$

□

With structured upper bounds on these conjugate functions, we can bound the original adversarial problem using the dual network described in Theorem 3. We can then optimize the bound using any standard deep learning toolkit using the same robust optimization procedure as in Section 3.1.4 but using the generalized dual bound instead.

Theorem 3. *Let g_{ij} and h_i be any functions such that*

$$\chi_i^*(-\nu_i, \nu_{i+1:k}) \leq h_i(\nu_{i:k}) \quad \text{subject to} \quad \nu_i = \sum_{j=i+1}^{k} g_{ij}(\nu_j) \tag{3.42}$$

for $i = 1, \ldots, k-1$. Then, the adversarial problem from Equation 3.33 is lower bounded by

$$J(x, \nu_{1:k}) = -\nu_1^T x - \epsilon \|\nu_1\|_* - \sum_{i=1}^{k-1} h_i(\nu_{i:k}) \tag{3.43}$$

where $\|\cdot\|_*$ is the dual norm, and $\nu_{1:k} = g(c)$ is the output of a k layer neural network g on input c, given by the equations

$$\nu_k = -c, \quad \nu_i = \sum_{j=i}^{k-1} g_{ij}(\nu_{j+1}), \quad \text{for } i = 1, \ldots, k-1. \tag{3.44}$$

Proof. First, we rewrite the primal problem by bringing the function and input constraints into the objective with indicator functions I. We can then apply Lemma 1 to get the following lower bound on the adversarial problem:

$$\underset{\nu_{1:k}}{\text{maximize}} \; \underset{z_1, z_k}{\text{minimize}} \; (c^T + \nu_k)^T z_k + I_{\mathcal{B}(x)}(z_1) - \nu_1^T z_1 - \sum_{i=1}^{k-1} \chi_i^*(-\nu_i, \nu_{i+1:k}) \tag{3.45}$$

Minimizing over z_1 and z_k, note that

$$\begin{aligned} \min_{\hat{z}_k} (c + \nu_k)^T \hat{z}_k &= -I(\nu_k = -c) \\ \min_{\hat{z}_1} I_{\mathcal{B}(x)}(z_1) - \nu_1^T z_1 &= -I^*_{\mathcal{B}(x)}(\nu_1) \end{aligned} \tag{3.46}$$

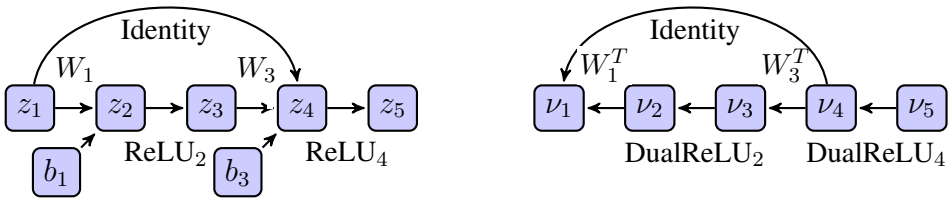


Figure 3.15: An example of the layers forming a typical residual block (left) and its dual (right), using the dual layers described in Corollaries 3 and 5. Note that the bias terms of the residual network go into the dual objective and are not part of the structure of the dual network, and the skip connections remain in the dual network but go in the opposite direction.

Note that if $\mathcal{B}(x) = \{x + \Delta : \|\Delta\| \leq \epsilon\}$ for some norm, then $I^*_{\mathcal{B}(x)}(\nu_1) = \nu_1^T x + \epsilon\|\nu_1\|_*$ where $\|\cdot\|$ is the dual norm, but any sort of input constraint can be used so long as its conjugate can be bounded. Finally, the last term can be bounded with the dual layer:

$$\min_{z_i} \nu_i^T z_i - \sum_{j=i+1}^{k} \nu_j^T f_{ji}(z_i) = -\chi_i^*(-\nu_i, \nu_{i+1:k}) \geq -h_i(\nu_{i:k}) \quad \text{subject to} \quad \nu_i = \sum_{j=i+1}^{k} g_{ij}(\nu_j) \tag{3.47}$$

Combining these all together, we get that the adversarial problem from Equation 3.33 is lower bounded by

$$\begin{aligned} \underset{\nu}{\text{maximize}} \quad & -\nu_1^T x - \epsilon\|\nu_1\|_* - \sum_{i=1}^{k-1} h_i(\nu_{i:k}) \\ \text{subject to} \quad & \nu_k = -c \\ & \nu_i = \sum_{j=i+1}^{k} g_{ij}(\nu_j) \end{aligned} \tag{3.48}$$

□

We denote the upper bound on the conjugate function from Equation 3.42 a *dual layer*.

3.4.2 Dual layers for common deep learning operators

To give concrete examples, we present dual layers for various operators in the following corollaries, and we also depict an example dual residual block in Figure 3.15. In general, dual layers tend to reflect the backward direction of the original layer. For example, the dual of a linear operator is given by Corollary 3 and involves applying the transpose weights in the backwards direction:

Corollary 3. *The dual layer for a linear operator* $\hat{z}_{i+1} = W_i z_i + b_i$ *is*

$$\chi_i^*(\nu_{i:k}) = \nu_{i+1}^T b_i \quad \text{subject to} \quad \nu_i = W_i^T \nu_{i+1}. \tag{3.49}$$

Proof. Suppose $f_i(z_i) = W_i z_i + b_i$ for some linear operator W_i and bias terms b_i. Then,

$$\begin{aligned}\chi_i^*(-\nu_i, \nu_{i+1}) &= \max_{z_i} -z_i^T \nu_i + (W_i z_i + b_i)^T \nu_{i+1} \\ &= \max_{z_i} z_i^T (W_i^T \nu_{i+1} - \nu_i) + b_i^T \nu_{i+1} \\ &= \max_{z_i} I(\nu_i = W_i^T \nu_{i+1}) + b_i^T \nu_{i+1} \\ &= b_i^T \nu_{i+1} \quad \text{subject to} \quad \nu_i = W_i^T \nu_{i+1}\end{aligned} \tag{3.50}$$

□

Residual connections have an intuitive dual layer as well, which has the same residual connection but in the opposite direction as seen in Corollary 4:

Corollary 4. *The dual layer for a residual linear connection operator $\hat{z}_{i+1} = f(z_i) + z_j$ and $z_{j+1} = W_j z_j + b_j$ for some $j < i - 1$ is*

$$\chi_j^*(\nu_{j:k}) = b_j^T \nu_j \quad \text{subject to} \quad \nu_j = W_j^T \nu_{j+1} + \nu_i \tag{3.51}$$

Proof. Writing down the definition of the conjugate function,

$$\begin{aligned}\chi_i^*(-\nu_j, \nu_{j+1}) &= \max_{z_j} -z_j^T \nu_j + z_j^T \nu_i + (W_j z_j + b_j)^T \nu_{j+1} \\ &= b_j^T \nu_j \quad \text{subject to} \quad \nu_j = W_j^T \nu_{j+1} + \nu_i\end{aligned} \tag{3.52}$$

□

Finally, we construct dual layers for two activation functions, ReLU and Hardtanh in Corollaries 5 and 6:

Corollary 5. *Suppose we have lower and upper bounds ℓ_{ij}, u_{ij} on the pre-activations. The dual layer for a ReLU activation $\hat{z}_{i+1} = \max(z_i, 0)$ is*

$$\chi_i^*(\nu_{i:k}) \leq -\sum_{j \in \mathcal{I}_i} \ell_{i,j} [\nu_{ij}]_+ \quad \text{subject to} \quad \nu_i = D_i \nu_{i+1}. \tag{3.53}$$

where $\mathcal{I}_i^-, \mathcal{I}_i^+, \mathcal{I}$ denote the index sets where the bounds are negative, positive or spanning the origin respectively, and where D_i is a diagonal matrix with entries

$$(D_i)_{jj} = \begin{cases} 0 & j \in \mathcal{I}_i^- \\ 1 & j \in \mathcal{I}_i^+ \\ \frac{u_{i,j}}{u_{i,j} - \ell_{i,j}} & j \in \mathcal{I}_i \end{cases}. \tag{3.54}$$

Proof. The conjugate function for the ReLU activation is the following:

$$\chi^*(-\nu_i, \nu_{i+1}) = \max_{z_i} -z_i^T \nu_i + \max(z_i, 0) \nu_{i+1} \tag{3.55}$$

Suppose we have lower and upper bounds ℓ_i, u_i on the input z_i. If $u_i \leq 0$, then $\max(z_i, 0) = 0$, and so

$$\chi^*(-\nu_i, \nu_{i+1}) = \max_{z_i} -z_i^T \nu_i = 0 \quad \text{subject to} \quad \nu_i = 0 \tag{3.56}$$

Otherwise, if $\ell_i \geq 0$, then $\max(z_i, 0) = z_i$ and we have

$$\chi^*(-\nu_i, \nu_{i+1}) = \max_{z_i} -z_i^T \nu_i + z_i^T \nu_{i+1} = 0 \quad \text{subject to} \quad \nu_i = \nu_{i+1} \tag{3.57}$$

Lastly, suppose $\ell_i < 0 < u_i$. Then, we can upper bound the conjugate by taking the maximum over a convex outer bound of the ReLU, namely $\mathcal{S}_i = \{(z_i, z_{i+1}) : z_{i+1} \geq 0, z_{i+1} \geq z_i, -u_i \odot z_i + (u_i - \ell_i) \odot z_{i+1} \leq -u_i \odot \ell_i\}$, where $\odot$ denotes element-wise multiplication:

$$\chi^*(-\nu_i, \nu_{i+1}) \leq \max_{\mathcal{S}_i} -z_i^T \nu_i + z_{i+1}^T \nu_{i+1} \tag{3.58}$$

The maximum must occur either at the origin $(0, 0)$ or along the line $-u_{ij} z_{ij} + (u_{ij} - \ell_{ij}) z_{i+1,j} = -u_{ij} \ell_{ij}$, so we can upper bound it again with

$$\begin{aligned}
\chi^*(-\nu_{ij}, \nu_{i+1,j}) &\leq \max_{z_{ij}} \left[-z_{ij} \nu_{ij} + \left(\frac{u_{ij}}{u_{ij} - \ell_{ij}} z_{ij} - \frac{u_{ij} \ell_{ij}}{u_{ij} - \ell_{ij}} \right) \nu_{i+1,j} \right]_+ \\
&= \max_{z_{ij}} \left[\left(\frac{u_{ij}}{u_{ij} - \ell_{ij}} \nu_{i+1,j} - \nu_{ij} \right) z_{ij} - \frac{u_{ij} \ell_{ij}}{u_{ij} - \ell_{ij}} \nu_{i+1,j} \right]_+ \\
&= \left[-\frac{u_{ij} \ell_{ij}}{u_{ij} - \ell_{ij}} \nu_{i+1,j} \right]_+ \quad \text{subject to} \quad \nu_{ij} = \frac{u_{ij}}{u_{ij} - \ell_{ij}} \nu_{i+1,j} \\
&= -\ell_{ij} \left[\nu_{ij} \right]_+ \quad \text{subject to} \quad \nu_{ij} = \frac{u_{ij}}{u_{ij} - \ell_{ij}} \nu_{i+1,j}
\end{aligned} \tag{3.59}$$

Let $\mathcal{I}_i^-, \mathcal{I}_i^+, \mathcal{I}$ and D_i be as defined in the corollary. Combining these three cases together, we get the final upper bound:

$$\chi_i^*(-\nu_i, \nu_{i+1:k}) \leq -\sum_{j \in \mathcal{I}_i} \ell_{i,j} [\nu_{i,j}]_+ \quad \text{subject to} \quad \nu_i = D_i \nu_{i+1} \tag{3.60}$$

□

Corollary 6. *Suppose we have lower and upper bounds ℓ_{ij}, u_{ij} on the pre-activations. The dual layer for the hardtanh activation $\hat{z}_{i+1} = \text{hardtanh}(z_i)$ is*

$$\chi_i^*(\nu_{i:k}) \leq d_i(\nu_i, \nu_{i+1}) \quad \text{subject to} \quad \nu_i = D_i \nu_{i+1}. \tag{3.61}$$

where D_i is a diagonal matrix with entries defined element-wise by

$$\text{diag}(D_i) = \begin{cases} \left| \left(1 - \frac{2}{1 + \max(u, \ell)} \right) \right| & u > 1 \wedge \ell < -1 \\ 0 & u \leq -1 \\ 0 & \ell \geq 1 \\ 1 & \ell \geq -1, u \leq 1 \\ \frac{1+u}{u-\ell} & \ell \leq -1, -1 \leq u \leq 1 \\ \frac{1-\ell}{u-\ell} & -1 \leq \ell \leq 1, 1 \leq u \end{cases}. \tag{3.62}$$

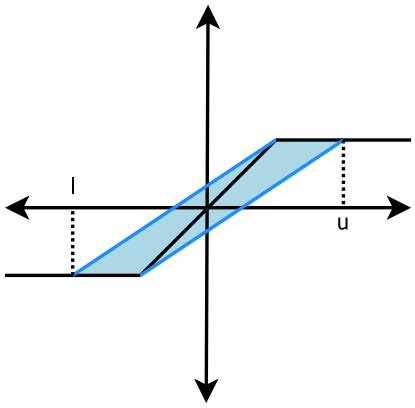


Figure 3.16: Convex relaxation of hardtanh given lower and upper bounds ℓ and u.

and $d_i(\nu_i, \nu_{i+1})$ *is a vector defined element-wise by*

$$d_i(\nu_i, \nu_{i+1}) = \begin{cases} \frac{2}{1+\max(u,\ell)}\nu_{i+1} & u > 1 \wedge \ell < -1 \\ -\nu_{i+1} & u \leq -1 \\ \nu_{i+1} & \ell \geq 1 \\ 0 & \ell \geq -1, u \leq 1 \\ \max\left(-\left(\frac{1+u}{u-\ell}\ell + 1\right)\nu_{i+1}, \nu_i - \nu_{i+1}\right) & \ell \leq -1, -1 \leq u \leq 1 \\ \max\left(-\left(\frac{1-\ell}{u-\ell}\ell - \ell\right)\nu_{i+1}, -\nu_i + \nu_{i+1}\right) & -1 \leq \ell \leq 1, 1 \leq u \end{cases}. \tag{3.63}$$

Proof. The hard tanh activation function is given by

$$\text{hardtanh}(x) = \begin{cases} -1 & \text{for} \quad x < -1 \\ x & \text{for} \quad -1 \leq x \leq 1 \\ 1 & \text{for} \quad x > 1 \end{cases} \tag{3.64}$$

Since this is an activation function (and has no skip connections), we only need to bound the following:

$$\chi^*(-\nu_i, \nu_{i+1}) = \max_{z_i} -z_i^T\nu_i + \text{hardtanh}(z_i)^T\nu_{i+1} \tag{3.65}$$

Given lower and upper bounds ℓ and u, we can use a similar convex relaxation as that used for ReLU and decompose this problem element-wise (we will now assume all terms are scalars for notational simplicity), so we have

$$\chi^*(\nu_i, \nu_{i+1}) \leq \max_{z_i, z_{i+1} \in \mathcal{S}} -z_i\nu_i + z_{i+1}\nu_{i+1} \tag{3.66}$$

where $\mathcal{S}$ is the convex relaxation. The exact form of the relaxation depends on the values of ℓ and u, and we proceed to derive the dual layer for each case. We depict the relaxation where $u > 1$ and $\ell < -1$ in Figure 3.16, and note that the remaining cases are either triangular relaxations similar to the ReLU case or exact linear regions.

Case 1: $u > 1 \wedge \ell < -1$ We can use the relaxation given in Figure 3.16. The upper bound goes through the points $(\ell, -1)$ and $(1, 1)$ while the lower bound goes through the points $(-1, -1)$ and $(u, 1)$. The slope of the first one is $\frac{2}{1-\ell}$ and the slope of the second one is $\frac{2}{u+1}$, so we have either

$$z_{i+1} = \frac{2}{1-\ell}(z_i - 1) + 1, \quad z_{i+1} = \frac{2}{u+1}(z_i + 1) - 1 \tag{3.67}$$

Taking the maximum over these two cases, we have our upper bound of the conjugate is

$$\chi^*(\nu_i, \nu_{i+1}) \leq \max\left(-z_i\nu_i + \left(\frac{2}{1-\ell}(z_i - 1) + 1\right)\nu_{i+1}, -z_i\nu_i + \left(\frac{2}{u+1}(z_i + 1) - 1\right)\nu_{i+1}\right) \tag{3.68}$$

Simplifying we get

$$\begin{aligned}\chi^*(\nu_i, \nu_{i+1}) \leq \max\Bigg(&z_i\left(-\nu_i + \frac{2}{1-\ell}\nu_{i+1}\right) + \left(1 - \frac{2}{1-\ell}\right)\nu_{i+1}, \\ &z_i\left(-\nu_i + \frac{2}{u+1}\nu_{i+1}\right) + \left(\frac{2}{u+1} - 1\right)\nu_{i+1}\Bigg)\end{aligned} \tag{3.69}$$

So each case becomes

$$\begin{aligned}\chi^*(\nu_i, \nu_{i+1}) \leq \max\Bigg(&\left(1 - \frac{2}{1-\ell}\right)\nu_{i+1} \quad \text{subject to} \quad \nu_i = \frac{2}{1-\ell}\nu_{i+1}\,, \\ &\left(\frac{2}{u+1} - 1\right)\nu_{i+1} \quad \text{subject to} \quad \nu_i = \frac{2}{u+1}\nu_{i+1}\Bigg)\end{aligned} \tag{3.70}$$

As a special case, note that when $u = -\ell$, we have

$$\chi^*(\nu_i, \nu_{i+1}) \leq \left|\left(1 - \frac{2}{1+u}\right)\nu_{i+1}\right| \quad \text{subject to} \quad \nu_i = \frac{2}{1+u}\nu_{i+1} \tag{3.71}$$

This dual layer is linear, and so we can continue to use random projections for efficient bound estimation.

Case 2: $u \leq -1$ Then, $\mathcal{S} = \{z_{i+1} = -1\}$ and so

$$\chi^*(\nu_i, \nu_{i+1}) = \max_{z_i} -z_i\nu_i - \nu_{i+1} = -\nu_{i+1} \quad \text{subject to} \quad \nu_i = 0 \tag{3.72}$$

Case 3: $\ell \geq 1$ Then, $\mathcal{S} = \{z_{i+1} = 1\}$ and so

$$\chi^*(\nu_i, \nu_{i+1}) = \max_{z_i} -z_i\nu_i + \nu_{i+1} = \nu_{i+1} \quad \text{subject to} \quad \nu_i = 0 \tag{3.73}$$

Case 4: $\ell \geq -1, u \leq 1$ Then, $\mathcal{S} = \{z_{i+1} = z_i\}$ and so

$$\chi^*(\nu_i, \nu_{i+1}) = \max_{z_i} -z_i\nu_i + z_i\nu_{i+1} = 0 \quad \text{subject to} \quad \nu_i = \nu_{i+1} \tag{3.74}$$

Case 5: $\ell \leq -1, -1 \leq u \leq 1$ Here, our relaxation consists of the triangle above the hardtanh function. Then, the maximum occurs either on the line $z_{i+1} = \frac{1+u}{u-\ell}(z_i - \ell) - 1$ or at $(-1, -1)$. This line is equivalent to $z_{i+1} = \frac{1+u}{u-\ell} z_i - \left(\frac{1+u}{u-\ell}\ell + 1\right)$, and the point $(-1, -1)$ has objective value $\nu_i - \nu_{i+1}$, so we get

$$\chi^*(\nu_i, \nu_{i+1}) \leq \max_{z_i} -z_i\nu_i + \frac{1+u}{u-\ell} z_i\nu_{i+1} - \left(\frac{1+u}{u-\ell}\ell + 1\right)\nu_{i+1} \tag{3.75}$$

Pulling out the maximization over z_i, we get

$$\chi^*(\nu_i, \nu_{i+1}) \leq \max\left(-\left(\frac{1+u}{u-\ell}\ell + 1\right)\nu_{i+1}, \nu_i - \nu_{i+1}\right) \quad \text{subject to} \quad \nu_i = \frac{1+u}{u-\ell}\nu_{i+1} \tag{3.76}$$

Case 6: $-1 \leq \ell \leq 1, 1 \leq u$ Here, our relaxation consists of the triangle below the hardtanh function. Then, the maximum occurs either on the line $z_{i+1} = \frac{1-\ell}{u-\ell}(z_i - \ell) + \ell$ or at $(1, 1)$. This line is equivalent to $z_{i+1} = \frac{1-\ell}{u-\ell} z_i - \left(\frac{1-\ell}{u-\ell}\ell - \ell\right)$, and at the point $(1, 1)$ has objective value $-\nu_i + \nu_{i+1}$, so we get

$$\chi^*(\nu_i, \nu_{i+1}) \leq \max_{z_i} -z_i\nu_i + \frac{1-\ell}{u-\ell} z_i\nu_{i+1} - \left(\frac{1-\ell}{u-\ell}\ell - \ell\right)\nu_{i+1} \tag{3.77}$$

Pulling out the maximization over z_i, we get

$$\chi^*(\nu_i, \nu_{i+1}) \leq \max\left(-\left(\frac{1-\ell}{u-\ell}\ell - \ell\right)\nu_{i+1}, -\nu_i + \nu_{i+1}\right) \quad \text{subject to} \quad \nu_i = \frac{1-\ell}{u-\ell}\nu_{i+1} \tag{3.78}$$

□

Finally, Corollary 7 handles batch normalization at test time with fixed mean and variance. During training, we can use the batch statistics as a training heuristic. Note however, that batch normalization has the effect of shifting the activations to be centered more around the origin, which is exactly the case in which the robust bound becomes looser. In practice, we find that while including batch normalization may improve convergence, it reduces the quality of the bound.

Corollary 7. *Let μ_i, σ_i be the fixed mean and variance statistics, so batch normalization has the following form:*

$$BN(z_i) = \gamma\frac{x_i - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}} + \beta \tag{3.79}$$

where γ, β are the batch normalization parameters. Then, the conjugate of the batch normalization layer is

$$\chi_i^*(-\nu_i, \nu_{i+1:k}) = d_i^T\nu_{i+1} \quad \text{subject to} \quad \nu_i = D_i\nu_{i+1} \tag{3.80}$$

where $D_{i+1} = \operatorname{diag}\left(\frac{\gamma}{\sqrt{\sigma^2+\epsilon}}\right)$ and $d_{i+1} = \beta - \frac{\mu}{\sqrt{\sigma^2+\epsilon}}$.

Proof. Rewrite the batch normalization operator as

$$z_i = \gamma \frac{\hat{z}_i - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta = D_i z_i + d_i \tag{3.81}$$

This is now just a sub-case of the linear case, and plugging this into Corollary 3 finishes the proof. □

We briefly note that these dual layers recover the original dual network described in Section 3.1.2. Furthermore, the dual linear operation is an exact conjugate and introduces no looseness to the bound, while the dual ReLU uses the same relaxation used in Ehlers [2017] and in Section 3.1.1. More generally, the strength of the bound from Theorem 3 relies entirely on the tightness of the individual dual layers to their respective conjugate functions in Equation 3.42. While any g_{ij}, h_i can be chosen to upper bound the conjugate function, a tighter bound on the conjugate results in a tighter bound on the adversarial problem.

If the dual layers for all operations are linear, the bounds for all layers can be computed with a single forward pass through the dual network using a direct generalization of the form used in Section 3.1.3, which will be discussed in Section 3.4.3. By trading off tightness of the bound with computational efficiency by using linear dual layers, we can efficiently compute all bounds and construct the dual network one layer at a time. The end result is that we can automatically construct dual networks from dual layers in a fully modular fashion, completely independent of the overall network architecture (similar to how auto-differentiation tools proceed one function at a time to compute all parameter gradients using only the local gradient of each function). With a sufficiently comprehensive toolkit of dual layers, we can compute provable bounds on the adversarial problem for any network architecture.

For analytical forms of conjugate functions of other activation functions such as tanh, sigmoid, and max pooling, we refer the reader to Dvijotham et al. [2018b].

3.4.3 AutoDual

In this section, we describe our generalization of Algorithm 1, the bounds computation algorithm from Section 3.1.3, to general networks using dual layers, which we call AutoDual.

Efficient construction of the dual network via linear dual operators The conjugate form, and consequently the dual layer, for certain activations requires knowing lower and upper bounds for the pre-activations, as was done for ReLU activations in Algorithm 1 of Wong and Kolter [2017]. While the bound in Equation 3.43 can be immediately used to compute all the bounds on intermediate nodes of the network one layer at a time, this requires performing a backwards pass through the dual network whenever we need to compute the bounds. However, if the operators g_{ij} of the dual layers are all affine operators $g_{ij}(\nu_{i+1}) = A_{ij}^T \nu_{i+1}$ for some affine operator A_{ij}, we can apply a generalization of the lower and upper bound computation found in Wong and Kolter [2017] to compute all lower and upper bounds, and consequently the dual layers, of the entire network with a single forward pass in a layer-by-layer fashion. With the lower and upper bounds, we can also use the same algorithm to automatically construct the dual network. The resulting algorithm, which we call AutoDual, is described in Algorithm 2.

Algorithm 2 Autodual: computing the bounds and dual of a general network

input: Network operations f_{ij}, data point x, ball size ϵ
// initialization
$\nu_1^{(1)} := I$
$\ell_2 := x - \epsilon$
$u_2 := x + \epsilon$
for $i = 2, \ldots, k-1$ **do**
  // initialize new dual layer
  Create dual layer operators A_{ji} and h_i from f_{ji}, ℓ_j and u_j for all $j \leq i$
  $\nu_i^{(i)} := I$.
  // update all dual variables
  for $j = 1, \ldots, i-1$ **do**
    $\nu_j^{(i)} := \sum_{k=1}^{j-1} A_{ki} \nu_j^{(k)}$
  end for
  // compute new bounds
  $\ell_{i+1} := x^T \nu_1^{(i)} - \epsilon \|\nu_1^{(i)}\|_: + \sum_{j=1}^{i} h_j(\nu_j^{(i)}, \ldots, \nu_i^{(i)})$
  $u_{i+1} := x^T \nu_1^{(i)} + \epsilon \|\nu_1^{(i)}\|_: - \sum_{j=1}^{i} h_j(-\nu_j^{(i)}, \ldots, -\nu_i^{(i)})$
  // $\|\cdot\|_:$ for a matrix here denotes the norm of all rows
end for
output: bounds $\{\ell_i, u_i\}_{i=2}^k$, dual layer operators A_{jk}, h_i

In practice, we can perform several layer-specific enhancements on top of this algorithm. First, many of the A_{ji} operators will not exist simply because most architectures (with a few exceptions) don't have a large number of skip connections, so these become no ops and can be ignored. Second, we can lazily skip the computation of layer-wise bounds until necessary, e.g. for constructing the dual layer of ReLU activations. Third, since many of the functions h_j in the dual layers are functions of $B^T \nu_i$ for some matrix B and some $i \geq j$, we can initialize $\nu_i^{(i)}$ with B instead of the identity matrix, typically passing a much smaller matrix through the dual network (in many cases, B is a single vector).

3.4.4 Connection from the linear program to the dual conjugate bound

Recall from section 3.1.2 that the dual feasible solution for the linear program selected a specific feasible solution for the dual variable $\alpha = \frac{u}{u-\ell}$. Here, we provide some additional motivation for this choice from the conjugate function perspective. Recall the conjugate of the ReLU from the third case of Corollary 5, which used a looser than necessary conjugate from maximizing over all possible preactivations $z \in \mathbb{R}$. We can instead look at a tighter conjugate function for the

ReLU operator by only maximizing over $z \in [\ell, u]$, which results in:

$$\begin{aligned}
\chi^*(-\nu_i, \nu_{i+1}) &= \left[\max_{0<\hat{z}<u} \nu_{i+1} \cdot \frac{u}{u-\ell}(\hat{z}-\ell) + -\nu_i \cdot \hat{z}\right]_+ \\
&= \left[\max_{0<\hat{z}<u} \left(\frac{u}{u-\ell} y - \nu_i\right) \hat{z} - \frac{u\ell}{u-\ell}\nu_{i+1}\right]_+ \\
&= \left[\max_{0<\hat{z}<u} \nu_{i+1} \cdot \frac{u}{u-\ell}(\hat{z}-\ell) - \nu_i \cdot \hat{z} = g(-\nu_i, \nu_{i+1})\right]_+ \\
&= \begin{cases} \left[-\frac{u\ell}{u-\ell}\nu_{i+1}\right]_+ & \text{if } \frac{u}{u-\ell}\nu_{i+1} - \nu_i \leq 0 \\ \left[\left(\frac{u}{u-\ell}\nu_{i+1} - \nu_i\right) u - \frac{u\ell}{u-\ell}\nu_{i+1}\right]_+ & \text{if } \frac{u}{u-\ell}\nu_{i+1} - \nu_i > 0 \end{cases}
\end{aligned} \tag{3.82}$$

Observe that the second case is always larger than first, so we get a tighter upper bound when $\frac{u}{u-\ell}\nu_{i+1} - \nu_i \leq 0$. If we plug in $\hat{y} = -\nu$ and $y = \hat{\nu}$, this condition is equivalent to the following:

$$\frac{u}{u-\ell}\nu_{i+1} \leq \nu_i \tag{3.83}$$

Also recall that in the LP form, the forward pass of the dual network in this case was defined as the following:

$$\nu_i = \frac{u}{u-\ell}[\nu_{i+1}]_+ + \alpha[\nu_{i+1}]_- \tag{3.84}$$

Then, $\alpha = \frac{u}{u-l}$ can be interpreted as the *largest choice of* α *which does not increase the bound* (because if α was any larger, we would enter the second case and add an additional $\left(\frac{u}{u-\ell}\nu_{i+1} - \nu_i\right) u$ term to the bound). We can further verify that the forward pass of the dual network obtained from choosing $\alpha = \frac{u}{u-\ell}$ results in the same conjugate of the ReLU derived in Corollary 5 by simply plugging this forward pass into Equation (3.82) and checking that the objective and constraints match exactly for this choice in α.

3.4.5 Efficient bound estimation for ℓ_∞ perturbations via random projections

While being extended to general architectures, the proposed algorithm is still limited by its computational complexity: for instance, to compute the bounds exactly for ℓ_∞ norm bounded perturbations in ReLU networks, it is computationally expensive to calculate $\|\nu_1\|_1$ and $\sum_{j \in \mathcal{I}_i} \ell_{ij}[\nu_{ij}]_+$. In contrast to other terms like $\nu_{i+1}^T b_i$ which require only sending a single bias vector through the dual network, the matrices ν_1 and $\nu_{i,\mathcal{I}_i}$ must be explicitly formed by sending an example through the dual network for each input dimension and for each $j \in \mathcal{I}_i$, which renders the entire computation *quadratic* in the number of hidden units. To scale the method for larger, ReLU networks with ℓ_∞ perturbations, we look to random Cauchy projections. Note that for an ℓ_2 norm bounded adversarial perturbation, the dual norm is also an ℓ_2 norm, so we can use traditional random projections [Vempala, 2005]. Experiments for the ℓ_2 norm are explored further in Section 3.5.4. However, for the remainder of this section we focus on the ℓ_1 case arising from ℓ_∞ perturbations.

Estimating with Cauchy random projections From the work of Li et al. [2007], we can use the sample median estimator with Cauchy random projections to directly estimate $\|\nu_1\|_1$ for linear dual networks, and use a variation to estimate $\sum_{j\in\mathcal{I}} \ell_{ij}[\nu_{ij}]_+$, as shown in Theorem 4.

Theorem 4. . *Let $\nu_{1:k}$ be the dual network from Equation 3 with linear dual layers and let $r > 0$ be the projection dimension. Then, we can estimate*

$$\|\nu_1\|_1 \approx \text{median}(|\nu_1^T R|) \tag{3.85}$$

where R is a $|z_1| \times r$ standard Cauchy random matrix and the median is taken over the second axis. Furthermore, we can estimate

$$\sum_{j\in\mathcal{I}} \ell_{ij}[\nu_{ij}]_+ \approx \frac{1}{2}\left(-\text{median}(|\nu_i^T \operatorname{diag}(d_i) R|) + \nu_i^T d_i\right), \quad d_{i,j} = \begin{cases} \frac{u_{i,j}}{u_{i,j}-\ell_{i,j}} & j \notin \mathcal{I}_i \\ 0 & j \in \mathcal{I}_i \end{cases} \tag{3.86}$$

where R is a $|z_i| \times r$ standard Cauchy random matrix, and the median is taken over the second axis.

Proof. We begin by estimating $\|\hat{\nu}_1\|_{1,:}$. Recall the form of $\hat{\nu}_1$,

$$\hat{\nu}_1 = I W_1^T D_2 W_2^T \dots D_n W_n^T = g(I)$$

where we include the identity term to make explicit the fact that we compute this by passing an identity matrix through the network g. Estimating this term is straightforward: we simply pass in a Cauchy random matrix R, and take the median absolute value:

$$\|\hat{\nu}_1\|_{1,:} \approx \text{median}(|R W_1^T D_2 W_2^T \dots D_n W_n^T|) = \text{median}(|g(R)|)$$

where the median is taken over the minibatch axis.

Next, we show how to estimate $\sum_i [\nu_{i,:}]_+$. Recall the form of $\nu = \nu_j$ for some layer j,

$$\nu_j = I D_j W_j^T \dots D_n W_n^T = g_j(I)$$

Note that for a vector x,

$$\sum_i [x]_+ = \frac{\|x\|_1 + 1^T x}{2}$$

So we can reuse the ℓ_1 approximation from before to get

$$\sum_i [\nu_{i,:}]_+ = \frac{\|\nu\|_{1,:} + 1^T \nu}{2} \approx \frac{|\,\text{median}(g_j(R)) + g_j(1^T)|}{2}$$

which involves using the same median estimator and also passing in a single example of ones through the network.

Finally we extend this to estimating $\sum_{i\in\mathcal{I}} \ell_i [\nu_{i,:}]_+$. The previous equation, while simple, is not exactly the term in the objective; there is an addition ℓ_1 factor for each row, and we only add

rows in the $\mathcal{I}$ set. However, we can deal with this by simply passing in a modified input to the network, as we will see shortly:

$$\begin{aligned}\sum_{i\in\mathcal{I}} \ell_i[\nu_{i,:}]_+ &= \sum_{i\in\mathcal{I}} \ell_i \frac{|\nu_{i,:}| + \nu_{i,:}}{2} \\ &= \frac{1}{2}\left(\sum_{i\in\mathcal{I}} \ell_i|\nu_{i,:}| + \sum_{i\in\mathcal{I}} \ell_i\nu_{i,:}\right) \\ &= \frac{1}{2}\left(\sum_{i\in\mathcal{I}} \ell_i|g_j(I)_i| + \sum_{i\in\mathcal{I}} \ell_i g_j(I)_i\right)\end{aligned} \tag{3.87}$$

Note that since g_j is just a linear function that does a forward pass through the network, for any matrix A, B,

$$Ag_j(B) = ABD_jW_j^T \dots D_nW_n^T = g_j(AB).$$

So we can take the multiplication by scaling terms ℓ to be an operation on the input to the network (note that we assume $\ell_i < 0$, which is true for all $i \in \mathcal{I}$)

$$\sum_{i\in\mathcal{I}} \ell_i[\nu_{i,:}]_+ = \frac{1}{2}\left(-\sum_{i\in\mathcal{I}} |g_j(\operatorname{diag}(\ell))_i| + \sum_{i\in\mathcal{I}} g_j(\operatorname{diag}(\ell))_i\right) \tag{3.88}$$

Similarly, we can view the summation over the index set $\mathcal{I}$ as a summation after multiplying by an indicator matrix $1_{\mathcal{I}}$ which zeros out the ignored rows. Since this is also linear, we can move it to be an operation on the input to the network.

$$\sum_{i\in\mathcal{I}} \ell_i[\nu_{i,:}]_+ = \frac{1}{2}\left(-\sum_{i} |g_j(1_{\mathcal{I}}\operatorname{diag}(\ell))_i| + \sum_{i} g_j(1_{\mathcal{I}}\operatorname{diag}(\ell))_i\right) \tag{3.89}$$

Let the linear, preprocessing operation be $h(X) = X1_{\mathcal{I}}\operatorname{diag}(\ell)$ so

$$h(I) = 1_{\mathcal{I}}\operatorname{diag}(\ell).$$

Then, we can observe that the two terms are simply an $\ell_{1,:}$ operation and a summation of the network output after applying g_j to $h(I)$ (where in the latter case, since everything is linear we can take the summation inside both g and h to make it $g_j(h(1^T))$):

$$\sum_{i\in\mathcal{I}} \ell_i[\nu_{i,:}]_+ = \frac{1}{2}\left(-\|g_j(h(I))\|_{1,:} + g_j(h(1^T))\right) \tag{3.90}$$

The latter term is cheap to compute, since we only pass a single vector. We can approximate the first term using the median estimator on the compound operations $g \circ h$ for a Cauchy random matrix R:

$$\sum_{i\in\mathcal{I}} \ell_i[\nu_{i,:}]_+ \approx \frac{1}{2}\left(-\operatorname{median}(|g_j(h(R))|) + g_j(h(1^T))\right) \tag{3.91}$$

and rewriting $h(R) = \operatorname{diag}(d_i)R$ completes the proof. □

Algorithm 3 Estimating $\|\nu_1\|_1$ and $\sum_{j\in\mathcal{I}} \ell_{ij}[\nu_{ij}]_+$

input: Linear dual network operations g_{ij}, projection dimension r, lower bounds ℓ_{ij}, d_{ij} from Equation 3.86, layer-wise sizes $|z_i|$
$R_1^{(1)} := \text{Cauchy}(r, |z_1|)$ *// initialize random matrix for ℓ_1 term*
for $i = 2, \ldots, k$ **do**
  // pass each term forward through the network
  for $j = 1, \ldots, i-1$ **do**
    $R_j^{(i)}, S_j^{(i)} := \sum_{k=1}^{i-1} g_{ki}^T(R_i^{(k)}), \sum_{k=1}^{i-1} g_{ki}^T(S_i^{(k)})$
  end for
  $R_i^{(i)}, S_i^{(i)} := \text{diag}(d_i)\text{Cauchy}(|z_i|, r), d_i$ *// initialize terms for layer i*
end for
output: $\text{median}(|R_1^{(k)}|), 0.5\left(-\text{median}(|R_2^{(k)}|) + S_2^{(k)}\right), \ldots, 0.5\left(-\text{median}(|R_k^{(k)}|) + S_k^{(k)}\right)$

The end result is that this term can be estimated by generating a Cauchy random matrix, scaling its terms by ℓ and zeroing out columns in $\mathcal{I}$, then passing it through the network and taking the median. This estimate has two main advantages: first, it is simple to compute, as evaluating $\nu_1^T R$ involves passing the random matrix forward through the dual network (similarly, the other term requires passing a modified random matrix through the dual network; the exact algorithm is detailed in 3). Second, it is memory efficient in the backward pass, as the gradient need only propagate through the median entries.

These random projections reduce the computational complexity of computing these terms to piping r random Cauchy vectors (and an additional vector) through the network. Crucially, the complexity is no longer a quadratic function of the network size: if we fix the projection dimension to some constant r, then the computational complexity is now linear with the input dimension and $\mathcal{I}_i$. At test time, the bound can be computed exactly, as the gradients no longer need to be stored. However, if desired, it is possible to use a different estimator (specifically, the geometric estimator) for the ℓ_∞ norm to calculate high probability bounds on the adversarial problem.

3.4.6 Efficient high probability estimates of the bound

In this section, we derive high probability certificates for robustness against adversarial examples. In contrast to the previous estimate of the bound using the median Cauchy estimator which may have some non-negligible variance, the bounds in this section hold with high probability. Recall that the original certificate is of the form

$$J(g(c, \alpha)) < 0,$$

so if this holds we are guaranteed that the example cannot be adversarial. What we will show is an equivalent high probability statement: for $\delta > 0$, with probability at least $(1 - \delta)$,

$$J(g(c, \alpha)) \leq \tilde{J}(g(c, \alpha))$$

where $\tilde{J}$ is equivalent to the original J but using a high probability ℓ_1 upper bound. Then, if $\tilde{J}(g(c, \alpha)) < 0$ then with high probability we have a certificate.

While the median estimator is a good heuristic for training, it is still only an estimate of the bound. At test time, it is possible to create a provable bound that holds with high probability, which may be desired if computing the exact bound is computationally impossible.

In this section, we derive high probability certificates for robustness against adversarial examples. Recall that the original certificate is of the form

$$J(g(c, \alpha)) < 0,$$

so if this holds we are guaranteed that the example cannot be adversarial. What we will show is an equivalent high probability statement: for $\delta > 0$, with probability at least $(1 - \delta)$,

$$J(g(c, \alpha)) \leq \tilde{J}(g(c, \alpha))$$

where $\tilde{J}$ is equivalent to the original J but using a high probability upper bound on the ℓ_1 norm. Then, if $\tilde{J}(g(c, \alpha)) < 0$ then with high probability we have a certificate.

Tail bounds for the geometric estimator From Li et al. [2007], the authors also provide a geometric mean estimator which comes with high probability tail bounds. The geometric estimator is

$$\|\hat{\nu}_1\|_{1,j} \approx \prod_{i=1}^{k} |g(R)_{i,j}|^{1/k}$$

and the relevant lower tail bound on the ℓ_1 norm is

$$P\left(\frac{1}{1-\epsilon}\prod_{i=1}^{k} |g(R)_{i,j}|^{1/k} \leq \|\hat{\nu}_1\|_{1,j}\right) \leq \exp\left(-k\frac{\epsilon^2}{G_{L,gm}}\right) \tag{3.92}$$

where

$$G_{L,gm} = \frac{\epsilon^2}{\left(-\frac{1}{2}\log\left(1 + \left(\frac{2}{\pi}\log(1-\epsilon)\right)^2\right) + \frac{2}{\pi}\tan^{-1}\left(\frac{2}{\pi}\log(1-\epsilon)\right)\log(1-\epsilon)\right)}$$

Thus, if $\exp\left(-k\frac{\epsilon^2}{G_{L,gm}}\right) \leq \delta$, then with probability $1 - \delta$ we have that

$$\|\hat{\nu}_1\|_{1,j} \leq \frac{1}{1-\epsilon}\prod_{i=1}^{k} |g(R)_{i,j}|^{1/k} = \text{geo}(R)$$

which is a high probability upper bound on the ℓ_1 norm.

Upper bound on $J(g(c,\alpha))$ In order to upper bound $J(g(c,\alpha))$, we must apply the ℓ_1 upper bound for *every* ℓ_1 term. Let $n_1,\dots,n_k$ denote the number of units in each layer of a k layer neural network, then we enumerate all estimations as follows:

1. The ℓ_1 norm computed at each intermediary layer when computing iterative bounds. This results in $n_2+\cdots+n_{k-1}$ estimations.
2. The $\sum_{j\in\mathcal{I}_i}\ell_{i,j}[\nu_{i,j}]_+$ term for each $i=2,\dots,k-1$, computed at each intermediary layer when computing the bounds. This results in $n_3+2n_4+\cdots+(k-3)n_{k-1}$.

In total, this is $n_2+2n_3+\cdots+(k-2)n_{k-1}=N$ total estimations. In order to say that *all* of these estimates hold with probability $1-\delta$, we can do the following: we bound each estimate in Equation 3.92 with probability δ/N, and use the union bound over all N estimates. We can then conclude that with probability at most δ, any estimate is not an upper bound, and so with probability $1-\delta$ we have a proper upper bound.

Achieving δ/N tail probability There is a problem here: if δ/N is small, then ϵ becomes large, and the bound gets worse. In fact, since $\epsilon<1$, when k is fixed, there's actually a lower limit to how small δ/N can be.

To overcome this problem, we take multiple samples to reduce the probability. Specifically, instead of directly using the geometric estimator, we use the maximum over multiple geometric estimators

$$\text{maxgeo}(R_1,\dots,R_m)=\max(\text{geo}(R_1),\dots,\text{geo}(R_m)),$$

where R_i are independent Cauchy random matrices. If each one has a tail probability of δ, then the maximum has a tail probability of δ^m, which allows us to get arbitrarily small tail probabilities at a rate exponential in m.

High probability tail bounds for network certificates Putting this altogether, let $\delta>0$, let $N>0$ be the number of estimates needed to calculate a certificate, and let m be the number of geometric estimators to take a maximum over. Then with probability $(1-\delta)$, if we bound the tail probability for each geometric estimate with $\hat{\delta}=\left(\frac{\delta}{N}\right)^{1/m}$, then we have an upper bound on the certificate.

As an example, suppose we use the MNIST network in this dissertation. Then, let $\delta=0.01$, $m=10$, and note that $N=6572$. Then, $\hat{\delta}=0.26$, which we can achieve by using $k=200$ and $\epsilon=0.22$.

3.4.7 Bias reduction with cascading ensembles

A final major challenge of training models to minimize a robust bound on the adversarial loss, is that the robustness penalty acts as a regularization. For example, in a two-layer ReLU network, the robust loss penalizes $\epsilon\|\nu_1\|_1=\epsilon\|W_1D_1W_2\|_1$, which effectively acts as a regularizer on the network with weight ϵ. Because of this, the resulting networks (even those with large representational capacity), are typically overregularized to the point that many filters/weights become identically zero (i.e., the network capacity is not used).

Algorithm 4 Training robust cascade of k networks and making predictions

input: Initialized networks $f_1, \ldots, f_k$, training examples X, y, robust training procedure denoted RobustTrain, test example $x^\star$
for $i = 1, \ldots, k$ **do**
  f_i := RobustTrain(f_i, X, y) *// Train network*
  // remove certified examples from dataset
  $X, y := \{x_i, y_i : J(x, g(e_{f(x_i)} - e_{y^{targ}})) > 0, \quad \forall y^{targ} \neq f(x_i)\}$
end for
for $i = 1, \ldots, k$ **do**
  if $f_i(x^\star) \neq f_1(x^\star)$ **then**
    output: no certificate
  end if
  if $J(x, g(e_{f_i(x^\star)} - e_{y^{targ}})) < 0 \;\; \forall y^{targ} \neq f_i(x^\star)$ **then**
    output: $f_i(x^\star)$ *// return label if certified*
  end if
end for
output: no certificate

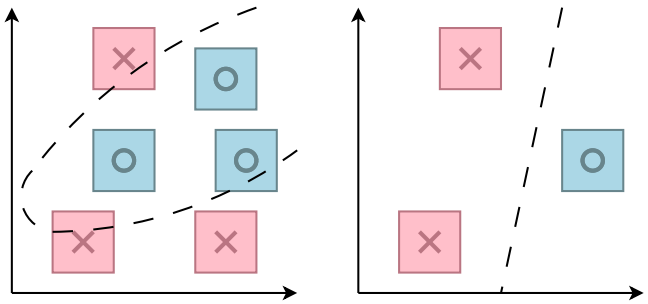


Figure 3.17: An example of a two stage cascade. The first model on the left can only robustly classify three of the datapoints. After removing the certified examples, the remaining examples can now easily be robustly classified by a second stage classifier.

To address this point, we advocate for using a robust *cascade* of networks: that is, we train a sequence of robust classifiers, where later elements of the cascade are trained (and evaluated) *only on those examples that the previous elements of the cascade cannot certify* (i.e., those examples that lie within ϵ of the decision boundary).

The full algorithm for constructing cascades is shown in Algorithm 4. To illustrate the use of the cascade, Figure 3.17 shows a two stage cascade on a few data points in two dimensional space. The boxes denote the adversarial ball around each example, and if the decision boundary is outside of the box, the example is certified.

Table 3.4: Number of hidden units, parameters, and time per epoch for various architectures.

Model	Dataset	# hidden units	# parameters	Time (s) / epoch
Small	MNIST	4804	166406	74
	CIFAR	6244	214918	48
Large	MNIST	28064	1974762	667
	CIFAR	62464	2466858	466
Resnet	MNIST	82536	3254562	2174
	CIFAR	107496	4214850	1685

3.5 Experiments for scaling provable defenses

We evaluate the techniques in this section on two main datasets: MNIST digit classification [LeCun et al., 1998] and CIFAR10 image classification [Krizhevsky, 2009].[4] We test on a variety of deep and wide convolutional architectures, with and without residual connections. All code for these experiments is available at `https://github.com/locuslab/convex_adversarial/`. The small network is the same as that used in [Wong and Kolter, 2017], with two convolutional layers of 16 and 32 filters and a fully connected layer of 100 units. The large network is a scaled up version of it, with four convolutional layers with 32, 32, 64, and 64 filters, and two fully connected layers of 512 units. The residual networks use the same structure used by [Zagoruyko and Komodakis, 2016] with 4 residual blocks with 16, 16, 32, and 64 filters. Similar to prior work, in all of our models we use strided convolutional layers with 4 by 4 kernels to downsample. When downsampling is not needed, we use 3 by 3 kernels without striding. We highlight a subset of the results in Table 3.5, and briefly describe a few key observations below. All results except where otherwise noted use random projection of 50 dimensions.

For all MNIST experiments, we use the Adam optimizer with a learning rate of 0.001 with a batch size of 50. We schedule ϵ starting from 0.01 to the desired value over the first 20 epochs, after which we decay the learning rate by a factor of 0.5 every 10 epochs for a total of 60 epochs.

For all CIFAR10 experiments, we use the SGD optimizer with a learning rate of 0.05 with a batch size of 50. We schedule ϵ starting from 0.001 to the desired value over the first 20 epochs, after which we decay the learning rate by a factor of 0.5 every 10 epochs for a total of 60 epochs.

3.5.1 Scaled and cascaded models for MNIST and CIFAR10 for ℓ_∞ provable robustness

For the different data sets and models, the final robust and nominal test errors are given in Table 3.5. We emphasize that in all cases we report the *robust test error*, that is, our *upper bound* on the possible test set error that the classifier can suffer under *any* norm-bounded attack (thus, considering different empirical attacks is orthogonal to our main presentation and not something

[4]We fully realize the irony of a section with "scaling" in the title that currently maxes out on CIFAR10 experiments. But we emphasize that when it comes to certifiably robust networks, the networks we consider here, as we illustrate below in Table 3.4, are more than an order of magnitude larger than those that have been considered previously in the literature. Thus, our emphasis is really on the potential scaling properties of these approaches rather than large-scale experiments on e.g. ImageNet sized data sets.

Table 3.5: Results on MNIST, and CIFAR10 with small networks, large networks, residual networks, and cascaded variants.

			Single model error		Cascade error	
Dataset	**Model**	**Epsilon**	**Robust**	**Standard**	**Robust**	**Standard**
MNIST	Small, Exact	0.1	4.48%	1.26%	-	-
MNIST	Small	0.1	4.99%	1.37%	**3.13%**	**3.13%**
MNIST	Large	0.1	**3.67%**	**1.08%**	3.42%	3.18%
MNIST	Small	0.3	**43.10%**	14.87%	**33.64%**	**33.64%**
MNIST	Large	0.3	45.66%	**12.61%**	41.62%	35.24%
CIFAR10	Small	2/255	52.75%	38.91%	39.35%	39.35%
CIFAR10	Large	2/255	46.59%	**31.28%**	38.84%	36.08%
CIFAR10	Resnet	2/255	**46.11%**	31.72%	**36.41%**	**35.93%**
CIFAR10	Small	8/255	79.25%	72.24%	71.71%	71.71%
CIFAR10	Large	8/255	83.43%	80.56	79.24%	79.14%
CIFAR10	Resnet	8/255	**78.22%**	**71.33%**	**70.95%**	**70.77%**

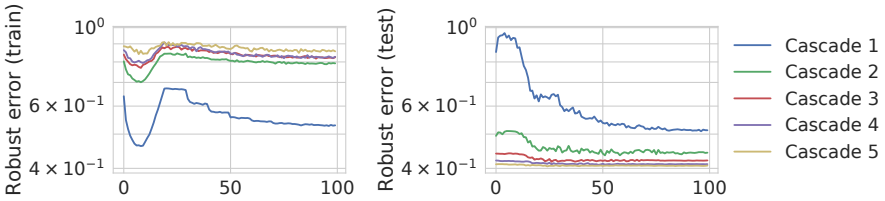


Figure 3.18: Robust error curves as we add models to the cascade for the CIFAR10 dataset on a small model. The ϵ value for training is scheduled to reach 2/255 after 20 epochs. The training curves are for each individual model, and the testing curves are for the whole cascade up to the stage.

that we include, as we are focused on verified performance). As we are focusing on the particular random projections discussed above, all experiments consider attacks with bounded ℓ_∞ norm, plus the ReLU networks highlighted above. On MNIST, the (non-cascaded) large model reaches a final robust error of 3.7% for $\epsilon = 0.1$, and the best cascade reaches 3.1% error. This contrasts with the best previous bound of 5.8% robust error for this epsilon, from [Wong and Kolter, 2017]. On CIFAR10, the ResNet model achieves 46.1% robust error for $\epsilon = 2/255$, and the cascade lowers this to 36.4% error. In contrast, the previous best *verified* robust error for this ϵ, from [Dvijotham et al., 2018b], was 80%. While the robust error is naturally substantially higher for $\epsilon = 8/255$ (the amount typically considered in empirical works), we are still able to achieve 71% provable robust error; for comparison, the best *empirical* robust performance against current attacks is 53% error at $\epsilon = 8/255$ [Madry et al., 2017], and most heuristic defenses have been broken to beyond this error [Athalye et al., 2018a].

Effect of cascaded networks In all cases, cascading the models is able to improve the robust error performance, sometimes substantially, for instance decreasing the robust error on CIFAR10

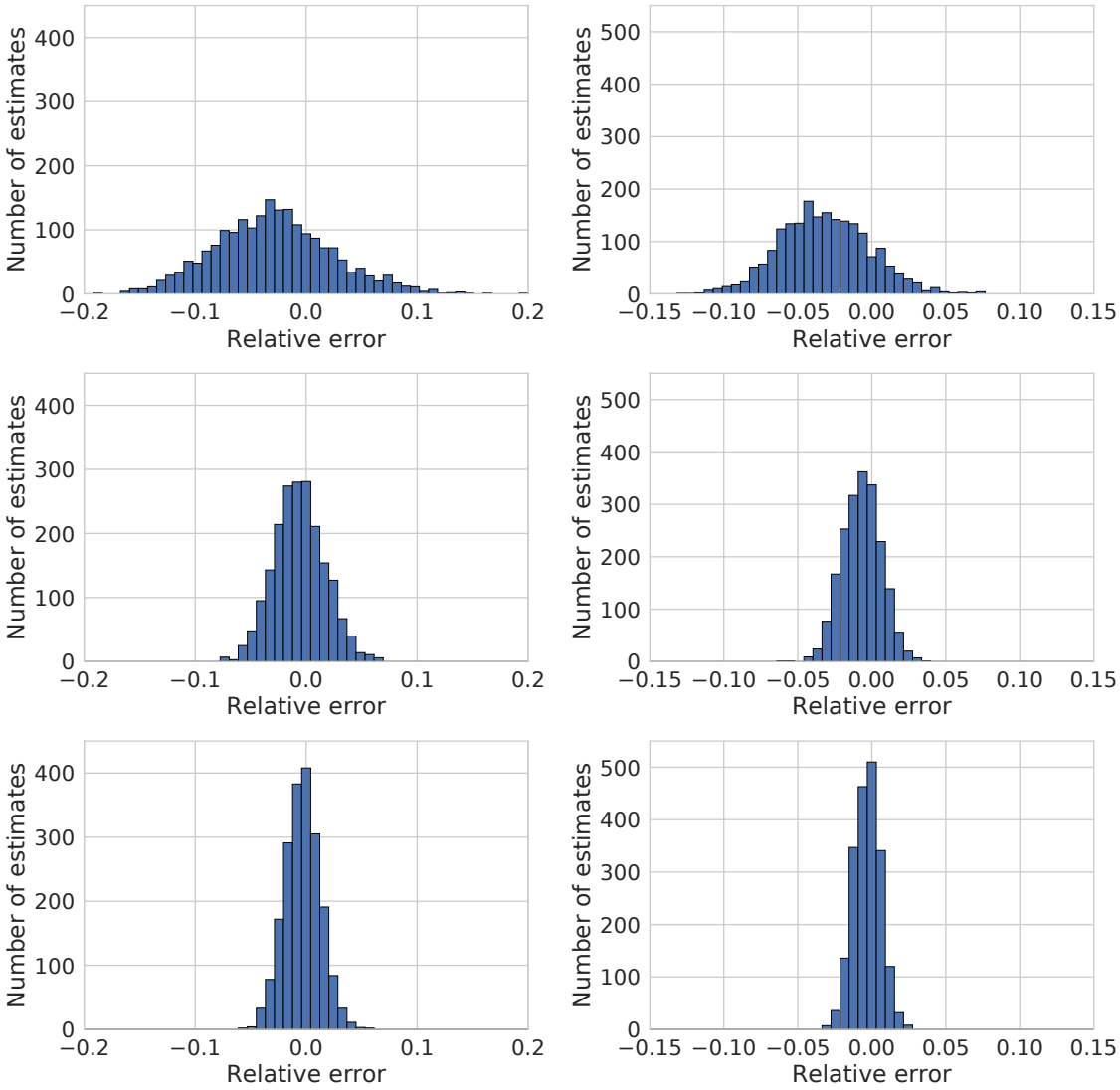


Figure 3.19: Histograms of the relative error of the median estimator for 10 (top), 50 (middle), and 100 (bottom) projections, for a (left) random and (right) robustly trained convolutional layer.

from 46.1% to 36.4% for $\epsilon = 2/255$. However, this comes at a cost as well: the *nominal* error *increases* throughout the cascade (this is to be expected, since the cascade essentially tries to force the robust and nominal errors to match). Thus, there is substantial value to both improving the single-model networks *and* integrating cascades into the prediction.

3.5.2 Exploring the effects of random projections in robust training

In this section, we discuss the empirical quality and speedup of the median estimator for ℓ_1 estimation (for a more theoretical understanding, we direct the reader to Li et al. [2007]).

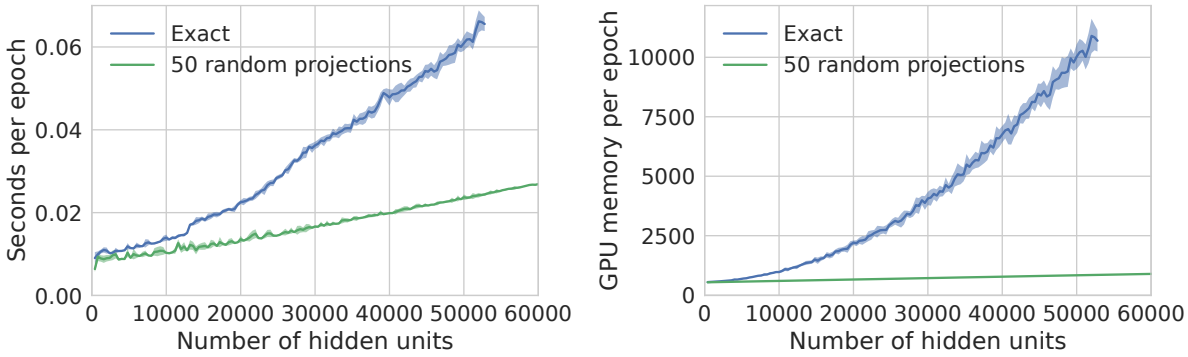


Figure 3.20: Timing (top) and memory in MB (bottom) plots for a single 3 by 3 convolutional layer to evaluate 10 MNIST sized examples with minibatch size 1, averaged over 10 runs. The number of hidden units is varied by increasing the number of filters. On a single Titan X, the exact method runs out of memory at 52,800 hidden units, whereas the random projections scales linearly at a slope of 2.26×10^{-7} seconds per hidden unit, up to 0.96 seconds for 4,202,240 hidden units.

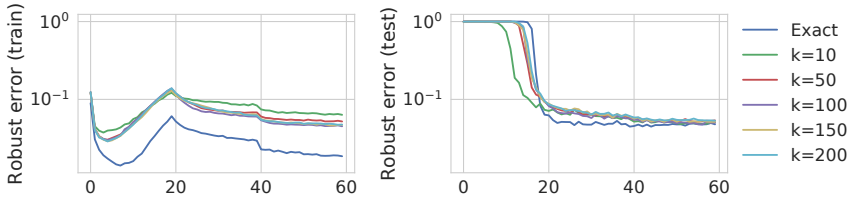


Figure 3.21: Training and testing robust error curves over epochs on the MNIST dataset using k projection dimensions. The ϵ value for training is scheduled from 0.01 to 0.1 over the first 20 epochs. The projections force the model to generalize over higher variance, reducing the generalization gap.

Estimation quality In Figure 3.19, we plot the relative error of the median estimator for varying dimensions on both an untrained and a trained convolutional layer, and see that regardless of whether the model is trained or not, the distribution of the estimate is normally distributed with decreasing variance for larger projections, and without degenerate cases. This matches the theoretical results derived in Li et al. [2007].

Improvements in time and memory usage In Figure 3.20, we benchmark the time and memory usage on a convolutional MNIST example to demonstrate the performance improvements. While the exact bound takes time and memory that is quadratic in the number of hidden units, the median estimator is instead linear, allowing it to scale up to millions of hidden units whereas the exact bound runs out of memory out at 50,280 hidden units.

Varying the number of projections In the MNIST dataset (the only data set where it is trivial to run exact training without projection), we have evaluated our approach using different projection dimensions as well as exact training (i.e., without random projections). We note that

Table 3.6: Results on different widths and depths for MNIST

Dataset	Model	Epsilon	Robust error	Error
MNIST	Wide(1)	0.1	6.51%	2.27%
MNIST	Wide(2)	0.1	5.46%	1.55%
MNIST	Wide(4)	0.1	4.94%	1.33%
MNIST	Wide(8)	0.1	4.79%	1.32%
MNIST	Wide(16)	0.1	5.27%	1.36%
MNIST	Deep(1)	0.1	5.28%	1.78%
MNIST	Deep(2)	0.1	4.37%	1.28%
MNIST	Deep(3)	0.1	4.20%	1.15%

using substantially lower projection dimension does not have a significant impact on the test error. This fact is highlighted in Figure 3.21. Using the same convolutional architecture used by Wong and Kolter [2017], which previously required gigabytes of memory and took hours to train, it is sufficient to use only 10 random projections to achieve comparable test error performance to training with the exact bound. Each training epoch with 10 random projections takes less than a minute on a single GeForce GTX 1080 Ti graphics card, while using less than 700MB of memory, achieving significant speedup and memory reduction over Wong and Kolter [2017].

3.5.3 The effect of increased width and depth

We use a parameter k to control the width and depth of the architectures used to measure the effect of increasing width and depth. The Wide(k) networks have two convolutional layers of $4 \times k$ and $8 \times k$ filters followed by a $128 \times k$ fully connected layer. The Deep(k) networks have k convolutional filters with 8 filters followed by k convolutional filters with 16 filters.

We find that increasing the capacity of the model by simply making the network deeper and wider on MNIST is able boost performance. However, when the model becomes overly wide, the test robust error performance begins to degrade due to overfitting. These results are shown in Table 3.6.

3.5.4 Large and cascaded models for MNIST and CIFAR10 for ℓ_2 provable robustness

We run similar experiments for ℓ_2 perturbations on the input instead of ℓ_∞ perturbations, which amounts to replacing the ℓ_1 norm in the objective with the ℓ_2 norm. This can be equivalently scaled using random normal projections [Vempala, 2005] instead of random Cauchy projections. We use the same network architectures as before, and pick ϵ_2 such that the volume of an ℓ_2 ball with radius ϵ_2 is approximately the same as the volume of an ℓ_∞ ball with radius ϵ_∞. A simple conversion (an overapproximation within a constant factor) is:

$$\epsilon_2 = \sqrt{\frac{d}{\pi}}\epsilon_\infty.$$

Table 3.7: Results on MNIST, and CIFAR10 with small networks, large networks, residual networks, and cascaded variants for ℓ_2 perturbations.

			Single model error		Cascade error	
Dataset	**Model**	**Epsilon**	**Robust**	**Standard**	**Robust**	**Standard**
MNIST	Small, Exact	1.58	56.48%	**11.86%**	**24.42%**	**19.57%**
MNIST	Small	1.58	56.32%	13.11%	25.34%	20.93%
MNIST	Large	1.58	**55.47%**	11.88%	26.16%	24.97%
CIFAR10	Small	36/255	53.73%	44.72%	50.13%	48.64%
CIFAR10	Large	36/255	49.40%	40.24%	**41.36%**	**41.16%**
CIFAR10	Resnet	36/255	**48.04%**	**38.80%**	41.44%	41.28%

For MNIST, we take an equivalent volume to $\epsilon_\infty = 0.1$. This ends up being $\epsilon_2 = 1.58$, and note that within the dataset, the minimum ℓ_2 distance between any two digits is at least 3.24, so ϵ_2 is roughly half of the minimum distance between any two digits. For CIFAR we take an equivalent volume to $\epsilon_\infty = 2/255$, which ends up being $\epsilon_2 = 36/255$.

The results for the complete suite of experiments are in Table 3.7, and we get similar trends in robustness for larger and cascaded models to that of ℓ_∞ perturbations.

3.6 Discussion

In this chapter, we started from using convex outer bounds utilizing linear programming, and ended with a general methodology for deriving dual networks from compositions of dual layers based on the methodology of conjugate functions to train classifiers that are provably robust to adversarial attacks. Importantly, the methodology is linearly scalable for ReLU based networks against ℓ_∞ norm bounded attacks, making it possible to train large scale, provably robust networks that were previously out of reach, and the obtained bounds can be improved further with model cascades.

Chapter 4

Adversarially robust learning

In this chapter, we switch gears from provable defenses and study more generally the optimization properties of the *robust learning* problem, or the process of training a robust classifier. While the convergence and generalization properties of training neural networks in the standard setting has received much attention both empirically and theoretically, the process of training adversarially robust models can behave quite differently from standard training and sometimes exhibit properties which go against common intuition. By understanding and leveraging these properties of robust learning, we can drastically accelerate and improve the robust training process to converge more quickly to more robust models. In this chapter, we focus our study on the inner maximization problem of adversarial training (the strength of the adversary), as well as general properties regarding overfitting in adversarial training.

The robust learning problem is often seen as fairly difficult, in part due to the number of defenses which have been broken over the years and also due to its computational cost. Even successful defenses such as adversarial training and provable defenses are often much more computationally expensive than standard training, limiting their ability to be applied to much larger networks and motivating the need for cheaper adversarial defenses which are still effective. In the first half of this section, we argue that adversarial training may not be as hard as has been suggested by this past line of work. In particular, we revisit one of the the *first* proposed methods for adversarial training, using the Fast Gradient Sign Method (FGSM) to add adversarial examples to the training process [Goodfellow et al., 2015]. Although this approach has long been dismissed as ineffective [Tramèr et al., 2017], we show that by simply introducing random initialization points, FGSM-based training can be *as effective as projected gradient descent based training* while being an order of magnitude more efficient.

Moreover, FGSM adversarial training (and to a lesser extent, other adversarial training methods) can be drastically accelerated using standard techniques for efficient training of deep networks, including e.g. cyclic learning rates [Smith and Topin, 2018], mixed-precision training [Micikevicius et al., 2017], and other similar techniques. These are common techniques from top submissions to the DAWNBench competition [Coleman et al., 2017] that are used to train CIFAR10 and ImageNet classifiers in mere minutes and hours using only a modest amount of computational resources. The resulting method has extremely few free parameters to tune, and can be easily adapted to most training procedures. We further identify a failure mode that we call "catastrophic overfitting", which may have caused previous attempts at FGSM adversarial train-

ing to fail against PGD-based attacks. The end result is that, with these approaches, we are able to train (empirically) robust classifiers with the robustness of PGD-based training at the speed of standard training. Despite the conventional wisdom, this suggests that adversarially robust training is not actually more challenging than standard training of deep networks, since rather coarse approximations to the inner maximization can be sufficient for successful robust optimization.

In the latter half of this chapter, we take a closer look at how the concept of overfitting manifests in adversarially robust training more generally beyond the catastrophic overfitting observed in the FGSM setting. Both regularization [Friedman et al., 2001] and early stopping [Strand, 1974] have been well-studied in classical statistical settings to reduce overfitting and improve generalization, and connections between the two have been established in various settings such as in kernel boosting algorithms [Wei et al., 2017], least squares regression [Ali et al., 2018], and strongly convex problems [Suggala et al., 2018]. Although ℓ_2 regularization (also known as weight decay) is commonly used for training deep networks [Krogh and Hertz, 1992], early stopping is less commonly used despite being studied as an implicit regularizer for controlling model complexity for neural networks at least 30 years ago [Morgan and Bourlard, 1990].[1]

Indeed, one of the surprising characteristics of deep learning is the relative *lack* of overfitting seen in practice [Zhang et al., 2016], where deep learning models can often be trained to zero training error without incurring the standard bias-variance trade-off from classical statistical learning theory. Despite effectively memorizing the training set, in many typical deep learning settings there are not any detrimental effects on the generalization performance [Neyshabur et al., 2017]. Consequently, it is now standard practice in many modern deep learning tasks to train for as long as possible and use large overparameterized models, since test set performance typically continues to improve past the point of dataset interpolation in what is known as "double descent" generalization [Belkin et al., 2019, Nakkiran et al., 2019], and remains such a hallmark of deep learning practice that it is often taken for granted.

Our key finding is that, unlike in traditional deep learning, *overfitting is a dominant phenomenon in adversarially robust training of deep networks.* That is, adversarially robust training has the property that, after a certain point, further training will continue to substantially decrease the robust training loss of the classifier, while increasing the robust test loss. This is shown, for instance, in Figure 4.1 for adversarial training on CIFAR10, where the robust test error dips immediately after the first learning rate decay, and only increases beyond this point. This phenomenon, which we refer to as "robust overfitting", can be observed on multiple datasets, different adversarial training algorithms, and various threat models.

Motivated by this initial finding, we further study and diagnose this problem. We begin by emphasizing that virtually all the recent gains in adversarial performance from newer algorithms beyond simple projected gradient descent (PGD) based adversarial training [Mosbach et al., 2018, Xie et al., 2019, Yang et al., 2019, Zhang et al., 2019b] can be attained by a much simpler approach: using early stopping. Specifically, by just using an earlier checkpoint, the robust performance of adversarially trained deep networks can be drastically improved, to the point where *the original PGD-based adversarial training method can actually achieve the same*

[1] It is common practice in deep learning to save the best checkpoint which can be seen as early stopping. However, in the standard setting, the test loss tends to gradually improve over training, and so the best checkpoint tends to just select the best performance at the end of training, rather than stopping before training loss has converged.

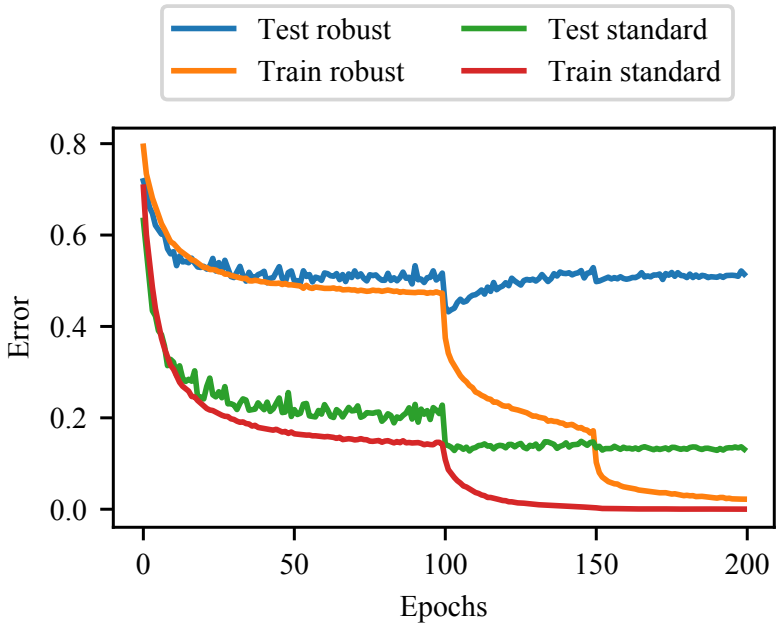


Figure 4.1: The learning curves for a robustly trained model replicating the experiment done by Madry et al. [2017] on CIFAR10. The curves demonstrate "robust overfitting"; shortly after the first learning rate decay the model momentarily attains 43.2% robust error, and is actually more robust than the model at the end of training, which only attains 51.4% robust test error against a 10-step PGD adversary for ℓ_∞ radius of $\epsilon = 8/255$. The learning rate is decayed at 100 and 150 epochs.

robust performance as state-of-the-art methods.

Finally, we study various empirical properties of overfitting for adversarially robust training and how they relate to standard training, such as how various learning rate schedules affect the prevalence of robust overfitting and the resulting impacts on model performance, and how known connections between hypothesis class size and generalization in deep networks translate to the robust setting. We wrap up this chapter by investigating several techniques, from both classical statistics and modern deep learning, for preventing robust overfitting. For example, Dropout is a commonly used stochastic regularization technique that randomly drops units and their connections from the network during training [Srivastava et al., 2014] with the intent of preventing complex co-adaptations on the training data. Data augmentation is another technique frequently used when training deep networks that has been empirically shown to reduce overfitting. Cutout [DeVries and Taylor, 2017] is a form of data augmentation that randomly masks out a section of the input during training, which can be considered as augmenting the dataset with occlusions. Another technique known as mixup [Zhang et al., 2017] trains on convex combinations of pairs of data points and their corresponding labels to encourage linear behavior in between data points. Semi-supervised learning methods augment the dataset with unlabeled data, and have been shown to improve generalization when used in the adversarially robust setting [Alayrac et al., 2019, Carmon et al., 2019, Zhai et al., 2019]. Ultimately, while these methods can mit-

Algorithm 5 PGD adversarial training for T epochs, given some radius ϵ, adversarial step size α and N PGD steps and a dataset of size M for a network f_θ

for $t = 1 \dots T$ **do**
  for $i = 1 \dots M$ **do**
    // Perform PGD adversarial attack
    $\delta = 0$ *// or randomly initialized*
    for $j = 1 \dots N$ **do**
      $\delta = \delta + \alpha \cdot \text{sign}(\nabla_\delta \ell(f_\theta(x_i + \delta), y_i))$
      $\delta = \max(\min(\delta, \epsilon), -\epsilon)$
    end for
    $\theta = \theta - \nabla_\theta \ell(f_\theta(x_i + \delta), y_i)$ *// Update model weights with some optimizer, e.g. SGD*
  end for
end for

igate robust overfitting to varying degrees, when trained to convergence, *we find that no other approach to combating robust overfitting performs better than simple early stopping*.

4.1 Fast adversarial training

As discussed in Chapter 2, adversarial training is a method for learning networks which are robust to adversarial attacks, typically with a PGD adversary (for example as shown in Algorithm 5 for the ℓ_∞ threat model). Note that the number of gradient computations for a PGD adversary is proportional to $O(MN)$ in a single epoch, where M is the size of the dataset and N is the number of steps taken by the PGD adversary. This is N times greater than standard training (which has $O(M)$ gradient computations per epoch), and so adversarial training is typically N times slower than standard training.

"Free" adversarial training To get around this slowdown of a factor of N, Shafahi et al. [2019] instead propose "free" adversarial training. This method takes FGSM steps with full step sizes $\alpha = \epsilon$ followed by updating the model weights for N iterations on the same minibatch (also referred to as "minibatch replays"). The algorithm is summarized in Algorithm 6. Note that perturbations are not reset between minibatches. To account for the additional computational cost of minibatch replay, the total number of epochs is reduced by a factor of N to make the total cost equivalent to T epochs of standard training. Although "free" adversarial training is faster than the standard PGD adversarial training, it is not as fast as we'd like: Shafahi et al. [2019] need to run over 200 epochs in over 10 hours to learn a robust CIFAR10 classifier and two days to learn a robust ImageNet classifier, whereas standard training can be accomplished in minutes and hours for the same respective tasks.

To speed up adversarial training and move towards the state of the art in fast standard training methods, we first highlight the main empirical contribution of the paper: that FGSM adversarial training combined with random initialization is just as effective a defense as PGD-based training. Following this, we discuss several techniques from the DAWNBench competition [Coleman

Algorithm 6 "Free" adversarial training for T epochs, given some radius ϵ, N minibatch replays, and a dataset of size M for a network f_θ

$\delta = 0$
// Iterate T/N times to account for minibatch replays and run for T total epochs
for $t = 1 \dots T/N$ **do**
  for $i = 1 \dots M$ **do**
    // Perform simultaneous FGSM adversarial attack and model weight updates T times
    for $j = 1 \dots N$ **do**
      // Compute gradients for perturbation and model weights simultaneously
      $\nabla_\delta, \nabla_\theta = \nabla \ell(f_\theta(x_i + \delta), y_i)$
      $\delta = \delta + \epsilon \cdot \text{sign}(\nabla_\delta)$
      $\delta = \max(\min(\delta, \epsilon), -\epsilon)$
      $\theta = \theta - \nabla_\theta$ *// Update model weights with some optimizer, e.g. SGD*
    end for
  end for
end for

Algorithm 7 FGSM adversarial training for T epochs, given some radius ϵ, N PGD steps, step size α, and a dataset of size M for a network f_θ

for $t = 1 \dots T$ **do**
  for $i = 1 \dots M$ **do**
    // Perform FGSM adversarial attack
    $\delta = \text{Uniform}(-\epsilon, \epsilon)$
    $\delta = \delta + \alpha \cdot \text{sign}(\nabla_\delta \ell(f_\theta(x_i + \delta), y_i))$
    $\delta = \max(\min(\delta, \epsilon), -\epsilon)$
    $\theta = \theta - \nabla_\theta \ell(f_\theta(x_i + \delta), y_i)$ *// Update model weights with some optimizer, e.g. SGD*
  end for
end for

et al., 2017] that are applicable to all adversarial training methods, which reduce the total number of epochs needed for convergence with cyclic learning rates and further speed up computations with mixed-precision arithmetic.

4.1.1 Revisiting FGSM adversarial training

Despite being quite similar to FGSM adversarial training, free adversarial training is empirically robust against PGD attacks whereas FGSM adversarial training is not believed to be robust. To analyze why, we identify a key difference between the methods: a property of free adversarial training is that the perturbation from the previous iteration is used as the initial starting point for the next iteration. However, there is little reason to believe that an adversarial perturbation for a previous minibatch is a reasonable starting point for the next minibatch. As a result, we hypothesize that the main benefit comes from simply starting from a non-zero initial perturbation.

Table 4.1: Standard and robust performance of various adversarial training methods on CIFAR10 for $\epsilon = 8/255$ and their corresponding training times

Method	Standard accuracy	PGD ($\epsilon = 8/255$)	Time (min)
FGSM + DAWNBench			
+ zero init	85.18%	0.00%	12.37
+ early stopping	71.14%	38.86%	7.89
+ previous init	86.02%	42.37%	12.21
+ random init	85.32%	44.01%	12.33
+ $\alpha = 10/255$ step size	83.81%	46.06%	12.17
+ $\alpha = 16/255$ step size	86.05%	0.00%	12.06
+ early stopping	70.93%	40.38%	8.81
"Free" ($m = 8$) [Shafahi et al., 2019][2]	85.96%	46.33%	785
+ DAWNBench	78.38%	46.18%	20.91
PGD-7 [Madry et al., 2017][3]	87.30%	45.80%	4965.71
+ DAWNBench	82.46%	50.69%	68.8

In light of this difference, our approach is to use FGSM adversarial training with random initialization for the perturbation, as shown in Algorithm 7. We find that, in contrast to what was previously believed, this simple adjustment to FGSM adversarial training can be used as an effective defense on par with PGD adversarial training. Crucially, we find that starting from a non-zero initial perturbation is the primary driver for success, regardless of the actual initialization. In fact, both starting with the previous minibatch's perturbation or initializing from a uniformly random perturbation allow FGSM adversarial training to succeed at being robust to full-strength PGD adversarial attacks. Note that randomized initialization for FGSM is not a new idea and was previously studied by Tramèr et al. [2017]. Crucially, Tramèr et al. [2017] use a different, more restricted random initialization and step size, which does not result in models robust to full-strength PGD adversaries. A more detailed comparison of their approach with ours is done later in Section 4.1.4.

To test the effect of initialization in FGSM adversarial training, we train several models to be robust at a radius $\epsilon = 8/255$ on CIFAR10, starting with the most "pure" form of FGSM, which takes steps of size $\alpha = \epsilon$ from a zero-initialized perturbation. The results, given in Table 4.1, are consistent with the literature, and show that the model trained with zero-initialization is not robust against a PGD adversary. However, surprisingly, simply using a random or previous-minibatch initialization instead of a zero initialization actually results in reasonable robustness levels (with random initialization performing slightly better) that are comparable to both free and PGD adversarial training methods. The adversarial accuracies in Table 4.1 are calculated using a PGD adversary with 50 iterations, step size $\alpha = 2/255$, and 10 random restarts.

[2]As reported by Shafahi et al. [2019] using a different network architecture and an adversary with 20 steps and 10 restarts, which is strictly weaker than the adversary used in this paper.

[3]As reported by Madry et al. [2017] using a different network architecture and an adversary and an adversary with 20 steps and no restarts, which is strictly weaker than the adversary used in this paper

Table 4.2: Training parameters used for the DAWNBench experiments of Table 4.1

Parameter	FGSM	PGD	Free
Epochs	30	40	96
Max learning rate	0.2	0.2	0.04

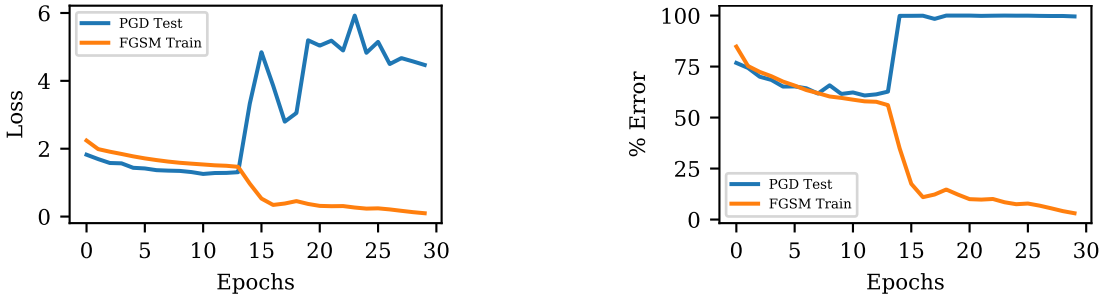


Figure 4.2: Learning curves for FGSM adversarial training plotting the training loss and error rates incurred by an FGSM and PGD adversary when trained with zero-initialization FGSM at $\epsilon = 8/255$, depicting the catastrophic overfitting where PGD performance suddenly degrades while the model overfits to the FGSM performance.

Experimental setup For all methods, we use a batch size of 128, and SGD optimizer with momentum 0.9 and weight decay $5 * 10^{-4}$. We report the average results over 3 random seeds. The remaining parameters for learning rate schedules and number of epochs for the DAWNBench experiments are in Table 4.2. For runs using early-stopping, we use a 5-step PGD adversary with 1 restart on 1 training minibatch to detect overfitting to the FGSM adversaries, as described in more detail in Section 4.1.2.

Computational complexity A second key difference between FGSM and free adversarial training is that the latter uses a single backwards pass to compute gradients for both the perturbation and the model weights while repeating the same minibatch m times in a row, called "minibatch replay". In comparison, the FGSM adversarial training does not need to repeat minibatches, but needs two backwards passes to compute gradients separately for the perturbation and the model weights. As a result, the computational complexity for an epoch of FGSM adversarial training is not truly free and is equivalent to two epochs of standard training.

4.1.2 Catastrophic overfitting

While FGSM adversarial training works in the context of this section, many other researchers have tried and failed to have FGSM adversarial training work. In addition to using a zero initialization or too large of a step size as seen in Table 4.1, other design decisions (like specific learning rate schedules or numbers of epochs) for the training procedure can also make it more likely for FGSM adversarial training to fail. However, all of these failure modes result in what we call "catastrophic overfitting", where the robust accuracy with respect to a PGD adversarial

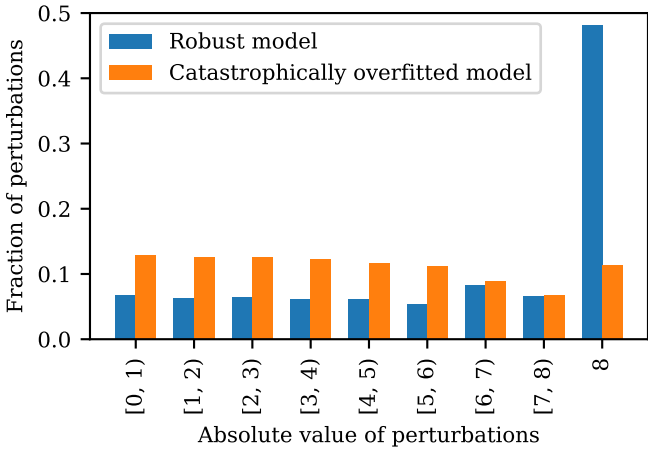


Figure 4.3: Histogram of the resulting perturbations from a PGD adversary for each feature for a successfully trained robust CIFAR10 model and a catastrophically overfitted CIFAR10 model.

suddenly and drastically drops to 0% (on the training data). What was previously a reasonably robust model will quickly transform into a non-robust model over the span of a couple epochs. This phenomenon can be seen in Figure 4.2 which plots the learning curves for standard, vanilla FGSM adversarial training from zero-initialization.

Indeed, one of the reasons for this failure may lie in the lack of diversity in adversarial examples generated by these FGSM adversaries. For example, using a zero initialization or using the random initialization scheme from Tramèr et al. [2017] will result in adversarial examples whose features have been perturbed by $\{-\epsilon, 0, \epsilon\}$, and so the network learns a decision boundary which is robust only at these perturbation values. This can be verified by running a PGD adversarial attack on models which have catastrophically overfitted, where the perturbations tend to be more in between the origin and the boundary of the threat model (relative to a non-overfitted model, which tends to have perturbations near the boundary), as seen in Figure 4.3.

Early stopping Catastrophic overfitting can be easily detected by evaluating the PGD performance on a small subset of the training data, as the catastrophic failure will result in 0% robust accuracy for a PGD adversary on the training set. Consequently, these alternative versions of FGSM adversarial training can be salvaged to some degree by checking an early stopping criteria (PGD accuracy) on the training set. In practice, we find that this can be a simple as a single minibatch with a 5-step PGD adversary, which can be quickly checked at the end of the epoch, and the recovered results for some of these failure modes are shown in Table 4.1. If robust accuracy with respect to this adversary suddenly drops, then we have catastrophic overfitting. Using a PGD adversary on a training minibatch to detect catastrophic overfitting, we can early stop to avoid catastrophic overfitting and achieve a reasonable amount of robust performance.

4.1.3 Effect of step size for FGSM adversarial training

Note that an FGSM step with size $\alpha = \epsilon$ from a non-zero initialization is not guaranteed to lie on the boundary of the ℓ_∞ ball, and so this defense could potentially be seen as too weak. In

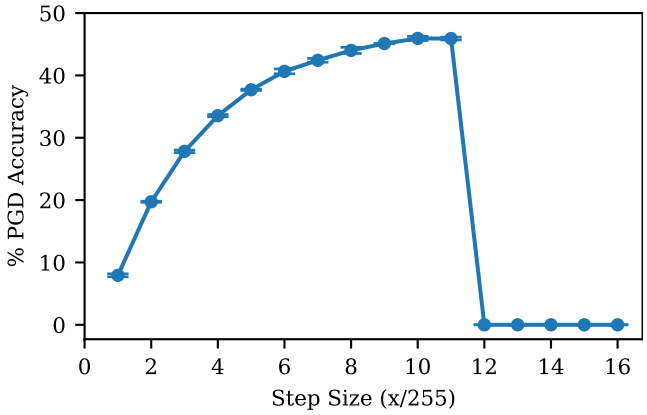


Figure 4.4: Robust test performance of FGSM adversarial training over different step sizes for $\epsilon = 8/255$.

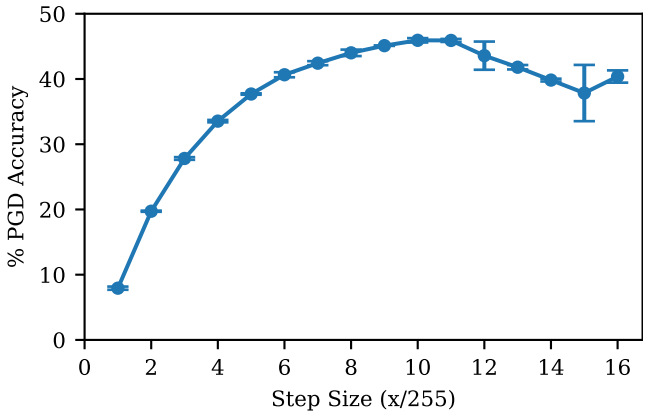


Figure 4.5: Robust test performance of FGSM adversarial training over different step sizes for $\epsilon = 8/255$ with early stopping to avoid catastrophic overfitting.

this section we test the effect of step size on the performance of FGSM adversarial training. We plot the mean and standard error of the robust accuracy for models trained for 30 epochs over 3 random seeds in Figure 4.4, and vary the step size from $\alpha = 1/255$ to $\alpha = 16/255$.

We find that we get increasing robust performance as we increase the step size up to $\alpha = 10/255$, which is on par with the best reported result from free adversarial training. Beyond this, we see no further benefit, or find that the model is prone to overfitting to the adversarial examples, since the large step size forces the model to overfit to the boundary of the perturbation region. For example, forcing the resulting perturbation to lie on the boundary with a step size of $\alpha = 2\epsilon$ results in catastrophic failure: it does not produce a model robust to adversarial attacks.

Although step sizes larger than $11/255$ result in 0% robust accuracy, we can salvage these runs to some degree with early stopping as described in Section 4.1.2. By validating robustness with a simple PGD adversary on the training set, we can catch the model at its peak performance before overfitting, showing that FGSM adversarial training with larger step sizes can actually achieve some degree of robust accuracy, as seen in Figure 4.5.

Table 4.3: Ablation study showing the performance of R+FGSM from Tramèr et al. [2017] and the various changes for the version of FGSM adversarial training done in this paper, over 10 random seeds.

Method	Step size	Initialization	Robust accuracy
R+FGSM [Tramèr et al., 2017]	0.15	Hypercube(0.15)	$34.58 \pm 36.06\%$
R+FGSM (+full step size)	0.30	Hypercube(0.15)	$26.53 \pm 32.48\%$
R+FGSM (+uniform init.)	0.15	Uniform(0.3)	$72.92 \pm 10.40\%$
Uniform + full (ours)	0.30	Uniform(0.3)	$86.21 \pm 00.75\%$

4.1.4 A direct comparison to R+FGSM from Tramèr et al. [2017]

While a randomized version of FGSM adversarial training was proposed by Tramèr et al. [2017], it was not shown to be as effective as adversarial training against a PGD adversary. Here, we note the two main differences between our approach and that of Tramèr et al. [2017].

1. The random initialization used is different. For a data point x, we initialize with the uniform distribution in the entire perturbation region with

   $$x' = x + \text{Uniform}(-\epsilon, \epsilon).$$

   In comparison, Tramèr et al. [2017] instead initialize on the surface of a hypercube with radius $\epsilon/2$ with

   $$x' = x + \frac{\epsilon}{2}\,\text{sign}(\text{Normal}(0, 1)).$$

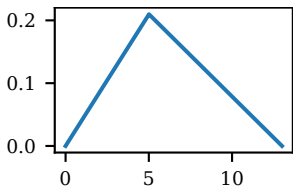
2. The step sizes used for the FGSM step are different. We use a full step size of $\alpha = \epsilon$, whereas Tramèr et al. [2017] use a step size of $\alpha = \epsilon/2$.

To study the effect of these two differences, we run all combinations of either initialization with either step size on MNIST. The results are summarized in Table 4.3.

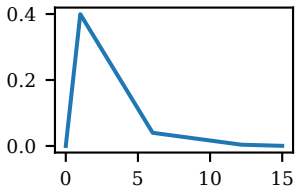
We find that using a uniform initialization adds the greatest marginal improvement to the original R+FGSM attack, while using a full step size doesn't seem to help on its own. Implementing both of these improvements results in the form of FGSM adversarial training presented in this paper. Additionally, note that R+FGSM as done by Tramèr et al. [2017] has high variance in robust performance when done over multiple random seeds, whereas our version of FGSM adversarial training is significantly more consistent and has a very low standard deviation over random seeds.

4.1.5 DAWNBench improvements

Although free adversarial training is of comparable cost per iteration to traditional standard training methods, it is not quite comparable in total cost to more recent advancements in fast methods for standard training. Notably, top submissions to the DAWNBench competition have shown that CIFAR10 and ImageNet classifiers can be trained at significantly quicker times and at much lower cost than traditional training methods. Although some of the submissions can be quite



(a) CIFAR10



(b) ImageNet

Figure 4.6: Cyclic learning rates used for FGSM adversarial training on CIFAR10 and ImageNet over epochs. The ImageNet cyclic schedule is decayed further by a factor of 10 in the second and third phases.

unique in their approaches, we identify two generally applicable techniques which have a significant impact on the convergence rate and computational speed of standard training.

Cyclic learning rate Introduced by Smith [2017] for improving convergence and reducing the amount of tuning required when training networks, a cyclic schedule for a learning rate can drastically reduce the number of epochs required for training deep networks [Smith and Topin, 2018]. A simple cyclic learning rate schedules the learning rate linearly from zero, to a maximum learning rate, and back down to zero (examples can be found in Figure 4.6). Using a cyclic learning rate allows CIFAR10 architectures to converge to benchmark accuracies in tens of epochs instead of hundreds, and is a crucial component of some of the top DAWNBench submissions.

Mixed-precision arithmetic With newer GPU architectures coming with tensor cores specifically built for rapid half-precision calculations, using mixed-precision arithmetic when training deep networks can also provide significant speedups for standard training [Micikevicius et al., 2017]. This can drastically reduce the memory utilization, and when tensor cores are available, also reduce runtime. In some DAWNBench submissions, switching to mixed-precision computations was key to achieving fast training while keeping costs low.

We adopt these two techniques for use in adversarial training, which allows us to drastically reduce the number of training epochs as well as the runtime on GPU infrastructure with tensor cores, while using modest amounts of computational resources. Notably, both of these improvements can be easily applied to existing implementations of adversarial training by adding a few lines of code with very little additional engineering effort, and so are easily accessible by the general research community.

4.2 Experiments for fast adversarial training

To demonstrate the effectiveness of FGSM adversarial training with fast training methods, we run a number of experiments on MNIST, CIFAR10, and ImageNet benchmarks. All CIFAR10 experiments in this paper are run on a single GeForce RTX 2080ti using the PreAct ResNet18 architecture, and all ImageNet experiments are run on a single machine with four GeForce RTX

Table 4.4: Robustness of FGSM and PGD adversarial training on MNIST

Method	Standard accuracy	PGD ($\epsilon = 0.1$)	PGD ($\epsilon = 0.3$)	Verified ($\epsilon = 0.1$)
PGD	99.20%	97.66%	89.90%	96.7%
FGSM	99.20%	97.53%	88.77%	96.8%

Table 4.5: Training parameters used for Figure 4.7

Parameter	FGSM	PGD	Free
Max learning rate	0.2	0.2	0.04

2080tis using the ResNet50 architecture [He et al., 2016a]. Repositories for reproducing all experiments and the corresponding trained model weights are available at `https://github.com/locuslab/fast_adversarial`.

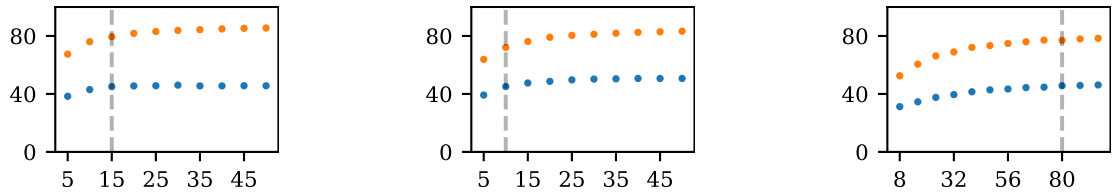
All experiments using FGSM adversarial training in this section are carried out with random initial starting points and step size $\alpha = 1.25\epsilon$ as described in Section 4.1.1. All PGD adversaries used at evaluation are run with 10 random restarts for 50 iterations (with the same hyperparameters as those used by Shafahi et al. [2019] but further strengthened with random restarts). Speedup with mixed-precision was incorporated with the Apex `amp` package at the `O1` optimization level for ImageNet experiments and `O2` without loss scaling for CIFAR10 experiments.[4]

4.2.1 Verified performance on MNIST

Since the FGSM attack is known to be significantly weaker than the PGD attack, it is understandable if the reader is still skeptical of the true robustness of the models trained using this method. To demonstrate that FGSM adversarial training confers real robustness to the model, in addition to evaluating against a PGD adversary, we leverage mixed-integer linear programming (MILP) methods from formal verification to calculate the exact robustness of small, but verifiable models [Tjeng et al., 2018]. We train two convolutional networks with 16 and 32 convolutional filters followed by a fully connected layer of 100 units, the same architecture used by Tjeng et al. [2018]. We use both PGD and FGSM adversarial training at $\epsilon = 0.3$, where the PGD adversary for training has 40 iterations with step size 0.01 as done by Madry et al. [2017]. The exact verification results can be seen in Table 4.4, where we find that FGSM adversarial training confers empirical and verified robustness which is nearly indistinguishable to that of PGD adversarial training on MNIST.[5]

[4]Since CIFAR10 did not suffer from loss scaling problems, we found using the `O2` optimization level without loss scaling for mixed-precision arithmetic to be slightly faster.

[5]Exact verification results at $\epsilon = 0.3$ for both the FGSM and PGD trained models are not possible since the size of the resulting MILP is too large to be solved in a reasonable amount of time. The same issue also prevents us from verifying networks trained on datasets larger than MNIST, which have to rely on empirical tests for evaluating robustness.



(a) FGSM (b) PGD (c) "Free" ($m = 8$)

Figure 4.7: Performance of models trained on CIFAR10 at $\epsilon = 8/255$ with cyclic learning rates and half precision, given varying numbers of epochs across different adversarial training methods. Each point denotes the average model performance over 3 independent runs, where the x axis denotes the number of epochs N the model was trained for, and the y axis denotes the resulting accuracy. The orange dots measure accuracy on natural images and the blue dots plot the empirical robust accuracy on adversarial images. The vertical dotted line indicates the minimum number of epochs needed to train a model to 45% robust accuracy.

Table 4.6: Time to train a robust CIFAR10 classifier to 45% robust accuracy using various adversarial training methods with the DAWNBench techniques of cyclic learning rates and mixed-precision arithmetic, showing significant speedups for all forms of adversarial training.

Method	Epochs	Seconds/epoch	Total time (minutes)
DAWNBench + PGD-7	10	104.94	17.49
DAWNBench + Free ($m = 8$)	80	13.08	17.44
DAWNBench + FGSM	15	25.36	6.34
PGD-7 [Madry et al., 2017][6]	205	1456.22	4965.71
Free ($m = 8$) [Shafahi et al., 2019][7]	205	197.77	674.39

4.2.2 Fast CIFAR10

We begin our CIFAR10 experiments by combining the DAWNBench improvements from Section 4.1.5 with various forms of adversarial training. For N epochs, we use a cyclic learning rate that increases linearly from 0 to λ over the first $N/2$ epochs, then decreases linearly from λ to 0 for the remaining epochs, where λ is the maximum learning rate. For each method, we individually tune λ to be as large as possible without causing the training loss to diverge, which is the recommended learning rate test from Smith and Topin [2018].

To identify the minimum number of epochs needed for each adversarial training method, we repeatedly run each method over a range of maximum epochs N, and then plot the final robustness of each trained model in Figure 4.7. For all methods, we use a batch size of 128, and SGD optimizer with momentum 0.9 and weight decay $5 * 10^{-4}$. We report the average results over 3 random seeds. Maximum learning rates used for the cyclic learning rate schedule are shown in Table 4.5. While all the adversarial training methods benefit greatly from the cyclic learning rate schedule, we find that both FGSM and PGD adversarial training require much fewer epochs than free adversarial training, and consequently reap the greatest speedups.

Table 4.7: Imagenet classifiers trained with adversarial training methods at $\epsilon = 2/255$ and $\epsilon = 4/255$.

Method	ϵ	Standard acc.	PGD+1 restart	PGD+10 restarts	Total time (hrs)
FGSM	$2/255$	60.90%	43.46%	43.43%	12.14
Free $(m = 4)$	$2/255$	64.37%	43.31%	43.28%	52.20
FGSM	$4/255$	55.45%	30.28%	30.18%	12.14
Free $(m = 4)$	$4/255$	60.42%	31.22%	31.08%	52.20

Using the minimum number of epochs needed for each training method to reach a baseline of 45% robust accuracy, we report the total training time in Table 4.6. We find that while all adversarial training methods benefit from the DAWNBench improvements, FGSM adversarial training is the fastest, capable of learning a robust CIFAR10 classifier in 6 minutes using only 15 epochs. Interestingly, we also find that PGD and free adversarial training take comparable amounts of time, largely because free adversarial training does not benefit from the cyclic learning rate as much as PGD or FGSM adversarial training.

4.2.3 Fast ImageNet

Finally, we apply all of the same techniques (FGSM adversarial training, mixed-precision, and cyclic learning rate) on the ImageNet benchmark. In addition, the top submissions from the DAWNBench competition for ImageNet utilize two more improvements on top of this, the first of which is the removal of weight decay regularization from batch normalization layers. The second addition is to progressively resize images during training, starting with larger batches of smaller images in the beginning and moving on to smaller batches of larger images later. Specifically, training is divided into three phases, where phases 1 and 2 use images resized to 160 and 352 pixels respectively, and phase 3 uses the entire image. We train models to be robust at $\epsilon = 2/255$ and $\epsilon = 4/255$ and compare to free adversarial training in Table 4.7, showing similar levels of robustness. In addition to using ten restarts, we also report the PGD accuracy with one restart to reproduce the evaluation done by Shafahi et al. [2019].

With these techniques, we can train an ImageNet classifier using 15 epochs in 12 hours using FGSM adversarial training, taking a fraction of the cost of free adversarial training as shown in Table 4.8.[8] We compare to the best performing variation of free adversarial training which which uses $m = 4$ minibatch replays over 92 epochs of training (scaled down accordingly to 23 passes over the data). Note that free adversarial training can also be enhanced with mixed-precision arithmetic, which reduces the runtime by 25%, but is still slower than FGSM-based training.

[6]Runtimes calculated on our hardware using the publicly available training code at https://github.com/MadryLab/cifar10_challenge.

[7]Runtimes calculated on our hardware using the publicly available training code at https://github.com/ashafahi/free_adv_train.

[8]We use the implementation of free adversarial training for ImageNet publicly available at https://github.com/mahyarnajibi/FreeAdversarialTraining and reran it on the our machines to account for any timing discrepancies due to differences in hardware

Table 4.8: Time to train a robust ImageNet classifier using various fast adversarial training methods

Method	Precision	Epochs	Min/epoch	Total time (hrs)
FGSM (phase 1)	single	6	22.65	2.27
FGSM (phase 2)	single	6	65.97	6.60
FGSM (phase 3)	single	3	114.45	5.72
FGSM	single	15	-	14.59
Free ($m = 4$)	single	92	34.04	52.20
FGSM (phase 1)	mixed	6	20.07	2.01
FGSM (phase 2)	mixed	6	53.39	5.34
FGSM (phase 3)	mixed	3	95.93	4.80
FGSM	mixed	15	-	12.14
Free ($m = 4$)	mixed	92	25.28	38.76

4.2.4 Combining free adversarial training with DAWNBench improvements on ImageNet

While adding mixed-precision is a direct speedup to free adversarial training without hurting performance, using other optimization tricks such as the cyclic learning rate schedule, progressive resizing, and batch-norm regularization may affect the final performance of free adversarial training. Since ImageNet is too large to run a comprehensive search over the various parameters as was done for CIFAR10 in Table 4.6, we instead test the performance of free adversarial training when used as a drop-in replacement for FGSM adversarial training with all the same optimizations used for FGSM adversarial training. We use free adversarial training with $m = 3$ minibatch-replay, with 2 epochs for phase one, 2 epochs for phase two, and 1 epoch for phase three to be equivalent to 15 epochs of standard training. PGD+N denotes the accuracy under a PGD adversary with N restarts.

A word of caution: this is not to claim that free adversarial training is completely incompatible with the DAWNBench optimizations on ImageNet. By giving free adversarial training more epochs, it may be possible recover the same or better performance. However, tuning the DAWNBench techniques to be optimal for free adversarial training is not the objective of this section, and so this is merely to show what happens if we naively apply the same DAWNBench tricks used for FGSM adversarial training to free adversarial training. Since free adversarial training requires more epochs even when tuned with DAWNBench improvements for CIFAR10, we suspect that the same behavior occurs here for ImageNet, and so 15 epochs is likely not enough to obtain top performance for free adversarial training. Since one epoch of FGSM adversarial training is equivalent to two epochs of free training, a fairer comparison is to give free adversarial training 30 epochs instead of 15. Even with double the epochs (and thus the same compute time as FGSM adversarial training), we find that it gets closer but doesn't quite recover the original performance of free adversarial training.

Table 4.9: ImageNet classifiers trained with free adversarial training methods at $m = 3$ minibatch replay when augmented with DAWNBench optimizations, against ℓ_∞ perturbations of radius $\epsilon = 4/255$, where 30 epochs of free training is equivalent to 15 epochs of FGSM training

Method	Step size	Epochs	Standard acc.	PGD+1	PGD+10
Free+DAWNBench	$4/255$	15	49.87%	22.78%	22.18%
Free+DAWNBench	$5/255$	15	50.48%	22.88%	22.25%
Free+DAWNBench	$4/255$	30	49.87%	28.17%	27.08%
Free+DAWNBench	$5/255$	30	50.48%	28.73%	27.81%
Free ($m = 4$)	$4/255$	92	60.42%	31.22%	31.08%
FGSM	$5/255$	15	55.45%	30.28%	30.18%

4.2.5 Takeaways from FGSM adversarial training

While it may be surprising that FGSM adversarial training can result in robustness to full PGD adversarial attacks, this work highlights some empirical hypotheses and takeaways which we describe below.

1. *Adversarial examples need to span the entire threat model.* One of the reasons why FGSM and R+FGSM as done by Tramèr et al. [2017] may have failed is due to the restricted nature of the generated examples: the restricted (or lack of) initialization results in perturbations which perturb each dimension by either 0 or $\pm\epsilon$, and so adversarial examples with feature perturbations in between are never seen.
2. *Defenders don't need strong adversaries during training.* This work suggests that rough approximations to the inner optimization problem are sufficient for adversarial training. This is in contrast to the usage of strong adversaries at evaluation time, where it is standard practice to use multiple restarts and a large number of PGD steps.

4.3 Adversarial training and robust overfitting

In order to learn networks that are robust to adversarial examples, a commonly used method is adversarial training, which solves the following robust optimization problem

$$\min_\theta \sum_i \max_{\delta \in \Delta} \ell(f_\theta(x_i + \delta), y_i), \tag{4.1}$$

where f_θ is a network with parameters θ, (x_i, y_i) is a training example, ℓ is the loss function, and Δ is the perturbation set. Typically the perturbation set Δ is chosen to be an ℓ_p-norm ball (e.g. ℓ_2 and ℓ_∞ perturbations, which we consider in this paper), such that $\Delta = \{\delta : ||\delta||_p \leq \epsilon\}$ for $\epsilon > 0$. Adversarial training approximately solves the inner optimization problem, also known as the robust loss, using some adversarial attack method, typically with projected gradient descent (PGD), and then updates the model parameters θ using gradient descent [Madry et al., 2017]. For example, an ℓ_∞ PGD adversary would start at some random initial perturbation $\delta^{(0)}$ and

Table 4.10: Robust performance showing the occurrence of robust overfitting across datasets and perturbation threat models. The "best" robust test error is the lowest test error observed during training. The final robust test error is averaged over the last five epochs. The difference between final and best robust test error indicates the degradation in robust performance during training.

			Robust test error (%)		
Dataset	Norm	Radius	Final	Best	Diff
SVHN	ℓ_∞	$8/255$	45.6 ± 0.40	39.0	6.6
	ℓ_2	$128/255$	26.4 ± 0.27	25.2	1.2
CIFAR10	ℓ_∞	$8/255$	51.4 ± 0.41	43.2	8.2
	ℓ_2	$128/255$	31.1 ± 0.46	28.4	2.7
CIFAR100	ℓ_∞	$8/255$	78.6 ± 0.39	71.9	6.7
	ℓ_2	$128/255$	62.5 ± 0.09	56.8	5.7
ImageNet	ℓ_∞	$4/255$	85.5 ± 8.87	62.7	22.8
	ℓ_2	$76/255$	94.8 ± 1.16	63.0	31.8

iteratively adjust the perturbation with the following ℓ_∞ gradient steps while projecting back onto the ℓ_∞ ball with radius ϵ:

$$\begin{aligned} \tilde{\delta} &= \delta^{(t)} + \alpha \cdot \text{sign} \nabla_x \ell(f(x), y)) \\ \delta^{(t+1)} &= \max(\min(\tilde{\delta}, \epsilon), -\epsilon) \end{aligned} \tag{4.2}$$

We denote error rates when attacked by a PGD adversary as the "robust error", and error rates on the clean, unperturbed data as "standard error".

4.3.1 Robust overfitting: a general phenomenon for adversarially robust deep learning

In the standard, non-robust deep learning setting, it is common practice to train for as long as possible to minimize the training loss, as modern convergence curves for deep learning generally observe that the testing loss continues to decrease with the training loss. On the contrary, for the setting of adversarially robust training we make the following discovery:

Unlike the standard setting of deep networks, overfitting for adversarially robust training can result in worse test set performance.

This phenomenon, which we refer to as "robust overfitting", results in convergence curves as shown earlier in Figure 4.1. Although training appears normal in the earlier stages, after the learning rate decays, the robust test error briefly decreases but begins to increase as training progresses. This behavior indicates that the optimal performance is not obtained at the end of training, unlike in standard training for deep networks.

We find that robust overfitting occurs across a variety of datasets, algorithmic approaches, and perturbation threat models, indicating that it is a general property of the adversarial training

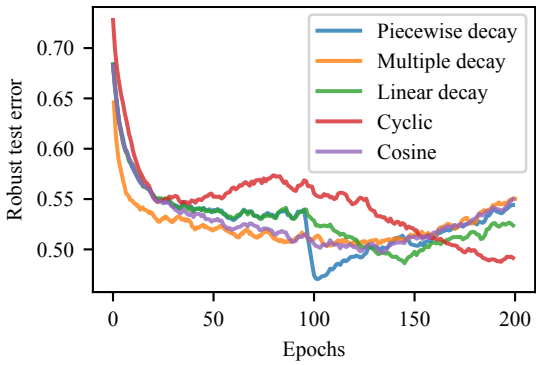


Figure 4.8: Robust test error over training epochs for various learning rate schedules on CIFAR10. None of the alternative smoother learning rate schedules can achieve a peak performance competitive with the standard piecewise decay learning rate, indicating that the peak performance is obtained by having a single discrete jump. Note that the multiple decay schedule is actually run for 500 epochs, but compressed into this plot for a clear comparison.

formulation and not specific to a particular problem, as can be seen in Table 4.10 for ℓ_∞ and ℓ_2 perturbations on SVHN, CIFAR10, CIFAR100, and ImageNet. A more detailed and expanded version of this table summarizing the full extent of robust overfitting as well as the corresponding learning curves for each setting can be found in Section 4.3.4. We consistently find that there is a significant gap between the best robust test performance during training and the final robust test performance at the end of training, observing an increase of 8.2% robust error for CIFAR10 and 22.8% robust error for ImageNet against an ℓ_∞ adversary, to highlight a few. Robust overfitting is also not specific to PGD-based adversarial training, and affects faster adversarial training methods such as FGSM adversarial training[9] [Wong et al., 2020a] as well as top performing algorithms for adversarially robust training such as TRADES [Zhang et al., 2019b].

4.3.2 Learning rate schedules and robust overfitting

Since the change in performance appears to be closely linked with the first drop in the scheduled learning rate decay, we explore how different learning rate schedules affect robust overfitting on CIFAR10, as shown in Figure 4.8 with a pre-activation ResNet18. Our search begins with a sweep over a range of different potential schedules which are commonly used in deep learning. Following this, we tune the best learning rate schedule to investigate its effect on the prevalence of robust overfitting.

We consider the following types of learning rates for our setting.

1. **Piecewise decay:** This is a fairly common learning rate used in deep learning, which decays the learning rate by a constant factor at fixed epochs. We begin with a learning rate of 0.1 and decay it by a factor of 10 at the 100th and 150th epochs, for 200 total epochs.
2. **Multiple decay:** This is a more gradual version of the piecewise decay schedule, with

[9] Wong et al. [2020a] also observe a different form of overfitting specifically for FGSM adversarial training which they refer to as "catastrophic overfitting". This is separate behavior from the robust overfitting described in this paper, and the specifics of this distinction are discussed further in Section 4.3.7.

a piecewise constant schedule which reduces the learning rate at a linear rate in order to make the drop in learning rate less drastic. Specifically, the learning rate begins at 0.1 and is reduced by 0.01 every 50 epochs over 500 total epochs, eventually reaching a learning rate of 0.01 in the last 50 epochs.

3. **Linear decay:** This schedule does a linear interpolation of the drop from 0.1 to 0.01, resulting in a piecewise linear schedule. The learning rate is trained at 0.1 for the first 100 epochs, then linearly reduced down to 0.01 over the next 50 epochs, and further trained at 0.01 for the last 50 epochs for a total of 200 epochs.
4. **Cyclic:** This schedule grows linearly from 0 to to some maximum learning rate λ, and then is reduced linearly back to 0 over training as proposed by Smith [2017]. We adopt the version from Wong et al. [2020a] which already computed the maximum learning rate for the CIFAR10 setting on the same architecture which peaks 2/5 of the way through training at a learning rate of 0.2 over 200 epochs.
5. **Cosine:** This schedule reduces the learning rate using the cosine function to interpolate from 0.1 to 0 over 200 epochs. This type of schedule was used by Carmon et al. [2019] when leveraging semi-supervised data augmentation to improve adversarial robustness.

Note that the piecewise decay schedule is the primary learning rate schedule used in this paper. All of these approaches beyond the standard piecewise decay schedule dampen the initial drop in robust test error experienced by the piecewise decay schedule. As a result, the best checkpoints of these alternatives end up with worse performance than the best checkpoint of the piecewise decay schedule, since all of the learning rates eventually start increasing in robust test error due to robust overfitting after the initial drop. Robust overfitting appears to be ubiquitous across different schedules, as most approaches achieve their best checkpoint well before training has converged.

The cyclic learning rate is the exception here, which has two phases corresponding to when the learning rate is growing and shrinking, with the best checkpoint occurring near the end of the second phase. In both phases, the robust performance begins to improve, but then robust overfitting eventually occurs and keeps the model from improving any further. We found that stretching the cyclic learning rate over a longer number of epochs (e.g. 300) results in a similar learning curve but with worse robust test error for both the best checkpoint and the final converged model.

In summary, we find that smoother learning rate schedules (which take smaller decay steps or interpolate the change in learning rate over epochs) simply result in smoother curves that still exhibit robust overfitting. Furthermore, with each smoother learning rate schedule, the best robust test performance during training is strictly worse than the best robust test performance during training with the discrete piecewise decay schedule.

4.3.3 Tuning the piecewise decay learning rates for robust overfitting

Since the piecewise decay schedule appeared to be the most effective method for finding a model with the best robust performance, we investigate whether this schedule can be potentially tuned to improve the robust performance of the best checkpoint even further. The discrete piecewise decay schedule has three possible parameters: the starting learning rate, the ending learning rate, and the epoch at which the decay takes effect. We omit the last 50 epochs of the final decay, since

Table 4.11: Tuning experiments using stochastic gradient descent to optimize the best robust test error obtained from the piecewise decay schedule for a pre-activation ResNet18 on CIFAR-10.

DECAY EPOCH	START LR	END LR	BEST ROB ERR
100	0.1	0.01	46.7%
60	0.1	0.01	47.4%
70			47.3%
80			46.9%
90			47.3%
100	0.06	0.01	47.4%
	0.08		46.7%
	0.3		48.7%
	0.5		51.0%
100	0.1	0.006	46.0%
		0.008	46.1%
		0.03	47.8%
		0.05	49.3%

the bulk of the impact from robust overfitting occurs shortly after the first decay in this setting.

While tuning the starting learning rate and the decay epoch largely results in either similar or worse performance, we find that adjusting the learning rate used after the decay epoch can actually slightly improve the robust performance of the best checkpoint by 0.5%, as seen in Table 4.11. Note that robust overfitting still occurs in these tuned learning rate schedules as seen in Figures 4.9, 4.10, and 4.11, which show the learning curves for each one of the models shown in Table 4.11.

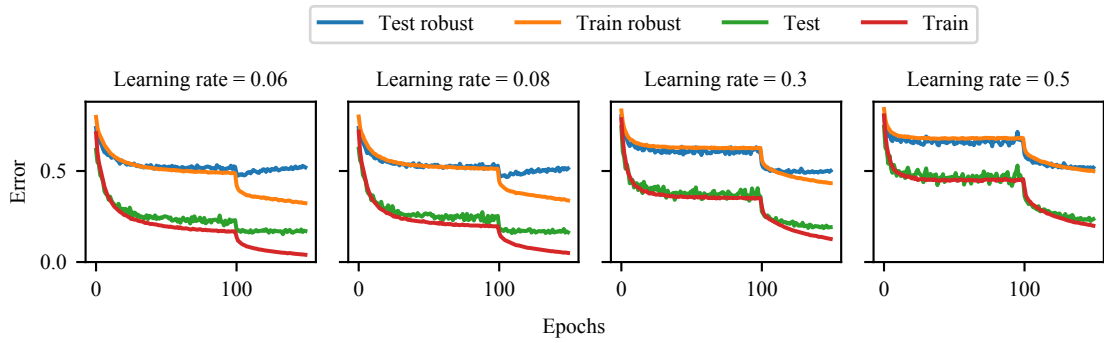


Figure 4.9: Learning curves for a piecewise decay schedule with a modified starting learning rate.

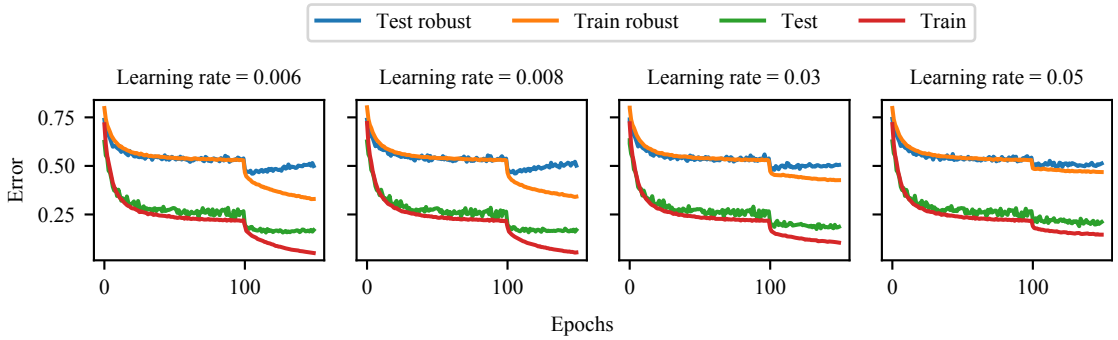


Figure 4.10: Learning curves for a piecewise decay schedule with a modified ending learning rate.

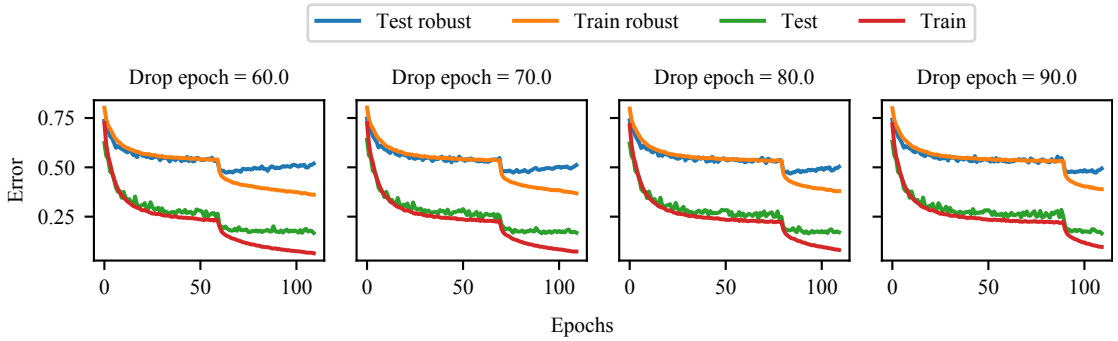


Figure 4.11: Learning curves for a piecewise decay schedule with a modified epoch at which the decay takes effect.

Table 4.12: Performance of adversarially robust training over a variety of datasets, adversarial training algorithms, and perturbation threat models, where the best error refers to the lowest robust test error achieved during training and the final error is an average of the robust test error over the last 5 epochs. We observe robust overfitting to occur across all experiments.

				Robust Test Error (%)			Standard Test Error (%)		
Dataset	Adversary	Norm	Radius	Final	Best	Diff	Final	Best	Diff
SVHN	PGD	ℓ_∞	$8/255$	45.6 ± 0.40	39.0	6.6	10.0 ± 0.15	10.2	-0.2
		ℓ_2	$128/255$	26.4 ± 0.27	25.2	1.2	7.0 ± 0.23	7.2	-0.2
CIFAR10	PGD	ℓ_∞	$8/255$	51.4 ± 0.41	43.2	8.2	13.4 ± 0.19	13.9	-0.5
		ℓ_2	$128/255$	31.1 ± 0.46	28.4	2.7	11.0 ± 0.08	11.3	-0.3
	FGSM	ℓ_∞	$8/255$	59.8 ± 0.09	53.7	6.1	12.4 ± 0.21	13.6	-1.2
		ℓ_2	$128/255$	31.6 ± 0.18	29.2	2.4	9.9 ± 0.16	10.5	-0.6
	TRADES	ℓ_∞	$8/255$	50.6 ± 0.31	45.0	5.6	14.97 ± 0.24	15.9	-0.9
		ℓ_2	$128/255$	58.2 ± 0.66	53.6	4.6	33.9 ± 0.95	15.7	18.2
CIFAR100	PGD	ℓ_∞	$8/255$	78.6 ± 0.39	71.9	6.7	45.9 ± 0.23	47.3	-1.4
		ℓ_2	$128/255$	62.5 ± 0.09	56.8	5.7	39.9 ± 0.22	37.5	2.4
ImageNet	PGD	ℓ_∞	$4/255$	85.5 ± 8.87	62.7	22.8	50.5 ± 14.32	37.0	13.5
		ℓ_2	$76/255$	94.8 ± 1.16	63.0	31.8	63.2 ± 6.80	40.1	23.1

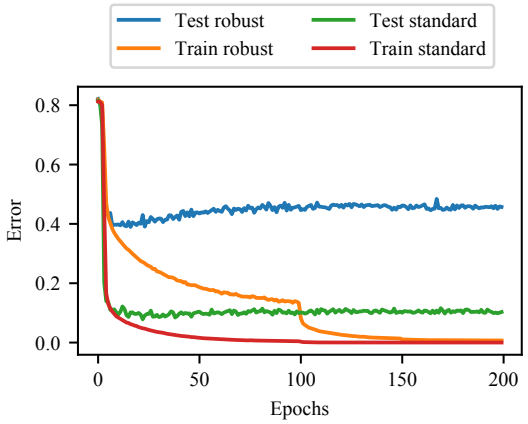


Figure 4.12: Learning curves for training an SVHN classifier which is adversarially robust to ℓ_∞ perturbations of radius 8/255. Note that robust overfitting occurs before the learning rate has decayed, likely due to the lower initial learning rate.

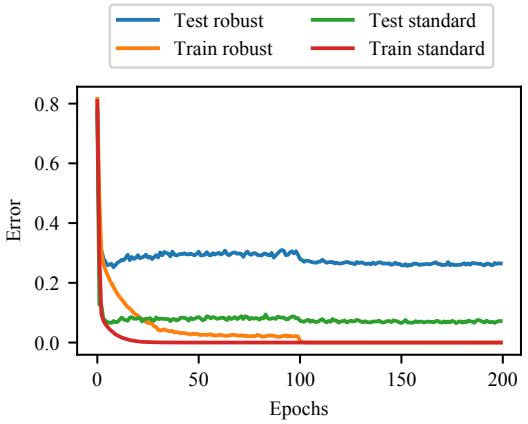


Figure 4.13: Learning curves for training an SVHN classifier which is adversarially robust to ℓ_2 perturbations of radius 128/255. Robust overfitting occurs early here as well, with robust test error increasing after the 9th epoch.

4.3.4 Detailed experimental results for robust overfitting

In this section, we extend Table 4.10 to additionally include standard error and results from different adversarial training schemes (FGSM and TRADES), as shown in Table 4.12. The final error is an average over the final 5 epochs of when the model has converged, along with the standard deviation. The best error is the lowest test error of all model checkpoints during training. For convenience we also show the difference in the final model's error and the best model's error, which indicates the amount of degradation incurred by robust overfitting.

The remainder of this section contains the experimental details for reproducing these experiments, as well as the learning curves for each experiment as visual evidence of robust overfitting. We default to using pre-activation ResNet18s for our experiments, with the exception of Wide ResNets with width factor 10 for ℓ_∞ adversaries on CIFAR10 (for a proper comparison to what is reported for TRADES), and ResNet50s for ImageNet. For CIFAR10 and CIFAR100, we train with the SGD optimizer using a batch size of 128, a step-wise learning rate decay set initially at 0.1 and divided by 10 at epochs 100 and 150, and weight decay $5 \cdot 10^{-4}$. For SVHN, we use the same parameters except with a starting learning rate of 0.01 instead. For ImageNet, we use the same learning configuration used to train the pretrained models and simply run them for longer epochs and lower learning rates using the publicly released repository available at `https://github.com/madrylab/robustness`.

ℓ_∞ adversary We consider the ℓ_∞ threat model with radius 8/255, with the PGD adversary taking 10 steps of size 2/255 on all datasets except for ImageNet. For ImageNet, we fine-tune the pretrained model from `https://github.com/madrylab/robustness` [Engstrom et al., 2019] and continue training with the exact same parameters with a learning rate of 0.001,

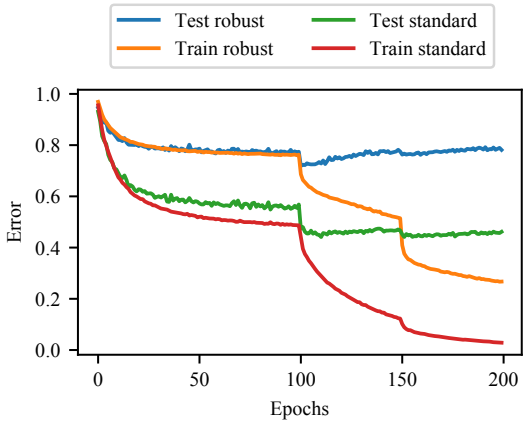


Figure 4.14: Learning curves showing robust overfitting on CIFAR100 for the ℓ_∞ perturbation model.

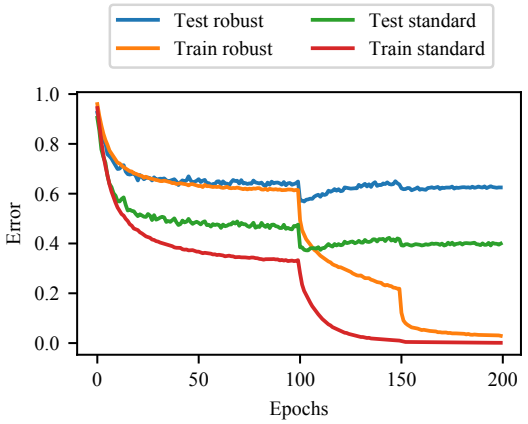


Figure 4.15: Learning curves showing robust overfitting on CIFAR100 for the ℓ_2 perturbation model.

which uses an adversary with 5 steps of size 0.9/255 within a ball of radius 4/255.

ℓ_2 adversary We consider the ℓ_2 threat model with radius 128/255, with the PGD adversary taking 10 steps of size 15/255 on all datasets except for ImageNet. For Imagenet, we fine-tune the pretrained model from `https://github.com/madrylab/robustness` [Engstrom et al., 2019] and continue training with the exact same parameters with a learning rate of 0.001, which uses an adversary with 7 steps of size 0.5 within a ball of radius 3.

4.3.5 Robust overfitting for SVHN and CIFAR100

Figures 4.12 and 4.13 contain the convergence plots for the PGD-based adversarial training experiments on SVHN for ℓ_∞ and ℓ_2 perturbations respectively. We find that robust overfitting occurs even earlier on this dataset, before the initial learning rate decay, indicating that the learning rate threshold at which robust overfitting begins to occur has already been passed. The best checkpoint for ℓ_∞ achieves 39.0% robust error, which is a 6.6% improvement over the 45.6% robust error achieved at the end of training.

Figures 4.14 and 4.15 contain the convergence plots for the PGD-based adversarial training experiments on CIFAR100 for ℓ_∞ and ℓ_2 perturbations respectively. We find that robust overfitting on this dataset reflects the CIFAR10 case, occurring after the initial learning rate decay. Note that in this case, both the robust test accuracy and the standard test accuracy are degraded from robust overfitting. The best checkpoint for ℓ_∞ achieves 71.9% robust error, which is a 6.7% improvement over the 78.6% robust error achieved at the end of training.

4.3.6 Robust overfitting in ImageNet

Figure 4.16 contains the convergence plots for our continuation of PGD-based adversarial training experiments on ImageNet for ℓ_∞ and ℓ_2 perturbations respectively. Thanks to logs provided

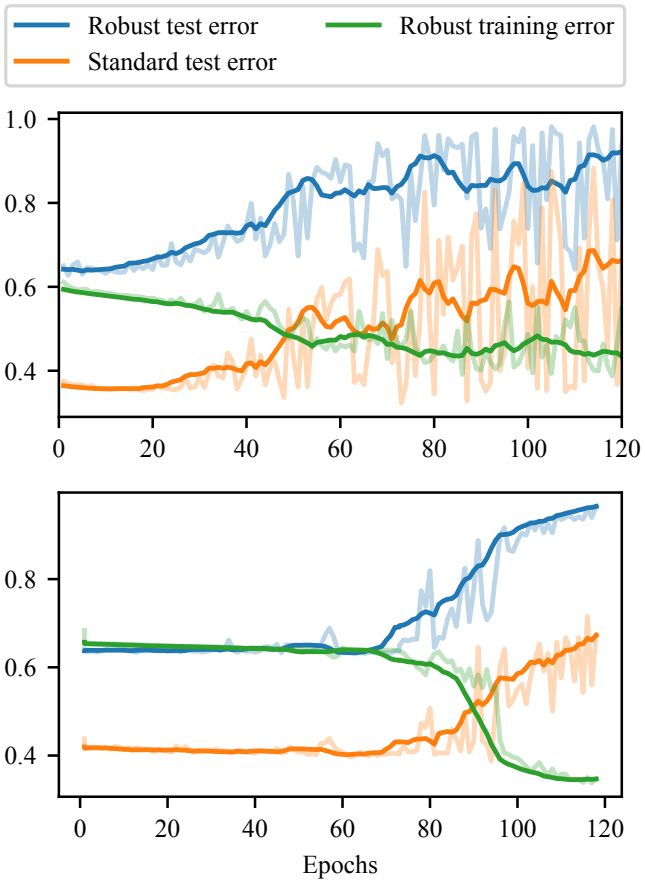


Figure 4.16: Continuation of training released pre-trained ImageNet models for ℓ_∞ (top) and ℓ_2 (bottom). The number of epochs indicate the number of additional epochs the pre-trained models were trained for.

by the authors [Engstrom et al., 2019], we know the pretrained ℓ_2 robust ImageNet model had already been trained for 100 epochs at learning rate 0.1 followed by at least 10 epochs at learning rate 0.01, and so we continue training from there and further decay the learning rate at the 150th epoch to 0.001. Logs could not be found for the pretrained ℓ_∞ model, and so it is unclear how long it was trained and under what schedule, however the pretrained model checkpoint indicated that the model had been trained for at least one epochs at learning rate 0.001, so we continue training from this point on.

The ℓ_∞ pre-trained model appeared to have not yet converged for the checkpointed learning rate, and so further training without any form of learning rate decay was able to gradually deteriorate the performance of the model. The ℓ_2 pre-trained model seemed to have already converged at the checkpointed learning rate, and so we do not see any significant changes in performance until after decaying the learning rate down to 0.001.

Note that the learning curves here are smoothed by taking an average over a consecutive 10 epoch window, as the actual curves are quite noisy in comparison to other datasets. This noise is reflected in Table 4.12, where ImageNet has the greatest variation in final error rates (both robust

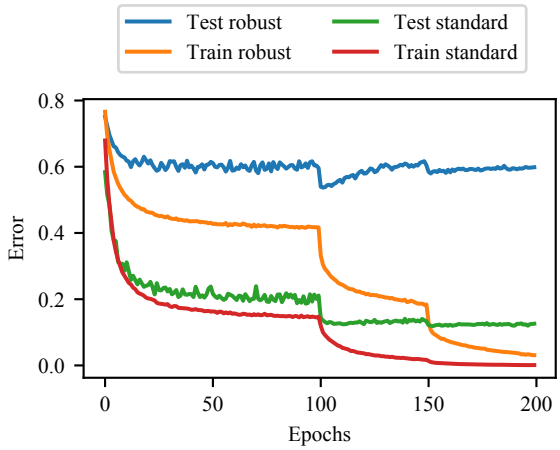


Figure 4.17: Learning curves showing robust overfitting from training with an FGSM adversary on CIFAR10 for the ℓ_∞ perturbation model.

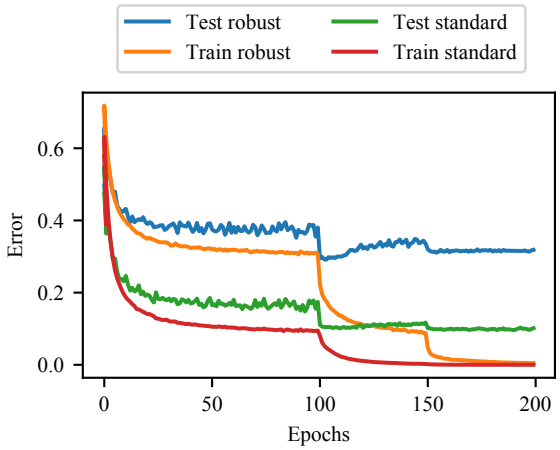


Figure 4.18: Learning curves showing robust overfitting from training with an FGSM adversary on CIFAR10 for the ℓ_2 perturbation model.

and standard). Training the models further can in fact improve the performance of the pretrained model slightly at specific checkpoints (e.g. from 66.4% initial robust test error down to 62.7% robust test error at the best checkpoint for ℓ_∞), however eventually the ImageNet models suffer greatly from robust overfitting, with an average increase of 22.8% robust error for the ℓ_∞ model and 31.8% robust error for the ℓ_2 model.

4.3.7 Robust overfitting for FGSM adversarial training

For CIFAR10, in addition to the standard PGD training algorithm, we also consider the FGSM adversarial training algorithm [Wong et al., 2020a] and TRADES [Zhang et al., 2019c]. The convergence curves showing that robust overfitting still occurs for these two algorithms in both the ℓ_∞ and ℓ_2 setting are shown in Figures 4.17 and 4.18 for FGSM and Figures 4.19 and 4.20 for TRADES.

FGSM adversarial training For FGSM adversarial training, we use the random initialization described by Wong et al. [2020a]. However, we find that when training until convergence using the piecewise decay learning rate schedule, the recommended step size of $\alpha = 10/255$ for ℓ_∞ training eventually results in catastrophic overfitting. We resort to reducing the step size of the ℓ_∞ FGSM adversary to 7/255 to avoid catastrophic overfitting, but still see robust overfitting.

We also note that Wong et al. [2020a] use a cyclic learning rate schedule to further boost the speed of convergence, which differs from the piecewise decay schedule we discuss in this paper. If we run FGSM adversarial training in a more similar fashion to Wong et al. [2020a] with the cyclic learning rate and fewer epochs, we find that this can sidestep the robust overfitting phenomenon and converge directly to the best checkpoint at the end of training. However, this requires a careful selection of the number of epochs: too few epochs and the final model

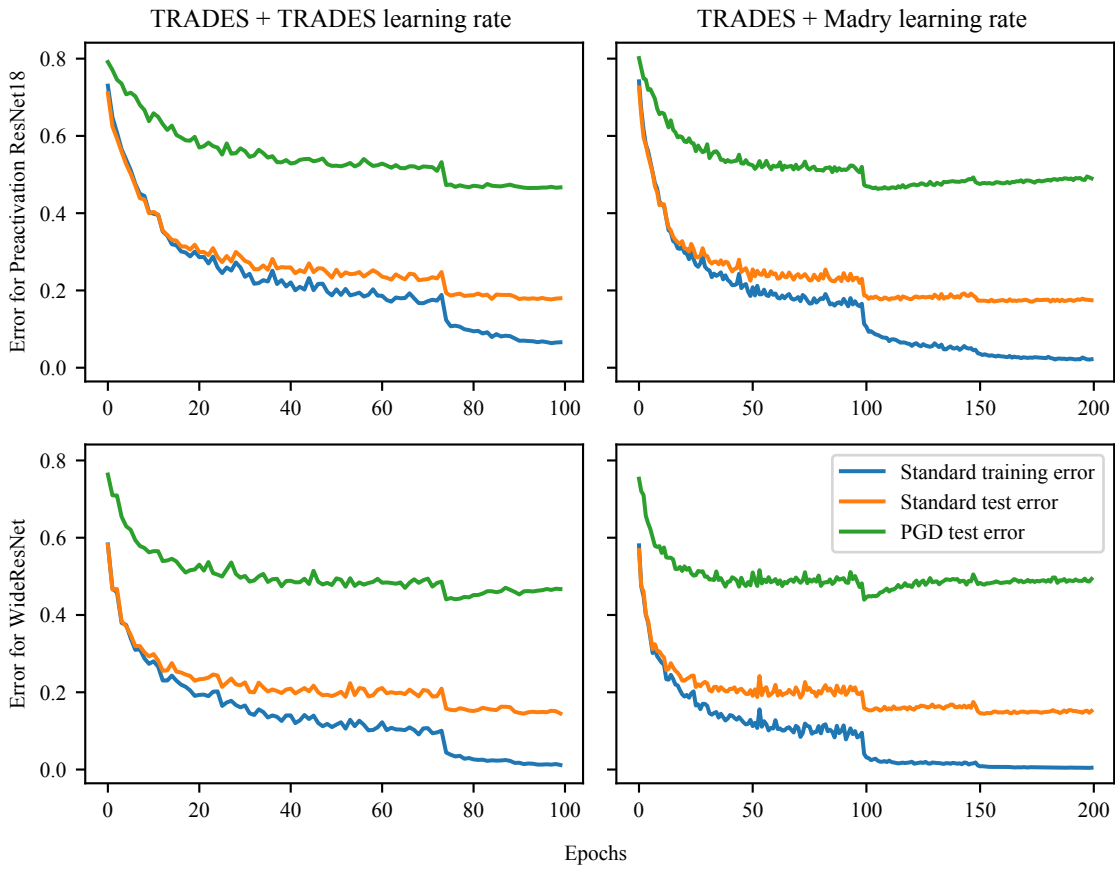


Figure 4.19: Learning curves when running TRADES for robustness to ℓ_∞ perturbations of radius 8/255 on combinations of learning rates and architectures for CIFAR10.

underperforms, whereas too many epochs and we observe robust overfitting. In our setting, we find that training against an FGSM adversary for 50 epochs using a cyclic learning rate with a maximum learning rate of 0.2 allows us to recover a final robust test error of 53.22%, similar to the best checkpoint of FGSM adversarial training with piecewise decay and 200 epochs which achieved 53.7% robust test error in Table 4.12.

Relation of robust overfitting to catastrophic overfitting Previous work studying the effectiveness of an FGSM adversary for robust training noted that it is necessary to prevent "catastrophic overfitting" in order for FGSM training to be successful, which can be avoided by evaluating a PGD adversary on a training minibatch [Wong et al., 2020a]. Here we note that this is a distinct and separate behavior from robust overfitting: while catastrophic overfitting is a product of a model overfitting to a weaker adversary and can be detected by a stronger adversary on the training set, robust overfitting is a degradation of robust test set performance under the *same* adversary used during training which *cannot be detected on the training set*. Indeed, even successful FGSM adversarial training can suffer from robust overfitting when given enough epochs without catastrophically overfitting, as shown in Figure 4.17, suggesting that this is related to the generalization properties of adversarially robust training rather than the strength of the adversary.

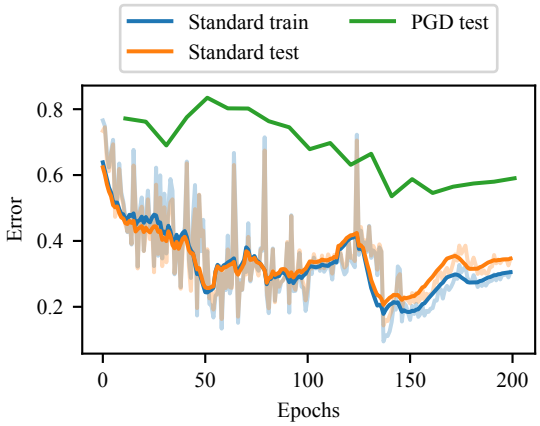


Figure 4.20: Learning curves when running TRADES for robustness to ℓ_2 perturbations of radius 128/255 for CIFAR10.

4.3.8 Robust overfitting for TRADES

For TRADES we use the publicly released implementation of both the defense and attack available at `https://github.com/yaodongyu/TRADES` to remove the potential for any confounding factors resulting from differences in implementation. We consider two possible options for learning rate schedules: the default schedule used by TRADES which decays at 75 and 90 epochs and runs for 100 epochs total (denoted TRADES learning rate),[10] and the standard learning rate schedule used by Madry et al. [2017] for PGD adversarial training, which decays at 100 epochs and 150 epochs. We additionally explore both the pre-activation ResNet18 architecture that we use extensively in this paper, as well as the Wide ResNet architecture which TRADES uses. The corresponding learning curves for each combination of learning rate and model can be found in Figure 4.19 for ℓ_∞.

We note that in three of the four cases, we see a clear instance of robust overfitting. Only the default learning rate schedule used by TRADES on the smaller, pre-activation ResNet18 model doesn't indicate any degradation in robust test set performance. This is likely due the shortened learning rate schedule which implicitly early stops combined with the regularization induced by a smaller architecture having less representational power. The results here are consistent with our earlier findings on the impact of architecture size, where the Wide ResNet architecture achieves better performance than the ResNet18. The shortened TRADES learning rate schedule does not show the full extent of robust overfitting, as the models have not yet converged, whereas the Madry learning rate does (and also achieves a slightly better best checkpoint).

Figure 4.20 shows a corresponding curve for ℓ_2 robustness using TRADES for the pre-activation ResNet18 model with the Madry learning rate, which was the optimal combination from ℓ_∞ training. Note that the TRADES repository does not provide default training param-

[10]This is the learning rate schedule described in the paper by Zhang et al. [2019b]. Note that this differs slightly from the implementation in the TRADES repository, which uses the same schedule but only trains for 76 epochs, which is one more epoch after decaying. In our reproduction of the TRADES experiment, the checkpoint after the initial learning rate decay ends up with the best test performance over all 100 epochs.

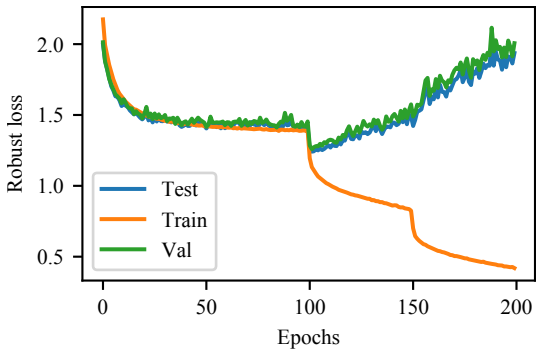


Figure 4.21: Learning curves for a CIFAR10 pre-activation ResNet18 model trained with a hold-out validation set of 1,000 examples. We find that the hold-out validation set is enough to reflect the test set performance, and stopping based on the validation set is able to prevent overfitting and recover 46.9% robust test error, in comparison to 46.7% achieved by the best-performing model checkpoint.

eters or a PGD adversary for ℓ_2 training on CIFAR-10 nor could we find any such description in the corresponding paper, and so we used our attack parameters which were successful for PGD-based adversarial training (10 steps of size 15/255).

4.3.9 Mitigating robust overfitting with early stopping

Proper early stopping, an old form of implicit regularization, calculates a metric on a hold-out validation set to determine when to stop training in order to prevent overfitting. Since the test performance does not monotonically improve during adversarially robust training due to robust overfitting, it is advantageous for robust networks to use early stopping to achieve the best possible robust performance.

We find that, for example, the TRADES approach relies heavily on using the best robust performance on the test set from an earlier checkpoint in order to achieve their top reported result of 43.4% robust error against an ℓ_∞ PGD adversary with radius $8/255$ on CIFAR10, a number which is typically viewed as a substantial algorithmic improvement in adversarial robustness over standard PGD-based adversarial training. In our own reproduction of the TRADES experiment, we confirm that allowing the TRADES algorithm to train until convergence results in significant degradation of robust performance as seen in Figure 4.19. Specifically, the robust test error of the model at the checkpoint with the best performance on the test set is 44.1% whereas the robust test error of the model at the end of training has increased to 50.6%.[11]

Surprisingly, when we early stop vanilla PGD-based adversarial training, selecting the model checkpoint with the best performance on the test set, we find that PGD-based adversarial training performs just as well as more recent algorithmic approaches such as TRADES. Specifically, when using the *same* architecture (a Wide ResNet with depth 28 and width factor 10) and the *same* 20-step PGD adversary for evaluation used by Zhang et al. [2019b] for TRADES, the model

[11]We used the public implementation of TRADES available at https://github.com/yaodongyu/TRADES and simply ran it to completion using the same learning rate decay schedule used by Madry et al. [2017].

checkpoint with the best performance on the test set from vanilla PGD-based adversarial training achieves 42.3% robust test error, which is actually slightly better than the best reported result for TRADES from Zhang et al. [2019b].[12]

Similarly, we find early stopping to be a factor in the robust test performance for publicly released pre-trained ImageNet models [Engstrom et al., 2019]. Continuing to train these models degrades the robust test performance from 62.7% to 85.5% robust test error for ℓ_∞ robustness at $\epsilon = 4/255$ and 63.0% to 94.8% robust test error for ℓ_2 robustness at $\epsilon = 128/255$, as seen in Section 4.3.6. This shows that these models are also susceptible to robust overfitting and benefit greatly from early stopping.[13]

Validation-based early stopping Early stopping based on the test set performance, however, leaks test set information and goes against the traditional machine learning paradigm. Instead, we find that it is still possible to recover the best test performance achieved during training with a true hold-out validation set. By holding out 1,000 examples from the CIFAR10 training set for validation purposes, we use validation-based early stopping to achieve 46.9% robust error on the test set *without looking at the test set*, in comparison to the 46.7% robust error achieved by the best-performing model checkpoint for a pre-activation ResNet18. The resulting validation curve during training closely matches the testing curve as seen in Figure 4.21, and suggests that although robust overfitting degrades the robust test set performance, selecting the best checkpoint in adversarially robust training for deep networks still does not appear to significantly overfit to the test set (which has been previously observed in the standard, non-robust setting [Recht et al., 2018]).

4.3.10 Reconciling double descent curves

Modern generalization curves for deep learning typically show improved test set performance for increased model complexity beyond data point interpolation in what is known as *double descent* [Belkin et al., 2019]. This suggests that overfitting by increasing model complexity using overparameterized neural networks is beneficial and improves test set performance. However, this appears to be at odds with the main findings of this paper; since training for longer can also be viewed as increasing model complexity, the fact that training for longer results in worst test set performance seems to contradict double descent.

We find that, while increasing either training time or architecture size can be viewed as increasing model complexity, these two approaches actually have separate effects; training for longer degrades the robust test set performance regardless of architecture size, while increasing the model architecture size still improves the robust test set performance despite robust overfitting. This was briefly noted by Nakkiran et al. [2019] for the ℓ_2 robust setting, and so in this section we show that this generally holds also in the ℓ_∞ robust setting.

For architecture size experiments, we use a Wide ResNet architecture [Zagoruyko and Komodakis, 2016] with depth 28 and varying widths to control the size of the network. For each

[12] We found that our implementation of the PGD adversary to be slightly more effective, increasing the robust test error of the TRADES model and the PGD trained model to 45.0% and 43.2% respectively.

[13] We use the publicly available framework from `https://github.com/madrylab/robustness` and continue training checkpoints obtained from the authors using the same learning parameters.

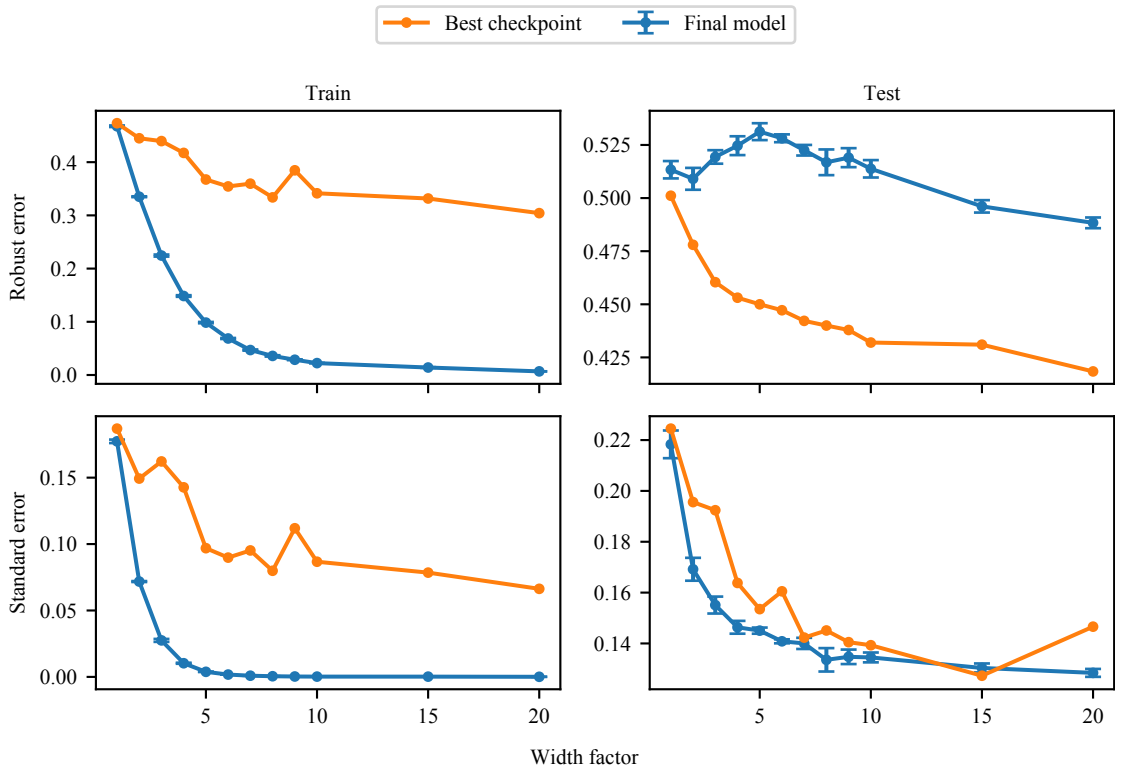


Figure 4.22: Standard and robust performance on the train and test set across Wide ResNets with varying width factors.

width tested, we plot the standard and robust performance from the best checkpoint and final model in Figure 4.22. Learning curves for each width can be found in Figure 4.23. All models were trained with the same training parameters described in Section 4.4. Mean and standard deviation of the final model was taken over the last 5 epochs.

From both the generalization curves and the individual convergence plots, we see that no matter how large the architecture is, the checkpoint which achieves the lowest robust test error always has higher training robust error than the final model at convergence. We also find that both the final model at the end of convergence as well as the best checkpoint found during training all benefit from the increase in architecture size. Consequently, we find that robust overfitting and double descent can occur at the same time, despite having seemingly opposite effects on the notion of overfitting.

In contrast to the standard setting, we observe that the double descent occurs well before robust interpolation of the training data at a width factor of 5, after which the robust test set performance of the final model continues to improve with even larger architecture sizes. The network with width factor 20, the largest that we could run on our hardware, achieves 48.8% robust test error at the end of training and 41.8% robust test error at the best checkpoint. This marks a further improvement over the more typical choice of width factor 10 which achieves 51.4% robust test error at the end of training and 43.2% robust test error at the best checkpoint.

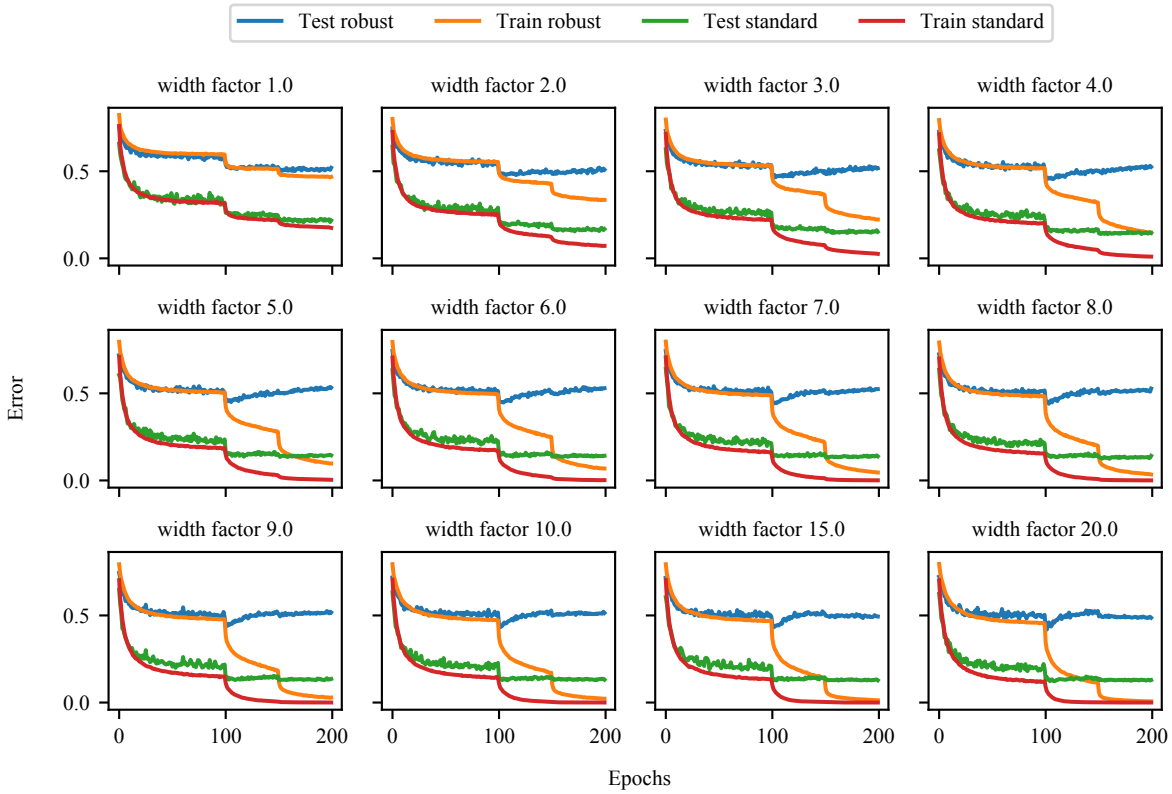


Figure 4.23: Learning curves for training Wide ResNets with different width factors.

4.4 Alternative methods to prevent robust overfitting

In this section, we explore whether common methods for combating overfitting in standard training are successful at mitigating robust overfitting in adversarial training. We run a series of ablation studies on CIFAR10 using classical and modern regularization techniques, yet ultimately find that no technique performs as well in isolation as early stopping, as shown in Table 4.13. Unless otherwise stated, we begin each experiment with the standard setup for ℓ_∞ PGD-based adversarial training with a 10-step adversary with step size $2/255$ using a pre-activation ResNet18 [He et al., 2016b]. All experiments in this section were run with one GeForce RTX 2080ti unless a Wide ResNet was trained, in which case two GPUs were used.

The final robust and standard errors are an average of over the final 5 epochs of training when the model has converged, from which the standard deviation is also computed. The one exception is validation-based early stopping, where the final error is taken from the checkpoint chosen by the validation set, and consequently does not have a standard deviation. The best robust error is the lowest test robust error of all checkpoints through training, and the best standard error is the corresponding standard error which comes from this same checkpoint. For convenience we also show the difference in the final model's error and the best model's error, which indicates the amount of degradation incurred by robust overfitting.

Table 4.13: Performance of adversarially robust training over a variety of regularization techniques for PGD-based adversarial training on CIFAR-10 for ℓ_∞ with radius $8/255$.

	Robust Test Error (%)			Standard Test Error (%)		
Regularization method	Final	Best	Diff	Final	Best	Diff
Early stopping w/ val	**46.9**	46.7	**0.2**	18.2	18.2	0.0
ℓ_1 regularization	53.0 ± 0.39	48.6	4.4	15.9 ± 0.13	15.4	0.5
ℓ_2 regularizaiton	51.4 ± 0.73	46.4	8.8	15.7 ± 0.21	14.9	0.8
Cutout	48.8 ± 0.79	46.7	2.1	16.8 ± 0.21	16.4	0.4
Mixup	49.1 ± 1.32	46.3	2.8	23.3 ± 3.04	19.0	4.3
Semi-supervised	47.1	40.2	6.9	23.0 ± 3.82	17.2	5.8

Training procedure and adversary parameters For the experiments in preventing overfitting, we use a PGD adversary with random initialization and 10 steps of step size 2/255. This is a slightly stronger adversary than considered in Madry et al. [2017] by using 3 additional steps, and we found the attack to be more effective than the adversary implemented by TRADES, achieving approximately 1% more PGD error than the TRADES adversary. However, our goal here is to explore the prevention of robust overfitting, and so it is not necessary to have strongest possible adversarial attack, and so for our purposes this adversary is good enough (and is known to be reasonably strong in the ℓ_∞ setting). For training, we use the same parameters as used for the CIFAR-10 experiments in Section 4.3.4 (batch size, learning rate, weight decay, number of epochs). We primarily use the pre-activation ResNet18 since it is already sufficient for exhibiting the robust overfitting behavior.

4.4.1 Explicit ℓ_1 and ℓ_2 regularization

A classical method for preventing overfitting is to add an explicit regularization term to the loss, penalizing the complexity of the model parameters. Specifically, the term is typically of the form $\lambda\Omega(\theta)$, where θ contains the model parameters, $\Omega(\theta)$ is some regularization penalty, and λ is a hyperparameter to control the regularization effect. A typical choice for Ω is ℓ_p regularization for $p \in \{1, 2\}$, where ℓ_2 regularization is canonically known as weight decay and commonly used in standard training of deep networks, and ℓ_1 regularization is known to induce sparsity properties.[14]

ℓ_1 regularization Figure 4.24 shows the training and testing performance of models using various degrees of ℓ_1 regularization. We performed a search over regularization parameters $\lambda = \{5 \cdot 10^{-6}, 5 \cdot 10^{-5}, 5 \cdot 10^{-4}, 5 \cdot 10^{-3}\}$, and found that both the final checkpoint and the best checkpoint have an optimal regularization parameter of $5 \cdot 10^{-5}$. Note that we only see robust overfitting at smaller amounts of regularization, since the larger amounts of regularization actually regularize the model to the point where the performance is being severely hurt. Figure

[14]Proper parameter regularization only applies the penalty to the weights w of the affine transformations at each layer, excluding the bias terms and batch normalization parameters.

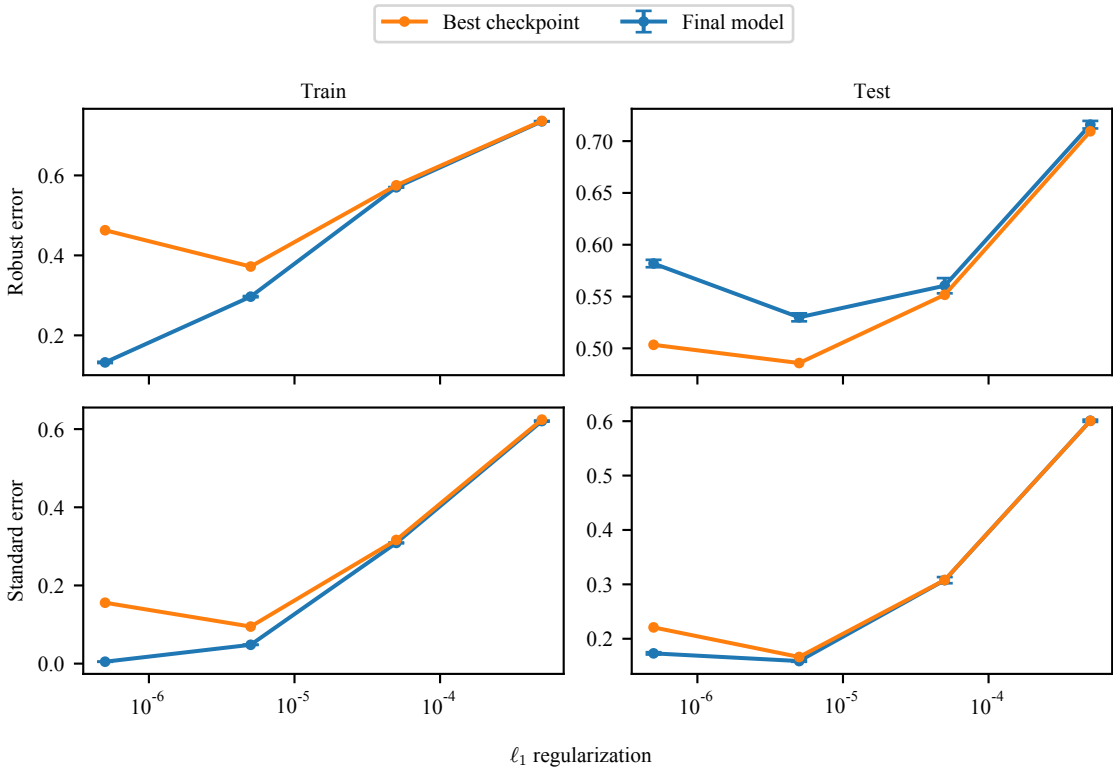


Figure 4.24: Standard and robust performance on the train and test set using varying degrees of ℓ_1 regularization.

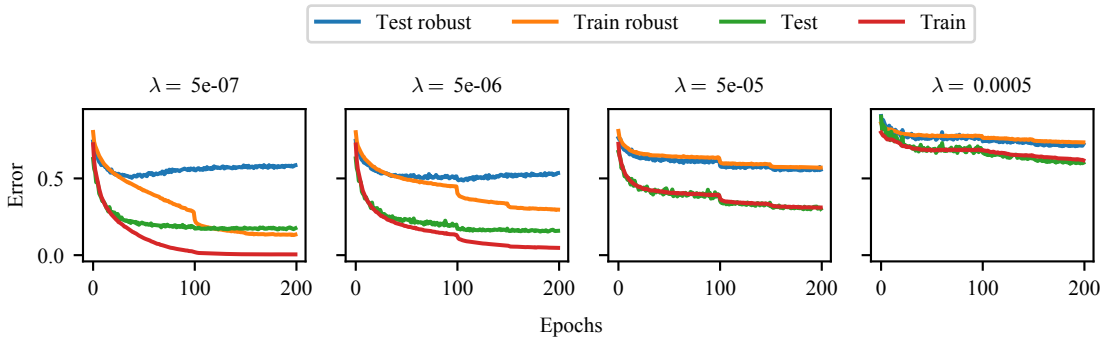


Figure 4.25: Learning curves for adversarial training using ℓ_1 regularization.

4.25 shows the corresponding learning curves for these four models. We see clear robust overfitting for the smaller two options in λ, and find no overfitting but highly regularized models for the larger two options, to the extent that there is no generalization gap and the training and testing curves actually appear to match.

ℓ_2 regularization Figure 4.26 shows the training and testing performance of models using various degrees of ℓ_2 regularization. We performed a search over regularization parameters $\lambda = \{5 \cdot 10^k\}$ for $k \in \{-4, -3, -2, -1, 0\}$ as well as $\lambda = 0.01$. Note that $5 \cdot 10^{-4}$ is a fairly widely

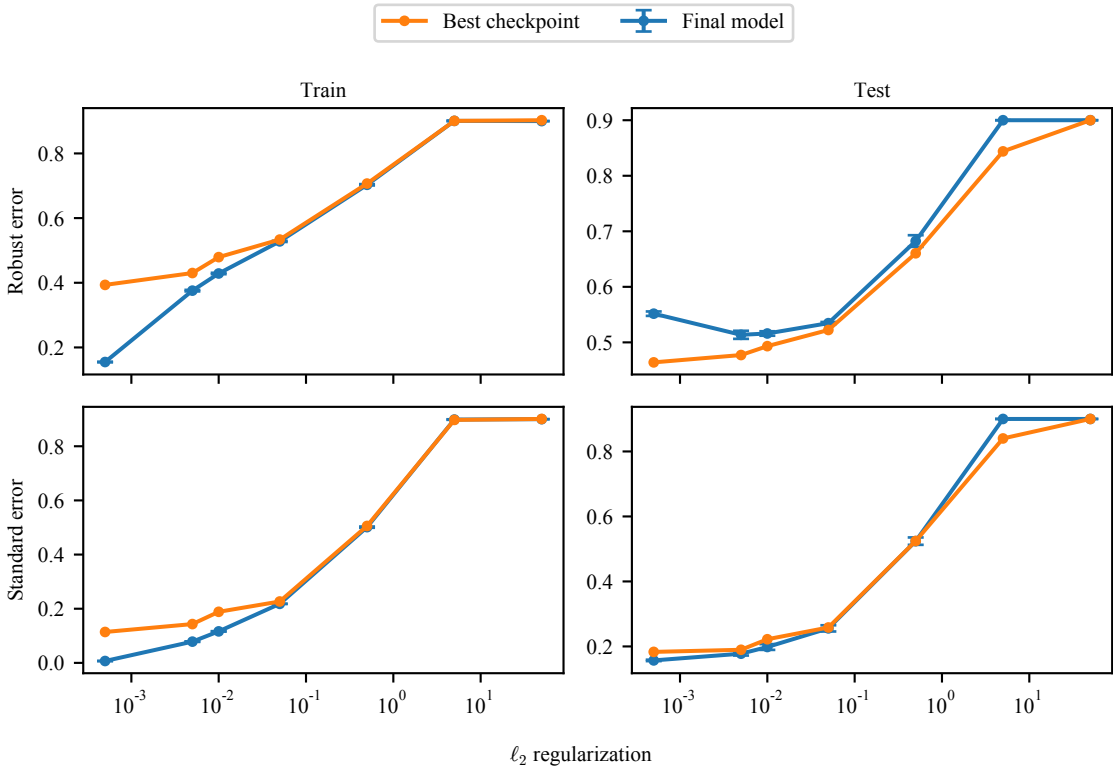


Figure 4.26: Standard and robust performance on the train and test set using varying degrees of ℓ_2 regularization.

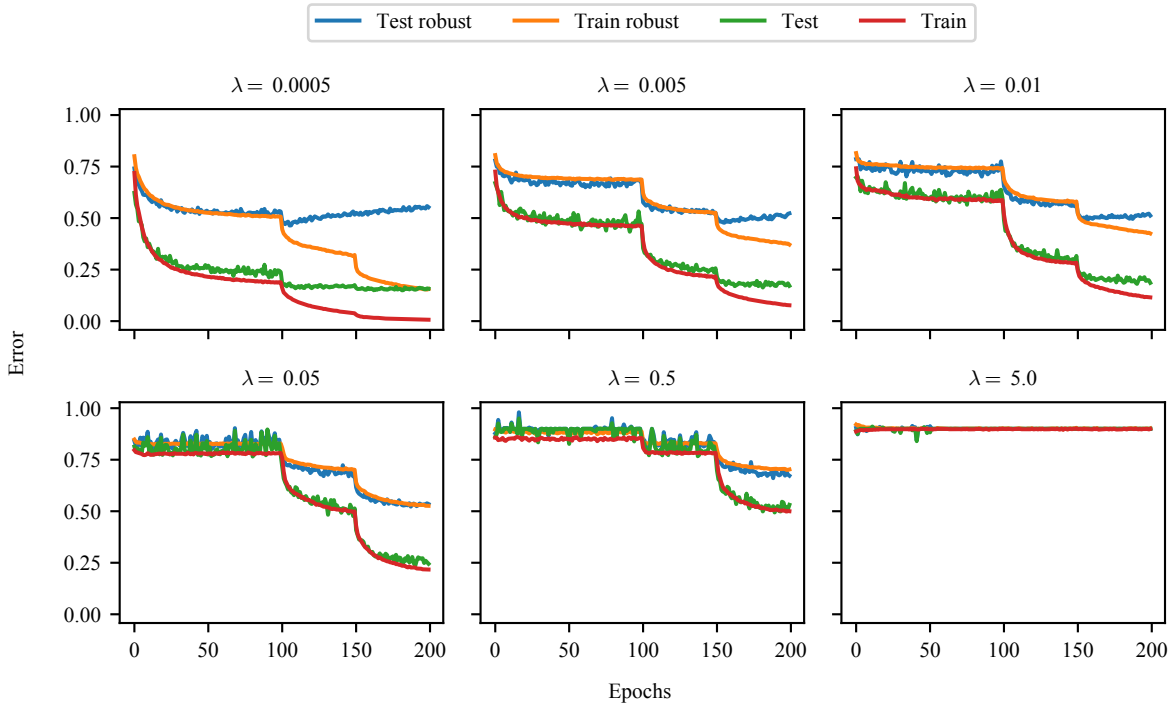


Figure 4.27: Learning curves for adversarial training using ℓ_2 regularization.

used value for weight decay in deep learning. We find that only the smallest choices for λ result in robust overfitting (e.g. $\lambda \leq 0.1$, with the best ℓ_2 regularizer achieving 55.2% robust test error with parameter $\lambda = 5 \cdot 10^{-2}$. However, inspecting the corresponding learning curves in Figure 4.27 reveals that the larger choices for λ have a similar behavior to the larger forms of ℓ_1 regularization, and end up with highly regularized models whose test performance perfectly matches the training performance at the cost of converging to a worse final robust test error.

In general, although explicit regularization does improve the performance to some degree, on its own, it is still not as effective as early stopping. Neither of these regularization techniques can completely remove the detrimental effects of robust overfitting without drastically over-regularizing the model

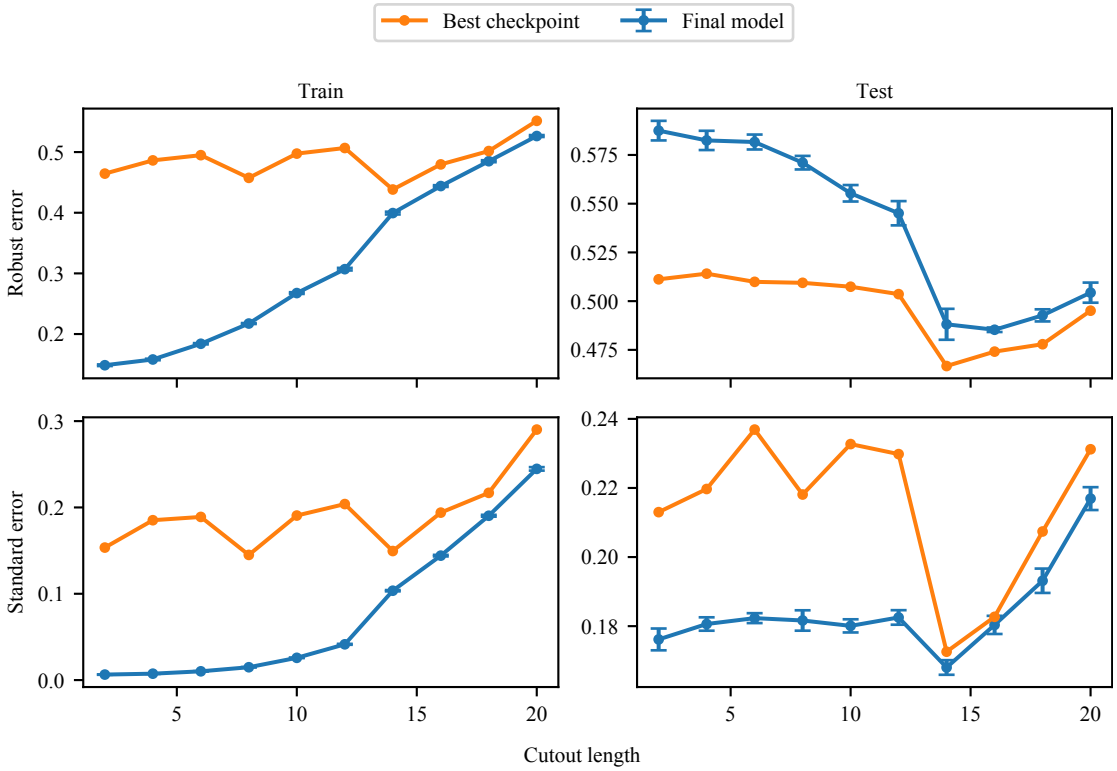


Figure 4.28: Standard and robust performance on the train and test set for varying cutout patch lengths.

4.4.2 Data augmentation for deep learning with Cutout and Mixup

Data augmentation has been empirically shown to reduce overfitting in modern deep learning tasks that involve very high-dimensional data by enhancing the quantity and diversity of the training data. Such techniques range from simple augmentations like random cropping and horizontal flipping, with more recent techniques such as cutout [DeVries and Taylor, 2017] and mixup [Zhang et al., 2017], all of which are known to reduce overfitting and improve generalization in the standard training setting. We scan a range of hyperparameters for these approaches

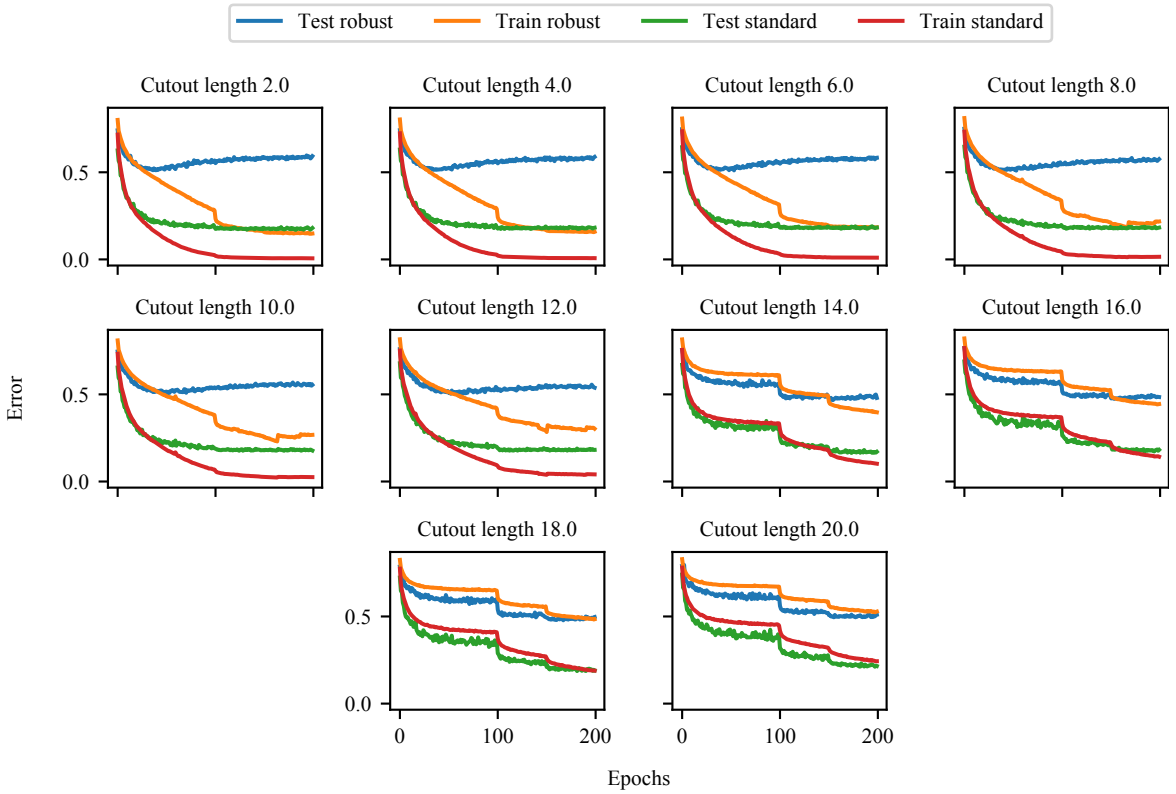


Figure 4.29: Learning curves for adversarial training using cutout data augmentation with different cutout patch lengths.

when applicable, and find a similar story to that of explicit ℓ_p regularization; either the regularization effect of cutout and mixup is too low to prevent robust overfitting, or too high and the model is over-regularized. When trained to convergence, neither cutout nor mixup is as effective as early stopping, achieving at best 48.8% robust test error for cutout with a patch length of 14 and 49.1% robust test error for mixup with $\alpha = 1.4$.[15]

Cutout To analyze the effect of cutout on generalization, we range the cutout hyperparameter of patch length from 2 to 20. Figure 4.28 shows the training and testing performance of models using varying choices of patch lengths. Additionally, for each hyperparameter choice, we plot the resulting learning curves in Figure 4.29.

We find the optimal length of cutout patches to be 14, which on it's own is not quite as good as vanilla early stopping, but when combined with early stopping merely matches the performance of vanilla early stopping. In all cases, we observe robust overfitting to steadily degrade the robust test performance throughout training, with less of an effect as we increase the cutout patch length.

[15]We used the public implementations of cutout and mixup available at https://github.com/davidcpage/cifar10-fast and https://github.com/facebookresearch/mixup-cifar10

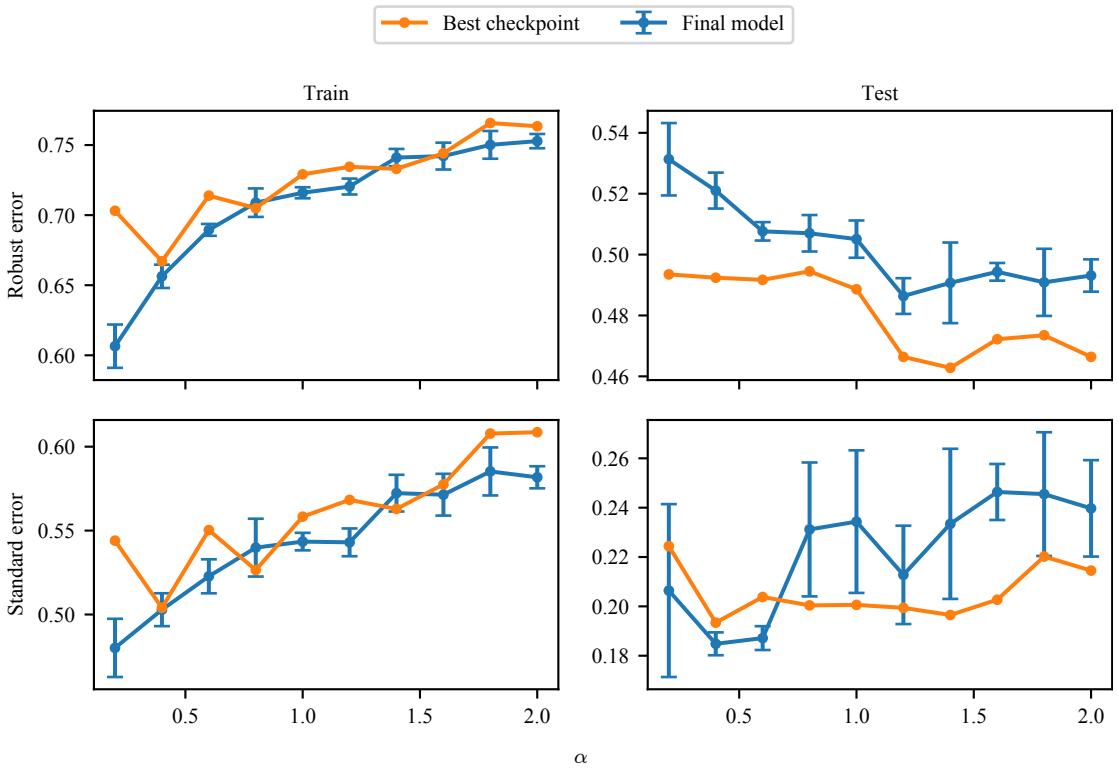


Figure 4.30: Standard and robust performance on the train and test set for varying degrees of mixup.

Mixup When training using mixup, we vary the hyperparameter α from 0.2 to 2.0. The training and testing performance of models using varying degrees of mixup can be found in Figure 4.30. The resulting learning curves for each choice of α can be found in Figure 4.31.

For mixup, we find an optimal parameter value of $\alpha = 1.4$. Similar to cutout, when combined with early stopping, it can only attain similar performance to vanilla early stopping, and otherwise converges to a worse model. However, although the learning curves for mixup training are significantly noisier than other methods, we do observe the robust test error to steadily decrease over training, indicating that mixup does stop robust overfitting to some degree (but does not obtain significantly better performance).

4.4.3 Robust overfitting and semi-supervised learning

Some work has argued that robust deep learning requires more data then standard deep learning [Schmidt et al., 2018], and recent work has leveraged unlabeled data with semi-supervised learning techniques to make substantial improvements on adversarial robustness benchmarks. We consider a self-supervised data augmentation technique [Alayrac et al., 2019, Carmon et al., 2019, Zhai et al., 2019] which uses a standard classifier to label unlabeled data for use in robust training.

For semi-supervised training, we use a batch size of 128 with equal parts labeled CIFAR-10

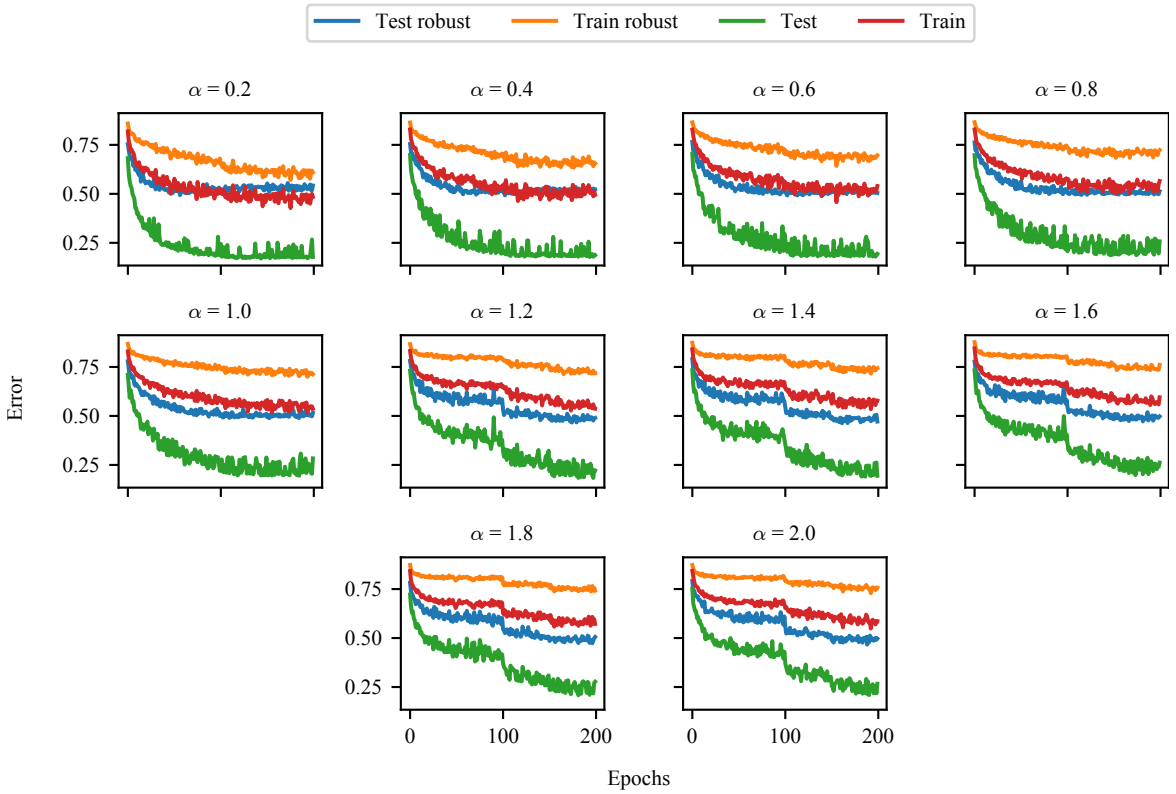


Figure 4.31: Learning curves for adversarial training using mixup with different choices of hyperparameter α.

data and pseudo-labeled TinyImages data, as recommended by Carmon et al. [2019]. Each epoch of training is now equivalent in computation to two epochs of standard adversarial training. Note that the pre-activation ResNet18 is a smaller architecture than used by Carmon et al. [2019], and so in our reproduction, the best checkpoint which achieves 40.2% error is about 2% higher than 38.5%, which is what Carmon et al. [2019] can achieve with a Wide ResNet. Note that in the typical adversarially robust setting without additional semi-supervised data, a Wide ResNet can achieve about 3.5% lower error than a pre-activation ResNet18.

Although there is a large gap between best and final robust performance shown in Table 4.13, we find that this is primarily driven by high variance in the robust test error during training rather than from robust overfitting, even when the model has converged as seen in Figure 4.32. In fact, we observe that the semi-supervised approach does not exhibit severe robust overfitting, as the smoothed learning curves don't show significant increases in robust test error.

However, relative to the base setting of using only the original dataset, the robust test performance is extremely variable, with a range spanning almost 10% robust error even when training error is relatively flat and has converged. Due to this variance, the final model's average robust performance of 47.1% robust test error is similar to the performance obtained by early stopping. By combining early stopping with semi-supervised data augmentation, this variance can be avoided. In fact, we find that the combination of early stopping and semi-supervised data augmentation is the only method that results in significant improvement over early stopping alone,

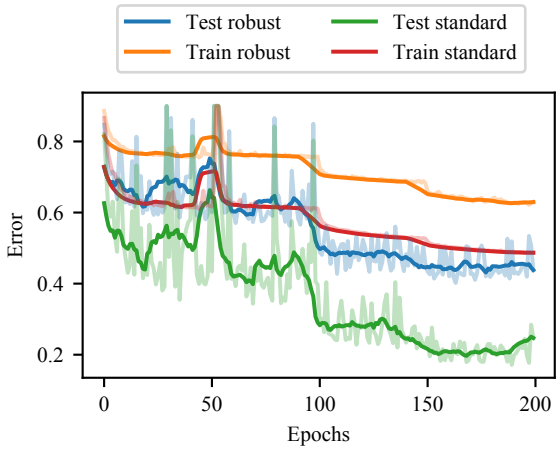


Figure 4.32: Learning curves for robust training with semi-supervised data augmentation, where we do not see a severe case of robust overfitting. When robust training error has converged, there is a significant amount of variance in the robust test error, so the average final model performance is on par with pure early stopping. Combining early stopping with semi-supervised data augmentation to avoid this variance is the only method we find that significantly improves on pure early stopping, reaching 40.2% robust test error.

resulting in 40.2% robust test error.[16]

4.5 Discussion

In this chapter, we presented two unexpected properties of robust learning. First, that FGSM adversarial training, when used with random initialization, can in fact be just as effective as the more costly PGD adversarial training. While a single iteration of FGSM adversarial training is double the cost of free adversarial training, it converges significantly faster, especially with a cyclic learning rate schedule. As a result, we are able to learn adversarially robust classifiers for CIFAR10 in minutes and for ImageNet in hours, even faster than free adversarial training but with comparable levels of robustness. By demonstrating that extremely weak adversarial training is capable of learning robust models, this work also exposes a new potential direction in more rigorously explaining when approximate solutions to the inner optimization problem are sufficient for robust optimization, and when they fail.

Second, unlike in standard training, overfitting in robust adversarial training decays test set performance during training in a wide variety of settings. While overfitting with larger architecture sizes results in better test set generalization, it does not reduce the effect of robust overfitting. Our extensive suite of experiments testing the effect of implicit and explicit regularization methods on preventing overfitting found that most of these techniques tend to over-regularize the model or do not prevent robust overfitting, and all of them in isolation do not improve upon early

[16]We used the data from https://github.com/yaircarmon/semisup-adv containing 500K pseudo-labeled TinyImages

stopping.

These findings have both practical and theoretical takeaways for the robust learning problem. By leveraging cheaper adversaries and early stopping, robust training time can be significantly reduced, accelerating research in learning models which are resistant to adversarial attacks and expanding the applicability of robust learning to larger domains. Indeed, this chapter re-establishes the competitiveness of the simplest adversarial training baseline, performing as well as state-of-the-art while being computationally fast. However, due to the prevalence of robust overfitting in adversarial training, validation sets should be used when performing model selection with early stopping. This overfitting behavior is a stark difference in generalization properties between standard and robust training, which is not fully explained by either classic statistics or modern deep learning.

Chapter 5

Threat models for adversarial robustness

While both provable defense and adversarial training can provide guaranteed and empirical robustness, both approaches depend crucially on having a well defined threat model. Afterall, a mathematically defined threat model is necessary in order to calculate bounds on the output of a network for provable defenses, and is also necessary to define the projection set of a PGD adversary in adversarial training. This also highlights the importance of having a well-defined threat model when researching adversarial examples: a well-defined threat model allows for meaningful measurements of robustness (e.g. by varying the strength of the adversary) while allowing for adversarial defenses to measure progress, both of which have been taken for granted in the ℓ_p setting that has been studied so far.

However, defining a meaningful threat model is not so trivial a task. The threat model needs to be large enough to contain meaningful variations in the input, and yet not be too large so as to contain irrelevant perturbations. This may be why formally defined alternative threat models for adversarial examples beyond the ℓ_p norm has been limited to fairly simple threat models such as image rotations and translations [Engstrom et al., 2017]. For example, although Xiao et al. [2018] study adversarial distortions of a images, the threat model does not have a natural parameter to control the complexity of the distortion, making it difficult to measure robustness at various thresholds or constrain the power of the adversary to reasonable levels. Similarly, adversarial examples in natural language need to first determine what a natural language perturbation is, as small ℓ_p norms do not apply as obviously in the discrete setting. Although provable defenses can leverage well-defined threat models from word substitutions [Jia et al., 2019], these are relatively simple threat models for language and it is unclear how to define a proper threat model which captures complex similarities in semantics and syntax [Alzantot et al., 2018].

Adversarial examples in the real world are often the least well-defined and furthest away from a small ℓ_p ball. Often they have to invent their own threat models and qualitatively evaluate them in a non-rigorous way, as the ℓ_p setting often doesn't apply under real conditions. For example, there is no obvious way to mathematically characterize the threat model induced by adversarial graffiti on traffic signs [Eykholt et al., 2018], or to define a restricted set of adversarial textures on 3D objects [Athalye et al., 2018b]. Sharif et al. [2016] generated their adversarial eyeglasses with restricted ℓ_p norm, and further restricted their adversarial glasses to those that looked "realistic", where Amazon Turk workers chose what glasses looked realistic. It is near impossible to even think about building a reasonable provable defense or adversarial training procedure on top of

a type of threat model defined by Amazon Turk workers. To extend the field of adversarial examples beyond the well-studied ℓ_p norm-bounded ball and make adversarial robustness more applicable and meaningful, in this chapter we propose two types of alternative adversarial threat models. One threat model differs fundamentally from the ℓ_p ball by incorporating structure into the perturbation, while the other defines a new threat model based on the union of existing threat models to generalize beyond a single perturbation.

First, we propose an attack model where the perturbed examples are bounded in Wasserstein distance from the original example. This distance can be intuitively understood for images as the cost of moving around pixel mass to move from one image to another, and has seen applications throughout machine learning including image classification [Snow et al., 2016] as well as image synthesis and restoration problems [Tartavel et al., 2016]. The traditional notion of Wasserstein distance has the drawback of being computationally expensive: computing a single distance involves solving an optimal transport problem (a linear program) with a number of variables quadratic in the dimension of the inputs. However, it was shown that by subtracting an entropy regularization term, one can compute approximate Wasserstein distances extremely quickly using the Sinkhorn iteration [Cuturi, 2013], which was later proven to converge in near-linear time [Altschuler et al., 2017].

Note that the Wasserstein ball and the ℓ_p ball can be quite different in their allowable perturbations: examples that are close in Wasserstein distance can be quite far in ℓ_p distance, and vice versa (a pedagogical example demonstrating this is in Figure 5.1). Crucially, this distance captures more "natural", semantically meaningful image perturbations such as translations, rotations, and distortions, and defining this threat model can be seen as a step towards building better adversarial defenses.

In order to generate Wasserstein adversarial examples, we use a PGD adversary which involves computing a projection operator onto the Wasserstein ball. To solve this projection problem, we leverage a similar entropy regularization term as done by the original Sinkhorn iteration [Cuturi, 2013] and derive a dual block coordinate ascent algorithm, which converges quickly and efficiently. To make the approach efficient and practical for high dimensional images, we leverage convolutional-style transport plans, resulting in a rapid Wasserstein adversarial attack which produces semantically meaningful adversarial perturbations that move mass around the boundaries of the objects in the image.

Relevant but orthogonal to this work, is that of Sinha et al. [2018b] on achieving distributional robustness using the Wasserstein distance. While we both use the Wasserstein distance in the context of learning robust networks, the setting and approach is quite different: Sinha et al. [2018b] use the Wasserstein distance to perturb the underlying *data distribution*, whereas we use the Wasserstein distance as an attack model for perturbing each *example*.

In the second half of this chapter, we study the problem of learning an classifier which is robust against the union of multiple threat models. This can be seen as the next natural step towards learning human-level classifiers, which are robust to many different types of perturbations. To solve this problem, Schott et al. [2019] used multiple variational autoencoders to construct a complex architecture for the MNIST dataset that is not as easily attacked by ℓ_∞, ℓ_2, and ℓ_0 adversaries. However the comparison to the PGD adversary baseline is not quite fair, as they only compare to a model trained against an ℓ_∞-bounded PGD adversary as described by Madry et al. [2017]. While not studied as a defense, Kang et al. [2019] study the transferability of adversarial

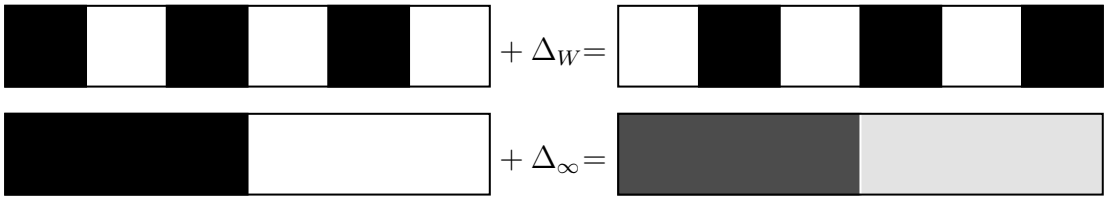


Figure 5.1: A minimal example exemplifying the difference between Wasserstein perturbations and ℓ_∞ perturbations on an image with six pixels. The top example utilizes a perturbation Δ_W to shift the image one pixel to the right, which is small with respect to Wasserstein distance since each pixel moved a minimal amount, but large with respect to ℓ_∞ distance since each pixel changed a maximal amount. In contrast, the bottom example utilizes a perturbation Δ_∞ which changes all pixels to be grayer. This is small with respect to ℓ_∞ distance, since each pixel changes by a small amount, but large with respect to Wasserstein distance, since the mass on each pixel on the left had to move halfway across the image to the right.

robustness between models trained against different threat models, finding that robustness against one perturbation type may even hurt another perturbation type. It is possible to have a provable defense against all ℓ_p norms for $p \geq 1$ simultaneously using a regularization term, however it suffers from looseness in the bound [Croce and Hein, 2019b]. Finally, Tramèr and Boneh [2019] concurrently studied the theoretical and empirical trade-offs of adversarial robustness in various settings when defending against two adversaries at a time for CIFAR10, and is most similar to our work but in a more limited setting against weaker adversaries.

We find that it is indeed possible to learn a model which is simultaneously robust to a union of threat models with adversarial training. Although we find that simple generalizations of adversarial training to multiple threat models can already achieve some degree of robustness against the union, the training procedure may not find the optimal balance between different threat models and converge to a poor local optima. We find that a slightly modified PGD-based algorithm called multi steepest descent (MSD) can help improve adversarial training in this regime. The approach more naturally incorporates the different perturbation sets within the PGD iterates, allowing us to further improve the empirical performance of adversarial training against the union over the simple baselines, and the robustness results have been externally validated by other work [Stutz et al., 2019].

5.1 Wasserstein adversarial examples

The crux of the work in this section relies on offering a fundamentally different type of adversarial example from typical, ℓ_p perturbations: the Wasserstein adversarial example.

5.1.1 Wasserstein distance

The Wasserstein distance (also referred to as the Earth mover's distance) is an optimal transport problem that can be intuitively understood in the context of distributions as the minimum cost of moving probability mass to change one distribution into another. When applied to images, this

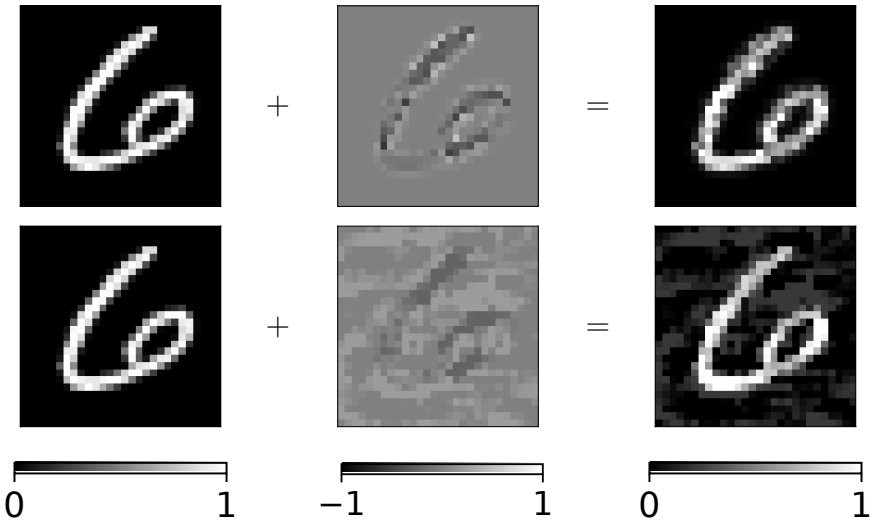


Figure 5.2: A comparison of a Wasserstein (top) vs an ℓ_∞ (bottom) adversarial example for an MNIST classifier (for $\epsilon = 0.4$ and 0.3 respectively), showing the original image (left), the added perturbation (middle), and the final perturbed image (right). We find that the Wasserstein perturbation has a structure reflecting the actual content of the image, whereas the ℓ_∞ perturbation also attacks the background pixels.

can be interpreted as the cost of moving pixel mass from one pixel to another another, where the cost increases with distance.

More specifically, let $x, y \in \mathbb{R}^n_+$ be two non-negative data points such that $\sum_i x_i = \sum_j y_j = 1$, so images and other inputs need to be normalized, and let $C \in \mathbb{R}^{n \times n}_+$ be some non-negative cost matrix where C_{ij} encodes the cost of moving mass from x_i to y_j. Then, the Wasserstein distance $d_{\mathcal{W}}$ between x and y is defined to be

$$\begin{aligned} d_{\mathcal{W}}(x, y) = \min_{\Pi \in \mathbb{R}^{n \times n}_+} \langle \Pi, C \rangle \\ \text{subject to } \Pi 1 = x, \;\; \Pi^T 1 = y \end{aligned} \tag{5.1}$$

where the minimization over transport plans Π, whose entries Π_{ij} encode how the mass moves from x_i to y_j. Then, we can define the Wasserstein ball with radius ϵ as

$$\mathcal{B}_{\mathcal{W}}(x, \epsilon) = \{x + \Delta : d_{\mathcal{W}}(x, x + \Delta) \leq \epsilon\} \tag{5.2}$$

5.1.2 Projection onto the Wasserstein ball with entropy regularization

In order to generate Wasserstein adversarial examples, we can run the projected gradient descent attack from Chapter 2, dropping in the Wasserstein ball $\mathcal{B}_{\mathcal{W}}$ from Equation (5.2) in place of $\mathcal{B}$. However, while projections onto regions such as ℓ_∞ and ℓ_2 balls are straightforward and have closed form computations, simply computing the Wasserstein distance itself requires solving an optimization problem. Thus, the first natural requirement to generating Wasserstein adversarial examples is to derive an *efficient* way to project examples onto a Wasserstein ball of radius ϵ.

Specifically, projecting w onto the Wasserstein ball around x with radius ϵ and transport cost matrix C can be written as solving the following optimization problem:

$$\begin{aligned} \underset{z\in\mathbb{R}_+^n,\Pi\in\mathbb{R}_+^{n\times n}}{\text{minimize}} \quad & \frac{1}{2}\|w-z\|_2^2 \\ \text{subject to} \quad & \Pi 1 = x, \quad \Pi^T 1 = z \\ & \langle \Pi, C\rangle \leq \epsilon \end{aligned} \tag{5.3}$$

While we could directly solve this optimization problem (using an off-the-shelf quadratic programming solver), this is prohibitively expensive to do for every iteration of projected gradient descent, especially since there is a quadratic number of variables. However, Cuturi [2013] showed that the standard Wasserstein distance problem from Equation (5.1) can be approximately solved efficiently by subtracting an entropy regularization term on the transport plan W, and using the Sinkhorn-Knopp matrix scaling algorithm. Motivated by these results, instead of solving the projection problem in Equation (5.3) exactly, the key contribution that allows us to do the projection efficiently is to instead solve the following entropy-regularized projection problem:

$$\begin{aligned} \underset{z\in\mathbb{R}_+^n,\Pi\in\mathbb{R}_+^{n\times n}}{\text{minimize}} \quad & \frac{1}{2}\|w-z\|_2^2 + \frac{1}{\lambda}\sum_{ij}\Pi_{ij}\log(\Pi_{ij}) \\ \text{subject to} \quad & \Pi 1 = x, \quad \Pi^T 1 = z \\ & \langle \Pi, C\rangle \leq \epsilon. \end{aligned} \tag{5.4}$$

Although this is an *approximate* projection onto the Wasserstein ball, importantly, the looseness in the approximation is only in finding the projection z which is closest (in ℓ_2 norm) to the original example x. All feasible points, including the optimal solution, are still within the actual ϵ-Wasserstein ball, so examples generated using the approximate projection are still within the Wasserstein threat model.

5.1.3 The dual of entropy regularized projections onto Wasserstein balls

Using the method of Lagrange multipliers, we can introduce dual variables (α, β, ψ) and derive an equivalent dual problem in Lemma 2.

Lemma 2. *The dual of the entropy-regularized Wasserstein projection problem in Equation (5.4) is*

$$\underset{\alpha,\beta\in\mathbb{R}^n,\psi\in\mathbb{R}_+}{\text{maximize}} \; g(\alpha,\beta,\psi) \tag{5.5}$$

where

$$\begin{aligned} g(\alpha,\beta,\psi) = & -\frac{1}{2\lambda}\|\beta\|_2^2 - \psi\epsilon + \alpha^T x + \beta^T w \\ & -\sum_{ij}\exp(\alpha_i)\exp(-\psi C_{ij} - 1)\exp(\beta_j) \end{aligned} \tag{5.6}$$

Proof. For convenience, we multiply the objective by λ and solve this problem instead:

$$\begin{aligned} \underset{z\in\mathbb{R}^n_+,\Pi\in\mathbb{R}^{n\times n}_+}{\text{minimize}} \quad & \frac{\lambda}{2}\|w-z\|_2^2 + \sum_{ij}\Pi_{ij}\log(\Pi_{ij}) \\ \text{subject to} \quad & \Pi 1 = x \\ & \Pi^T 1 = z \\ & \langle \Pi, C\rangle \le \epsilon. \end{aligned} \tag{5.7}$$

Introducing dual variables (α, β, ψ) where $\psi \ge 0$, the Lagrangian is

$$\begin{aligned} & L(z,\Pi,\alpha,\beta,\psi) \\ =& \frac{\lambda}{2}\|w-z\|_2^2 + \sum_{ij}\Pi_{ij}\log(\Pi_{ij}) + \psi(\langle\Pi, C\rangle - \epsilon) \\ & + \alpha^T(x-\Pi 1) + \beta^T(z-\Pi^T 1). \end{aligned} \tag{5.8}$$

The KKT optimality conditions are now

$$\begin{aligned} \frac{\partial L}{\partial \Pi_{ij}} &= \psi C_{ij} + (1+\log(\Pi_{ij})) - \alpha_i - \beta_j = 0 \\ \frac{\partial L}{\partial z_j} &= \lambda(z_j - w_j) + \beta_j = 0 \end{aligned} \tag{5.9}$$

so at optimality, we must have

$$\begin{aligned} \Pi_{ij} &= \exp(\alpha_i)\exp(-\psi C_{ij} - 1)\exp(\beta_j) \\ z &= -\frac{\beta}{\lambda} + w \end{aligned} \tag{5.10}$$

Plugging in the optimality conditions, we get

$$\begin{aligned} & L(z^*,\Pi^*,\alpha,\beta,\psi) \\ =& -\frac{1}{2\lambda}\|\beta\|_2^2 - \psi\epsilon + \alpha^T x + \beta^T w \\ & - \sum_{ij}\exp(\alpha_i)\exp(-\psi C_{ij} - 1)\exp(\beta_j) \\ =& g(\alpha,\beta,\psi) \end{aligned} \tag{5.11}$$

so the dual problem is to maximize g over $\alpha, \beta, \psi \ge 0$. □

Once we have solved the dual problem, we can recover the primal solution (to get the actual projection), which is described in Lemma 3.

Lemma 3. *Suppose $\alpha^*, \beta^*, \psi^*$ maximize the dual problem g in Equation (5.6). Then,*

$$\begin{aligned} z_i^* &= w_i - \beta_i/\lambda \\ \Pi_{ij}^* &= \exp(\alpha_i^*)\exp(-\psi^* C_{ij} - 1)\exp(\beta_j^*) \end{aligned} \tag{5.12}$$

are the corresponding solutions that minimize the primal problem in Equation (5.4).

Proof. These equations follow directly from the KKT optimality conditions from Equation (5.10). □

5.1.4 Projected Sinkhorn iteration to solve the dual

Note that the dual problem here differs from the traditional dual problem for Sinkhorn iterates by having an additional quadratic term on β and an additional dual variable ψ. Nonetheless, we can still derive a Sinkhorn-like algorithm by performing block coordinate ascent over the dual variables. Note that since this is a strictly convex problem, to get the α and β iterates we can set the gradient to 0 and solve for α and β. Specifically, the derivative with respect to α is

$$\frac{\partial g}{\partial \alpha_i} = x_i - \exp(\alpha_i) \sum_j \exp(-\psi C_{ij} - 1) \exp(\beta_j) \tag{5.13}$$

and so setting this to 0 and solving for α_i gives the following α iterate.

$$\arg\max_{\alpha_i} g(\alpha, \beta, \psi) = \log(x_i) - \log\left(\sum_j \exp(-\psi C_{ij} - 1) \exp(\beta_j) \right), \tag{5.14}$$

which is identical (up to a log transformation of variables) to the original Sinkhorn iterate proposed in Cuturi [2013]. Similarly, the derivative with respect to β is

$$\frac{\partial g}{\partial \beta_j} = -\frac{1}{\lambda}\beta + w - \exp(\beta_j) \sum_i \exp(\alpha_i) \exp(-\psi C_{ij} - 1) \tag{5.15}$$

and setting this to 0 and solving for β_j gives the β iterate (this step can be done analytically using a symbolic solver, we used Mathematica):

$$\arg\max_{\beta_j} g(\alpha, \beta, \psi) = \lambda w_j - W\left(\lambda \exp(\lambda w_j) \sum_i \exp(\alpha_i) \exp(-\psi C_{ij} - 1) \right) \tag{5.16}$$

where W is the Lambert W function, which is defined as the inverse of $f(x) = xe^x$. Finally, since ψ cannot be solved for analytically, we can perform the following Newton step

$$\psi' = \psi - t \cdot \frac{\partial g / \partial \psi}{\partial^2 g / \partial \psi^2} \tag{5.17}$$

where

$$\begin{aligned} \partial g / \partial \psi &= -\epsilon + \sum_{ij} \exp(\alpha_i) C_{ij} \exp(-\psi C_{ij}) \exp(\beta_j) \\ \partial^2 g / \partial \psi^2 &= -\sum_{ij} \exp(\alpha_i) C_{ij}^2 \exp(-\psi C_{ij}) \exp(\beta_j) \end{aligned} \tag{5.18}$$

and where t is small enough such that $\psi' \geq 0$.

The whole algorithm can then be vectorized and implemented as Algorithm 8, which we call projected Sinkhorn iterates. The algorithm uses a simple line search to ensure that the constraint $\psi \geq 0$ is not violated. Each iteration has 8 $O(n^2)$ operations (matrix-vector product or matrix-matrix element-wise product), in comparison to the original Sinkhorn iteration which has 2 matrix-vector products.

Algorithm 8 Projected Sinkhorn iteration to project x onto the ϵ Wasserstein ball around y. We use $\cdot$ to denote element-wise multiplication. The $\log$ and $\exp$ operators also apply element-wise.

input: $x, w \in \mathbb{R}^n, C \in \mathbb{C}^{n \times n}, \lambda \in \mathbb{R}$
Initialize $\alpha_i, \beta_i \coloneqq \log(1/n)$ for $i = 1, \ldots, n$ and $\psi \coloneqq 1$
$u, v \coloneqq \exp(\alpha), \exp(\beta)$
while α, β, ψ not converged **do**
  // update K
  $K_\psi \coloneqq \exp(-\psi C - 1)$

  // block coordinate descent iterates
  $\alpha \coloneqq \log(x) - \log(K_\psi v)$
  $u \coloneqq \exp(\alpha)$
  $\beta \coloneqq \lambda w - W\left(u^T K_\psi \cdot \lambda \exp(\lambda w)\right)$
  $v \coloneqq \exp(\beta)$

  // Newton step
  $g \coloneqq -\epsilon + u^T (C \cdot K_\psi) v$
  $h \coloneqq -u^T (C \cdot C \cdot K_\psi) v$

  // ensure $\psi \geq 0$
  $\alpha \coloneqq 1$
  while $\psi - \alpha g/h < 0$ **do**
    $\alpha \coloneqq \alpha/2$
  end while
  $\psi \coloneqq \psi - \alpha g/h$
end while
return: $w - \beta/\lambda$

Matrix scaling interpretation The original Sinkhorn iteration has a natural interpretation as a matrix scaling algorithm, iteratively rescaling the rows and columns of a matrix to achieve the target distributions. To see how the Projected Sinkhorn iteration is also a (modified) matrix scaling algorithm, we can interpret certain quantities before optimality as primal iterates. Namely, at each iteration t, let

$$\begin{aligned} z_i^{(t)} &= w_i - \beta_i^{(t)}/\lambda \\ \Pi_{ij}^{(t)} &= \exp(\alpha^{(t)}) \exp(-\psi^{(t)} C_{ij} - 1) \exp(\beta^{(t)}) \end{aligned} \tag{5.19}$$

Then, since the α and β steps are equivalent to setting Equations (5.13) and (5.15) to 0, we know that after an update for $\alpha^{(t)}$, we have that

$$x_i = \sum_j \Pi_{ij}^{(t)} \tag{5.20}$$

so the α step rescales the transport matrix to sum to x. Similarly, after an update for $\beta^{(t)}$, we have that

$$z_i^{(t)} = \sum_i \Pi_{ij}^{(t)} \tag{5.21}$$

which is a rescaling of the transport matrix to sum to the projected value. Lastly, the numerator of the $\psi^{(t)}$ step can be rewritten as

$$\psi^{(t+1)} = \psi^{(t)} + t \cdot \frac{\langle \Pi^{(t)}, C \rangle - \epsilon}{\langle \Pi^{(t)}, C \cdot C \rangle} \tag{5.22}$$

as a simple adjustment based on whether the current transport plan $\Pi^{(t)}$ is above or below the maximum threshold ϵ.

5.1.5 Local transport plans

The quadratic runtime dependence on input dimension can grow quickly, and this is especially true for images. Rather than allowing transport plans to move mass to and from any pair of pixels, we instead restrict the transport plan to move mass only within a $k \times k$ region of the originating pixel, similar in spirit to a convolutional filter. As a result, the cost matrix C only needs to define the cost within a $k \times k$ region, and we can utilize tools used for convolutional filters to efficiently apply the cost to each $k \times k$ region. This reduces the computational complexity of each iteration to $O(nk^2)$. For images with more than one channel, we can use the same transport plan for each channel and only allow transport within a channel, so the cost matrix remains $k \times k$. For 5×5 local transport plans on CIFAR10, the projected Sinkhorn iterates typically converge in around 30-40 iterations, taking about 0.02 seconds per iteration on a Titan X for minibatches of size 100. Note that if we use a cost matrix C that reflects the 1-Wasserstein distance, then this problem could be solved even more efficiently using Kantrovich duality, however we use this formulation to enable more general p-Wasserstein distances, or even non-standard cost matrices.

Projected gradient descent on the Wasserstein ball With local transport plans, the method is fast enough to be used within a projected gradient descent routine to generate adversarial examples on images, and further used for adversarial training as in Algorithm 5 from Chapter 4 (using steepest descent with respect to ℓ_∞ norm), except that we do an approximate projection onto the Wasserstein ball using Algorithm 8.

5.1.6 Provable defense with conjugate Sinkhorn iteration

Lastly, we present some analysis on how this attack fits into the context of provable defenses, along with a negative result demonstrating a fundamental gap that needs to be solved. The Wasserstein attack can be naturally incorporated into duality based defenses: the work from Chapter 3 of this dissertation shows that to generate certificates which can defend against inputs other than ℓ_p norm, one only needs to solve the following optimization problem:

$$\max_{x \in B(x,\epsilon)} -x^T y \tag{5.23}$$

for some constant y and for some perturbation region $B(x, \epsilon)$ (a similar approach can be taken to adapt the dual verification from Dvijotham et al. [2018a]). For the Wasserstein ball, this is highly similar to the problem of projecting onto the Wasserstein ball from Equation (5.4), with a linear objective instead of a quadratic objective and fewer variables. In fact, a Sinkhorn-like algorithm can be derived to solve this problem, which ends up being a simplified version of Algorithm 8.

Conjugate Sinkhorn iteration By subtracting the same entropy term to the conjugate objective from Equation (5.23), we can get a problem similar to that of projecting onto the Wasserstein ball.

$$\begin{aligned} \underset{z \in \mathbb{R}^n_+, \Pi \in \mathbb{R}^{n \times n}_+}{\text{minimize}} \quad & -\lambda z^T y + \sum_{ij} \Pi_{ij} \log(\Pi_{ij}) \\ \text{subject to} \quad & \Pi 1 = x \\ & \Pi^T 1 = z \\ & \langle \Pi, C \rangle \leq \epsilon. \end{aligned} \tag{5.24}$$

where again we've multiplied the objective by λ for convenience. Following the same framework as before, we introduce dual variables (α, β, ψ) where $\psi \geq 0$, to construct the Lagrangian as

$$\begin{aligned} & L(z, \Pi, \alpha, \beta, \psi) \\ =& -\lambda z^T y + \sum_{ij} \Pi_{ij} \log(\Pi_{ij}) + \psi(\langle \Pi, C \rangle - \epsilon) \\ & + \alpha^T (x - \Pi 1) + \beta^T (z - \Pi^T 1). \end{aligned} \tag{5.25}$$

Note that since all the terms with Π_{ij} are the same, the corresponding KKT optimality condition for Π_{ij} also remains the same. The only part that changes is the optimality condition for z, which becomes

$$\beta = \lambda y \tag{5.26}$$

Plugging the optimality conditions into the Lagrangian, we get the following dual problem:

$$\begin{aligned} & L(z^*, \Pi^*, \alpha, \beta, \psi) \\ =& -\psi \epsilon + \alpha^T x \\ & - \sum_{ij} \exp(\alpha_i) \exp(-\psi C_{ij} - 1) \exp(\beta_j) \\ =& g(\alpha, \psi) \end{aligned} \tag{5.27}$$

Finally, if we minimize this with respect to α and ψ we get exactly the same update steps as the Projected Sinkhorn iteration. Consequently, the Conjugate Sinkhorn iteration is identical to the Projected Sinkhorn iteration except that we replace the β step with the fixed value $\beta = \lambda y$.

Fundamental limitations However, there is a fundamental obstacle towards generating provable certificates against Wasserstein attacks: these defenses (and many other, non-duality based

Table 5.1: Classification accuracies for models used in the experiments.

DATA SET	MODEL	NOMINAL ACCURACY
MNIST	STANDARD	98.90%
	BINARIZE	98.73%
	ROBUST	98.20%
	ADV. TRAINING	96.95%
CIFAR10	STANDARD	94.70%
	ROBUST	66.33%
	ADV. TRAINING	80.69%

approaches) depend heavily on propagating interval bounds from the input space through the network, in order to efficiently bound the output of ReLU units. This concept is inherently at odds with the notion of Wasserstein distance: a "small" Wasserstein ball can use a low-cost transport plan to move all the mass at a single pixel to its neighbors, or vice versa. As a result, when converting a Wasserstein ball to interval constraints, the interval bounds immediately become vacuous: each individual pixel can attain their minimum or maximum value under some ϵ cost transport plan. In order to guarantee robustness against Wasserstein adversarial attacks, significant progress must be made to overcome this limitation.

5.2 Experiments for Wasserstein adversarial examples

In this section, we run the Wasserstein examples through a range of typical experiments in the literature of adversarial examples. Table 5.1 summarizes the nominal error rates obtained by all considered models. All experiments can be run on a single GPU, and all code for the experiments is available at `https://github.com/locuslab/projected_sinkhorn`.

Architectures For MNIST we used the convolutional ReLU architecture used in Wong and Kolter [2017], with two convolutional layers with 16 and 32 4×4 filters each, followed by a fully connected layer with 100 units, which achieves a nominal accuracy of 98.89%. For CIFAR10 we focused on the standard ResNet18 architecture [He et al., 2016a], which achieves a nominal accuracy of 94.76%.

Hyperparameters For all experiments in this section, we focused on using 5×5 local transport plans for the Wasserstein ball, and used an entropy regularization constant of 1000 for MNIST and 3000 for CIFAR10. The cost matrix used for transporting between pixels is taken to be the 2-norm of the distance in pixel space (e.g. the cost of going from pixel (i, j) to (k, l) is $\sqrt{|i-j|^2 + |k-l|^2}$), which makes the optimal transport cost a metric more formally known as the 1-Wasserstein distance.

Evaluation at test time We use the follow evaluation procedure to attack models with projected gradient descent on the Wasserstein ball. For each MNIST example, we start with $\epsilon = 0.3$

and increase it by a factor of 1.1 every 10 iterations until either an adversarial example is found or until 200 iterations have passed, allowing for a maximum perturbation radius of $\epsilon = 2$. For CIFAR10, we start with $\epsilon = 0.001$ and increase it by a factor of 1.17 until either and adversarial example is found or until 400 iterations have passed, allowing for a maximum perturbation radius of $\epsilon = 0.53$.

5.2.1 Wasserstein robustness on MNIST

For MNIST, we consider a standard model, a model with binarization, a model provably robust to ℓ_∞ perturbations of at most $\epsilon = 0.1$, and an adversarially trained model. We provide a visual comparison of the Wasserstein adversarial examples generated on each of the four models in Figure 5.3. The susceptibility of all four models to the Wasserstein attack is plotted in Figure 5.4.

Adaptive ϵ During adversarial training for MNIST, we adopt an adaptive ϵ scheme to avoid selecting a specific ϵ. Specifically, to find an adversarial example, we first let $\epsilon = 0.1$ on the first iteration of projected gradient descent, and increase it by a factor of 1.4 every 5 iterations. We terminate the projected gradient descent algorithm when either an adversarial example is found, or when 50 iterations have passed, allowing ϵ to take on values in the range $[0.1, 2.1]$

Optimizer hyperparameters To update the model weights during adversarial training, we use the SGD optimizer with 0.9 momentum and 0.0005 weight decay, and batch sizes of 128. We begin with a learning rate of 0.1, reduce it to 0.01 after 10 epochs.

Standard model and binarization For MNIST, despite restricting the transport plan to local 5×5 regions, a standard model is easily attacked by Wasserstein adversarial examples. In Figure 5.4, we see that Wasserstein attacks with $\epsilon = 0.5$ can successfully attack a typical MNIST classifier 50% of the time, which goes up to 94% for $\epsilon = 1$. A Wasserstein radius of $\epsilon = 0.5$ can be intuitively understood as moving 50% of the pixel mass over by 1 pixel, or alternatively moving less than 50% of the pixel mass more than 1 pixel. Furthermore, while preprocessing images with binarization is often seen as a way to trivialize adversarial examples on MNIST, we find that it performs only marginally better than the standard model against Wasserstein perturbations.

ℓ_∞ robust model We also run the attack on the model trained by Wong et al. [2018], which is guaranteed to be provably robust against ℓ_∞ perturbations with $\epsilon \leq 0.1$. While not specifically trained against Wasserstein perturbations, in Figure 5.4 we find that it is substantially more robust than either the standard or the binarized model, requiring a significantly larger ϵ to have the same attack success rate.

Adversarial training Finally, we apply this attack as an inner procedure within an adversarial training framework for MNIST. To save on computation, during training we adopt a weaker adversary and use only 50 iterations of projected gradient descent. We also let ϵ grow within a range and train on the first adversarial example found (essentially a budget version of the attack

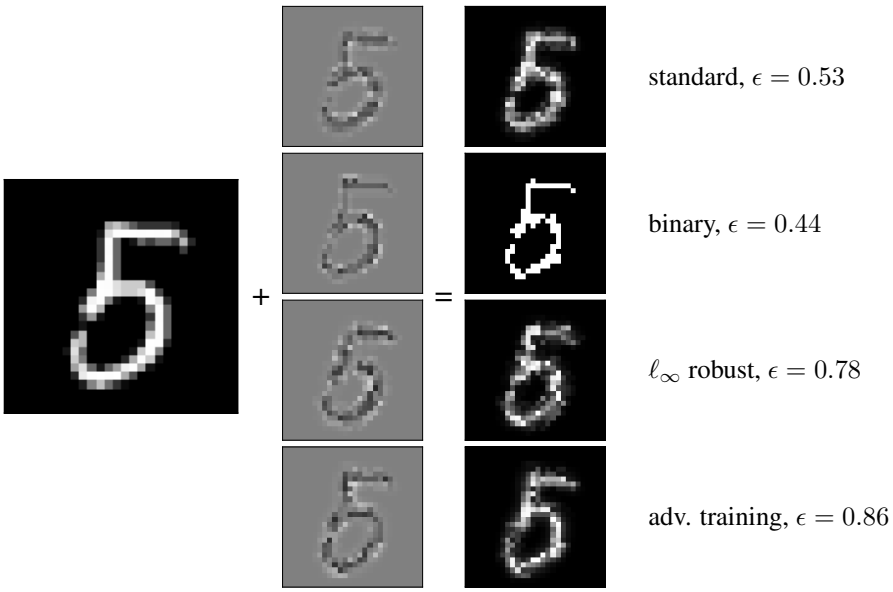


Figure 5.3: Wasserstein adversarial examples on the MNIST dataset for the four different models. Note that the ℓ_∞ robust and the adversarially trained models require a much larger ϵ radius for the Wasserstein ball in order to generate an adversarial example. Each model classifies the corresponding perturbed example as an 8 instead of a 5, except for the first one which classifies the perturbed example as a 6.

used at test time). We find that the adversarially trained model is empirically the most well defended against this attack of all four models, and cannot be attacked down to 0% accuracy (Figure 5.4).

5.2.2 Wasserstein robustness on CIFAR10

For CIFAR10, we consider a standard model, a model provably robust to ℓ_∞ perturbations of at most $\epsilon = 2/255$, and an adversarially trained model. We plot the susceptibility of each model to the Wasserstein attack in Figure 5.5.

Adaptive ϵ We also use an adaptive ϵ scheme for adversarial training in CIFAR10. Specifically, we let $\epsilon = 0.01$ on the first iteration of projected gradient descent, and increase it by a factor of 1.5 every 5 iterations. Similar to MNIST, we terminate the projected gradient descent algorithm when either an adversarial example is found, or 50 iterations have passed, allowing ϵ to take on values in the range $[0.01, 0.38]$.

Optimizer hyperparameters Similar to MNIST, to update the model weights, we use the SGD optimizer with 0.9 momentum and 0.0005 weight decay, and batch sizes of 128. The learning rate is also the same as in MNIST, starting at 0.1, and reducing to 0.01 after 10 epochs.

Standard model We find that for a standard ResNet18 CIFAR10 classifier, a perturbation radius of as little as 0.01 is enough to misclassify 25% of the examples, while a radius of 0.1 is

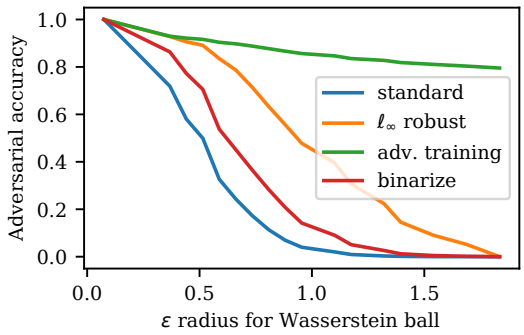


Figure 5.4: Adversarial accuracy of various models on MNIST when attacked by a Wasserstein adversary over varying sizes of ϵ-Wasserstein balls. We find that all models not trained with adversarial training against this attack eventually achieve 0% accuracy, however we do observe that models trained to be provably robust against ℓ_∞ perturbations are still somewhat more robust than standard models, or models utilizing binarization as a defense.

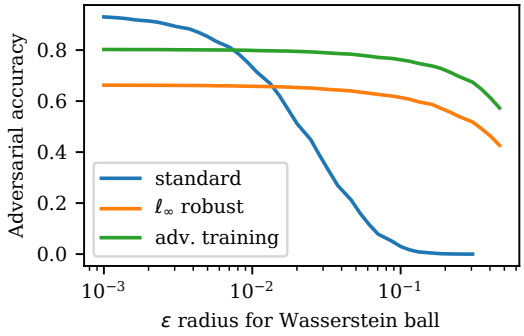


Figure 5.5: Adversarial accuracy of various models on the CIFAR10 dataset when attacked by a Wasserstein adversary. We find that the model trained to be robust against ℓ_∞ perturbations is actually more robust than adversarial training.

enough to fool the classifier 97% of the time (Figure 5.5). Despite being such a small ϵ, we see in Figure 5.6 that the structure of the perturbations still reflect the actual content of the images, though certain classes require larger magnitudes of change than others.

ℓ_∞ robust model We further empirically evaluate the attack on a model that was trained to be provably robust against ℓ_∞ perturbations. We use the models weights from Wong et al. [2018], which are trained to be provably robust against ℓ_∞ perturbations of at most $\epsilon = 2/255$. Further note that this CIFAR10 model actually is a smaller ResNet than the ResNet18 architecture considered in this paper, and consists of 4 residual blocks with 16, 16, 32, and 64 filters. Nonetheless, we find that while the model suffers from poor nominal accuracy (achieving only 66% accuracy on unperturbed examples as noted in Table 5.1), the robustness against ℓ_∞ attacks remarkably seems to transfer quite well to robustness against Wasserstein attacks in the CIFAR10 setting, achieving 61% adversarial accuracy for $\epsilon = 0.1$ in comparison to 3% for the standard model.

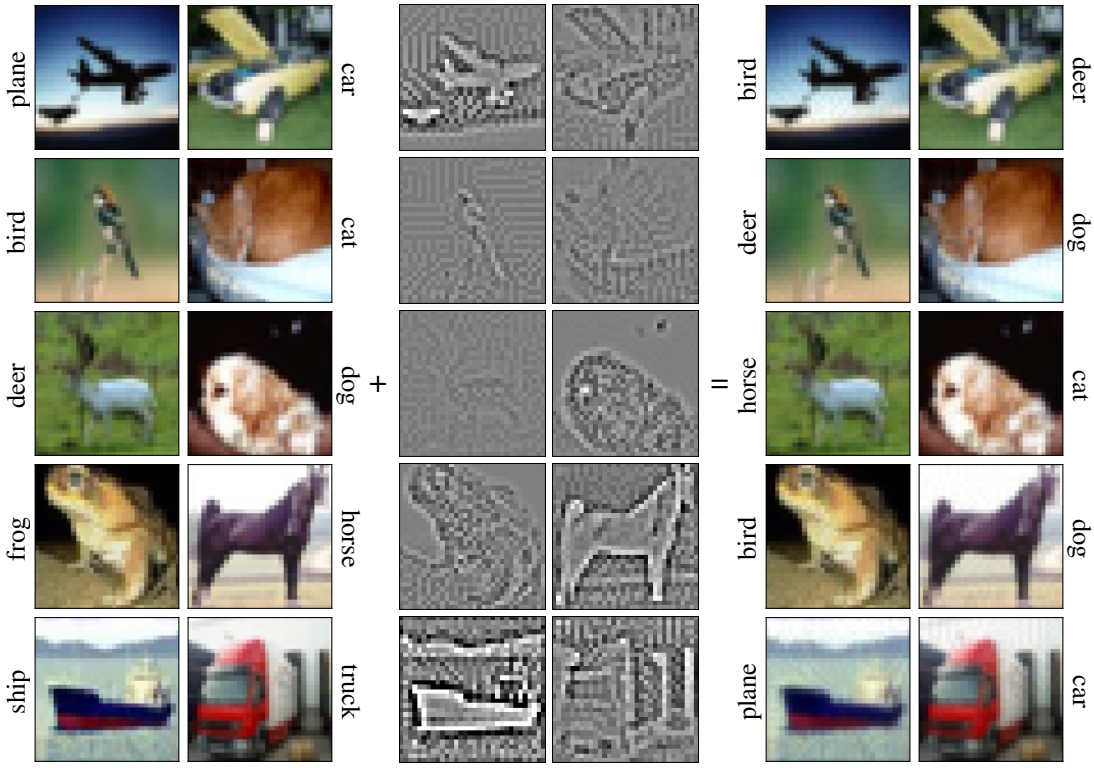


Figure 5.6: Wasserstein adversarial examples for CIFAR10 on a typical ResNet18 for all 10 classes. The perturbations here represents the total change across all three channels, where total change is plotted within the range ± 0.165 (the maximum total change observed in a single pixel) for images scaled to [0,1].

Adversarial training We find that adversarial training here is also able to defend against this attack, and at the same threshold of $\epsilon = 0.1$, we find that the adversarial accuracy has been improved from 3% to 76%.

5.2.3 Using adaptive perturbation budgets during adversarial training

A commonly asked question of models trained to be robust against adversarial examples is "what if the adversary has a perturbation budget of $\epsilon + \delta$ instead of ϵ?" This is referring to a "robustness cliff," where a model trained against an ϵ strong adversary has a sharp drop in robustness when attacked by an adversary with a slightly larger budget. To address this, we advocate for the slightly modified version of typical adversarial training used in this work: rather than picking a fixed ϵ and running projected gradient descent, we instead allow for an adversarial to have a range of $\epsilon \in [\epsilon_{min}, \epsilon_{max}]$. To do this, we begin with $\epsilon = \epsilon_{\min}$, and then gradually increase it by a multiplicative factor γ until either an adversarial example is found or until ϵ_{max} is reached. While similar ideas have been used before for evaluating model robustness, we specifically advocate for using this schema *during adversarial training*. This has the advantage of extending robustness of the classifier beyond a single ϵ threshold, allowing a model to achieve a potentially higher robustness threshold while not being significantly harmed by "impossible" adversarial examples.

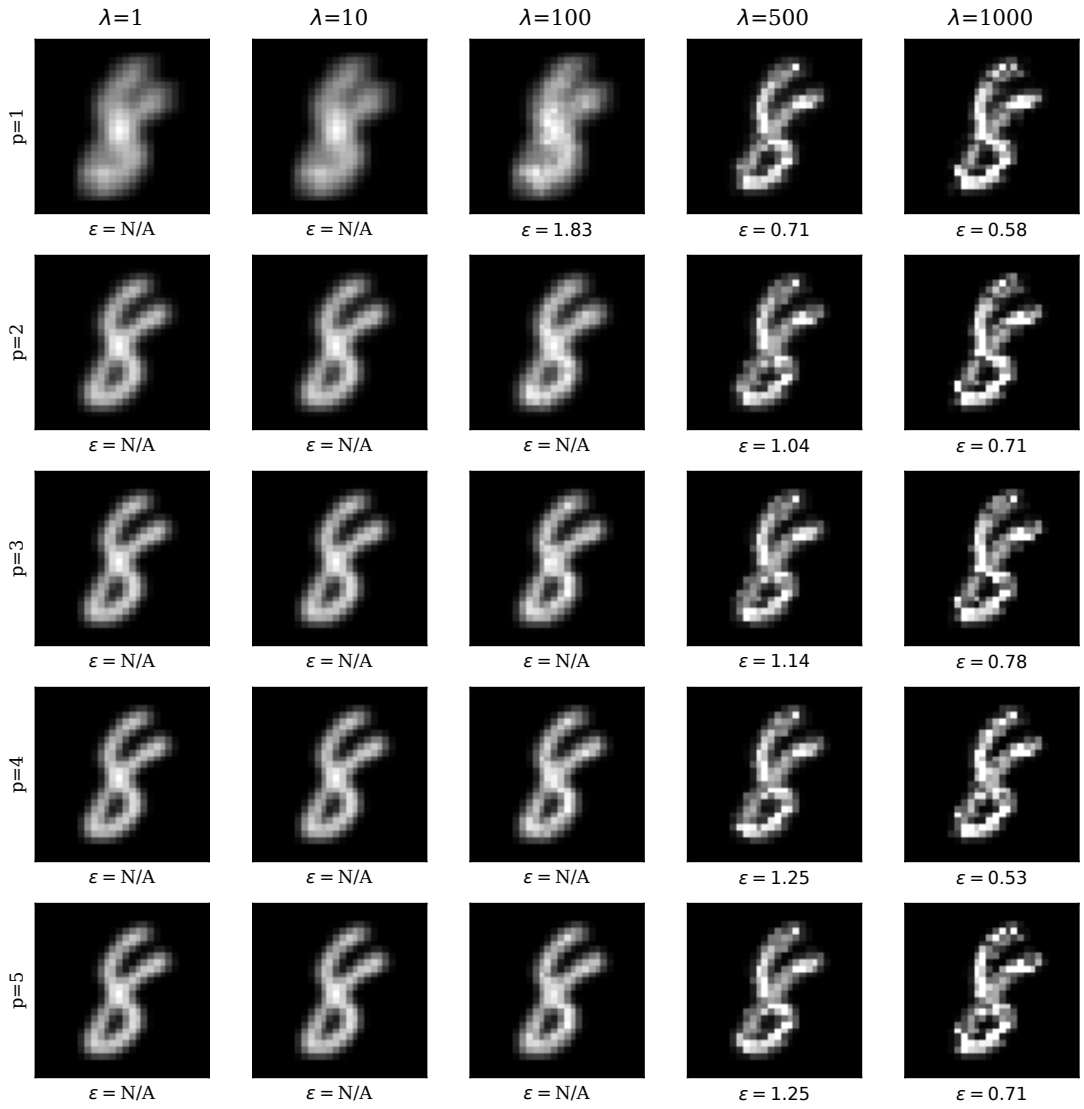


Figure 5.7: A plot of the adversarial examples generated with different p-Wasserstein metrics used for the cost matrix C and different regularization parameters λ. Note that when regularization is low, the image becomes blurred, and it is harder to find adversarial examples. In contrast, changing p does not seem to make any significant changes.

5.2.4 Effect of λ and C

We study the effect of entropy hyperparameter λ and the cost matrix C. First, note that λ could be any positive value. Furthermore, note that to construct C we used the 2-norm which reflects the 1-Wasserstein metric, but in theory we could use any p-Wasserstein metric, where the the cost of moving from pixel (i, j) to (k, l) is $(|i - j|^2 + |k - l|^2)^{p/2}$. Figure 5.7 shows the effects of λ and p on both the adversarial example and the radius at which it was found for varying values of $\lambda = [1, 10, 100, 500, 1000]$ and $p = [1, 2, 3, 4, 5]$.

We find that it is important to ensure that λ is large enough, otherwise the projection of the image is excessively blurred. In addition to qualitative changes, smaller λ seems to make it

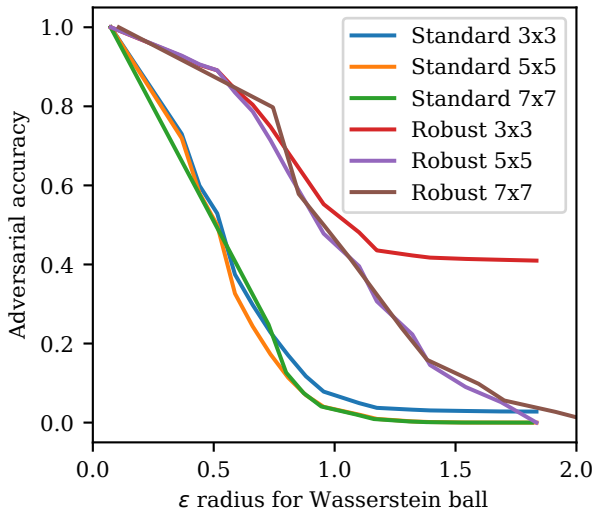


Figure 5.8: Adversarial accuracy of a standard model and a model trained to be provably robust against ℓ_∞ attacks for different sizes of transport plans. In most cases the size of the transport plan doesn't seem to matter, except for the 3×3 local transport plan. In this case, the adversary isn't quite able to reach 0% accuracy for the standard model, reaching 2.8% for for $\epsilon = 1.83$. The adversary is also unable to attack the robust MNIST model, bottoming out at 41% adversarial accuracy at $\epsilon = 1.83$.

harder to find Wasserstein adversarial examples, making the ϵ radius go up as λ gets smaller. In fact, for $\lambda = (1, 10)$ and almost all of $\lambda = 100$, the blurring is so severe that no adversarial example can be found.

In contrast, we find that increasing p for the Wasserstein distance used in the cost matrix C seems to make the images more "blocky". Specifically, as p gets higher tested, more pixels seem to be moved in larger amounts. This seems to counteract the blurring observed for low λ to some degree. Naturally, the ϵ radius also grows since the overall cost of the transport plan has gone up.

5.2.5 Size of local transport plan

In this section we explore the effects of different sized transport plans. Although we primarily used a 5×5 local transport plan, this could easily be something else, e.g. 3×3 or 7×7. We can see a comparison on the robustness of a standard and the ℓ_∞ robust model against these different sized transport plans in Figure 5.8, using $\lambda = 1000$. We observe that while 3×3 transport plans have difficulty attacking the robust MNIST model, all other plan sizes seem to have similar performance. This is primarily due to the cost matrix C: since the cost of moving pixel mass scales with the distance in pixel space, transport plans are already biased to be local anyways. As a result, as long as the local region is large enough to contain the optimal transport plan, restricting the transport plan to be local does not significantly restrict the Wasserstein adversarial example.

5.3 Defending against multiple threat models simultaneously

In this section, we describe adversarial training procedures for obtaining robustness against multiple threat models. More formally, let $\mathcal{S}$ represent a set of threat models, such that $p \in \mathcal{S}$ corresponds to the ℓ_p perturbation model $\Delta_{p,\epsilon}$, and let $\Delta_{\mathcal{S}} = \bigcup_{p \in \mathcal{S}} \Delta_{p,\epsilon}$ be the union of all perturbation models in $\mathcal{S}$. Note that the ϵ chosen for each ball is *not* typically the same, but we still use the same notation ϵ for simplicity, since the context will always make clear which ℓ_p-ball we are talking about. Then, the generalization of the robust optimization problem in Equation (2.7) to multiple perturbation models is

$$\min_{\theta} \sum_{i} \max_{\delta \in \Delta_{\mathcal{S}}} \ell(x_i + \delta; \theta). \tag{5.28}$$

The key difference is in the inner maximization, where the worst case adversarial loss is now taken over *multiple* ℓ_p perturbation models. In order to perform adversarial training, using the same motivational idea from Danskin's theorem, we can backpropagate through the inner maximization by first finding (empirically) the optimal perturbation,

$$\delta^* = \arg\max_{\delta \in \Delta_{\mathcal{S}}} \ell(x + \delta; \theta). \tag{5.29}$$

To find the optimal perturbation over the union of threat models, we begin by considering straightforward generalizations of standard adversarial training, which will use PGD to approximately solve the inner maximization over multiple adversaries.

5.3.1 Simple combinations of multiple perturbations

First, we propose two simple approaches to generalizing adversarial training to multiple threat models which were concurrently analyzed by Tramèr and Boneh [2019]. These methods already perform quite well in practice and are competitive with existing, state-of-the-art approaches without relying on complicated architectures, showing that adversarial training can in fact generalize to multiple threat models.

Worst-case perturbation One way to generalize adversarial training to multiple threat models is to use each threat model independently, and train on the adversarial perturbation that achieved the maximum loss. Specifically, for each adversary $p \in \mathcal{S}$, we solve the innermost maximization with an ℓ_p PGD adversary to get an approximate worst-case perturbation δ_p,

$$\delta_p = \arg\max_{\delta \in \Delta_{p,\epsilon}} \ell(x + \delta; \theta), \tag{5.30}$$

and then approximate the maximum over all adversaries as

$$\delta^* \approx \arg\max_{\delta_p} \ell(x + \delta_p; \theta). \tag{5.31}$$

When $|\mathcal{S}| = 1$, then this reduces to standard adversarial training. Note that if each PGD adversary solved their subproblem from Equation (5.30) exactly, then this is exactly the optimal perturbation $\delta^\star$.

Algorithm 9 Multi steepest descent for learning classifiers that are simultaneously robust to ℓ_p attacks for $p \in \mathcal{S}$

Input: classifier f_θ, data x, labels y
Parameters: ϵ_p, α_p for $p \in \mathcal{S}$, maximum iterations T, loss function ℓ
$\delta^{(0)} = 0$
for $t = 0 \dots T-1$ **do**
  for $p \in \mathcal{S}$ **do**
    $\delta_p^{(t+1)} = P_{\Delta_{p,\epsilon}}(\delta^{(t)} + v_p(\delta^{(t)}))$
  end for
  $\delta^{(t+1)} = \arg\max_{\delta_p^{(t+1)}} \ell(f_\theta(x + \delta_p^{(t+1)}), y)$
end for
return $\delta^{(T)}$

PGD augmentation with all perturbations Another way to generalize adversarial training is to train on all the adversarial perturbations for all $p \in \mathcal{S}$ to form a larger adversarial dataset. Specifically, instead of solving the robust problem for multiple adversaries in Equation (5.28), we instead solve

$$\min_\theta \sum_i \sum_{p \in \mathcal{S}} \max_{\delta \in \Delta_{p,\epsilon}} \ell(x_i + \delta; \theta) \tag{5.32}$$

by using individual ℓ_p PGD adversaries to approximate the inner maximization for each threat model. Again, this reduces to standard adversarial training when $|\mathcal{S}| = 1$.

While these methods work quite well in practice (which is shown later in Section 5.4), both approaches solve the inner maximization problem independently for each adversary, so each individual PGD adversary is not taking advantage of the fact that the perturbation region is enlarged by other threat models. To take advantage of the full perturbation region, we propose a modification to standard adversarial training, which combines information from all considered threat models into a single PGD adversary that is potentially stronger than the combination of independent adversaries.

5.3.2 Multi steepest descent

To create a PGD adversary with full knowledge of the perturbation region, we propose an algorithm that incorporates the different threat models within each step of projected steepest descent. Rather than generating adversarial examples for each threat model with separate PGD adversaries, the core idea is to create a single adversarial perturbation by simultaneously maximizing the worst case loss over all threat models at each projected steepest descent step. We call our method *multi steepest descent* (MSD), which can be summarized as the following iteration:

$$\begin{aligned} \delta_p^{(t+1)} &= P_{\Delta_{p,\epsilon}}(\delta^{(t)} + v_p(\delta^{(t)})) \;\text{ for }\; p \in \mathcal{S} \\ \delta^{(t+1)} &= \arg\max_{\delta_p^{(t+1)}} \ell(x + \delta_p^{(t+1)}) \end{aligned} \tag{5.33}$$

The key difference here is that at each iteration of MSD, we choose a projected steepest descent direction that maximizes the loss over all attack models $p \in \mathcal{S}$, whereas standard adversarial

training and the simpler approaches use comparatively myopic PGD subroutines that only use one threat model at a time. The full algorithm is in Algorithm 9, and can be used as a drop in replacement for standard PGD adversaries to learn robust classifiers with adversarial training.

5.3.3 Steepest descent and projections for ℓ_∞, ℓ_2, and ℓ_1 adversaries

In this section, we show what the steepest descent and projection steps are for ℓ_p adversaries for $p \in \{\infty, 2, 1\}$; these are standard results, but included for a complete description of the algorithms. Note that this differs slightly from the adversaries considered in Schott et al. [2019]: while they used an ℓ_0 adversary, we opted to use an ℓ_1 adversary with the same radius. The ℓ_0 ball with radius ϵ is contained within an ℓ_1 ball with the same radius, so achieving robustness against an ℓ_1 adversary is strictly more difficult.

ℓ_∞ space The direction of steepest descent with respect to the ℓ_∞ norm is

$$v_\infty(\delta) = \alpha \cdot \text{sign}(\nabla l(x + \delta; \theta)) \tag{5.34}$$

and the projection operator onto $\Delta_{\infty,\epsilon}$ is

$$\mathcal{P}_{\Delta_{\infty,\epsilon}}(\delta) = \text{clip}_{[-\epsilon,\epsilon]}(\delta) \tag{5.35}$$

ℓ_2 space The direction of steepest descent with respect to the ℓ_2 norm is

$$v_2(\delta) = \alpha \cdot \frac{\nabla \ell(x + \delta; \theta)}{\|\nabla \ell(x + \delta; \theta)\|_2} \tag{5.36}$$

and the projection operator onto the ℓ_2 ball around x is

$$\mathcal{P}_{\Delta_{2,\epsilon}}(\delta) = \epsilon \cdot \frac{\delta}{\max\{\epsilon, \|\delta\|_2\}} \tag{5.37}$$

ℓ_1 space The direction of steepest descent with respect to the ℓ_1 norm is

$$v_1(\delta) = \alpha \cdot \text{sign}\left(\frac{\partial \ell(x + \delta; \theta)}{\partial \delta_{i^\star}}\right) \cdot e_{i^\star} \tag{5.38}$$

where

$$i^\star = \arg\max_i |\nabla l(x + \delta; \theta)_i| \tag{5.39}$$

and e_{i^*} is a unit vector with a one in position i^*. Finally, the projection operator onto the ℓ_1 ball,

$$\mathcal{P}_{\Delta_{1,\epsilon}}(\delta) = \arg\min_{\delta' : \|\delta'\|_1 \leq \epsilon} \|\delta - \delta'\|_2^2, \tag{5.40}$$

can be solved with Algorithm 10, and we refer the reader to Duchi et al. [2008] for its derivation.

Algorithm 10 Projection of some perturbation $\delta \in \mathbb{R}^n$ onto the ℓ_1 ball with radius ϵ. We use $|\cdot|$ to denote element-wise absolute value.

Input: perturbation δ, radius ϵ
Sort $|\delta|$ into $\gamma : \gamma_1 \geq \gamma_2 \geq \cdots \geq \gamma_n$
$\rho := \max \left\{ j \in [n] : \gamma_j - \frac{1}{j} \left(\sum_{r=1}^{j} \gamma_r - \epsilon \right) > 0 \right\}$
$\eta := \frac{1}{\rho} \left(\sum_{i=1}^{\rho} \gamma_i - \epsilon \right)$
$z_i := \text{sign}(\delta_i) \max \{ \gamma_i - \eta, 0 \}$ for $i = 1 \ldots n$
return z

5.3.4 Special considerations for ℓ_1 steepest descent

Since the ℓ_0 and ℓ_1 attacks are not as universally consistent as ℓ_2 and ℓ_∞, we take the following two additional considerations to improve the attack. These improvements enhance the diversity of the attack and keep the attack from getting stuck at the boundary of pixel space.

Enhanced ℓ_1 steepest descent step Note that the steepest descent step for ℓ_1 only updates a single coordinate per step. This can be quite inefficient, as pointed out by Tramèr and Boneh [2019]. To tackle this issue, and also empirically improve the attack success rate, Tramèr and Boneh [2019] instead select the top k coordinates according to Equation 5.39 to update. In this work, we adopt a similar but slightly modified scheme: we randomly sample k to be some integer within some range $[k_1, k_2]$, and update each coordinate with step size $\alpha' = \alpha/k$. We find that the randomness induced by varying the number of coordinates aids in avoiding the gradient masking problem observed by Tramèr and Boneh [2019].

Restricting the steepest descent coordinate The steepest descent direction for both the ℓ_0 and ℓ_1 norm end up selecting a single coordinate direction to move the perturbation. However, if the perturbation is already at the boundary of pixel space (for MNIST, this is the range [0,1] for each pixel), then it's possible for the PGD adversary to get stuck in a loop trying to use the same descent direction to escape pixel space. To avoid this, we only allow the steepest descent directions for these two attacks to choose coordinates that keep the image in the range of real pixels.

5.4 Experiments for defending against multiple threat models

In this section, we present experimental results on using generalizations of adversarial training to achieve simultaneous robustness to ℓ_∞, ℓ_2, and ℓ_1 perturbations on the MNIST and CIFAR10 datasets. Our primary goal is to show that adversarial training can in fact be adapted to a union of perturbation models using standard architectures to achieve competitive results, without the pitfalls described by Schott et al. [2019]. Our results improve upon the state-of-the-art in three key ways. First, we can use simpler, standard architectures for image classifiers, without relying on complex architectures or input binarization. Second, our method is able to learn a single MNIST model which is simultaneously robust to all three threat models, whereas previous work

was only robust against two at a time. Finally, our method is easily scalable to datasets beyond MNIST, providing the first CIFAR10 model trained to be simultaneously robust against ℓ_∞, ℓ_2, and ℓ_1 adversaries.

We trained models using both the simple generalizations of adversarial training to multiple adversaries and also using MSD. Since the analysis by synthesis model is not scalable to CIFAR10, we additionally trained CIFAR10 models against individual PGD adversaries to measure the changes and tradeoffs in universal robustness. We evaluated these models with a broad suite of both gradient and non-gradient based attacks using Foolbox[1] (the same attacks used by Schott et al. [2019]), and also incorporated all the PGD-based adversaries discussed in this paper. All aggregate statistics that combine multiple attacks compute the worst case error rate over all attacks for *each* example.

Summaries of these results at specific thresholds can be found in Tables 5.2 and 5.3, where B-ABS and ABS refer to binarized and non-binarized versions of the analysis by synthesis models from Schott et al. [2019], P_p refers to a model trained against a PGD adversary with respect to the p-norm, Worst-PGD and PGD-Aug refer to models trained using the worst-case and data augmentation generalizations of adversarial training, and MSD refers to models trained using multi steepest descent.

5.4.1 Experimental setup

Architectures and hyperparameters For MNIST, we use a four layer convolutional network with two convolutional layers consisting of 32 and 64 5×5 filters and 2 units of padding, followed by a fully connected layer with 1024 hidden units, where both convolutional layers are followed by 2×2 Max Pooling layers and ReLU activations (this is the same architecture used by Madry et al. [2017]). This is in contrast to past work on MNIST, which relied on per-class variational autoencoders to achieve robustness against multiple threat models [Schott et al., 2019], which was also not easily scalable to larger datasets. Since our methods have the same complexity as standard adversarial training, they also easily apply to standard CIFAR10 architectures, and in this paper we use the well known pre-activation version of the ResNet18 architecture consisting of nine residual units with two convolutional layers each [He et al., 2016b]. For all the models, we used the SGD optimizer with momentum 0.9 and weight decay $5 \cdot 10^{-4}$.

MNIST adversary and training parameters We train the models to a maximum of 20 epochs. We used a variation of the learning rate schedule from Smith [2018], which is piecewise linear from 0 to 0.1 over the first 7 epochs, down to 0.001 over the next 8 epochs, and finally back down to 0.0001 in the last 5 epochs. At test time, we increase the number of iterations for the PGD adversaries to $(100, 200, 100)$ for $(\ell_\infty, \ell_2, \ell_1)$.

- The ℓ_∞ adversary used a step size $\alpha = 0.01$ within a radius of $\epsilon = 0.3$ for 50 iterations.
- The ℓ_2 adversary used a step size $\alpha = 0.1$ within a radius of $\epsilon = 1.5$ for 100 iterations.
- The ℓ_1 adversary used a step size of $\alpha = 0.05$ within a radius of $\epsilon = 12$ for 50 iterations. By default the attack is run with two restarts, once starting with $\delta = 0$ and once by randomly

[1] https://github.com/bethgelab/foolbox [Rauber et al., 2017]

initializing δ in the allowable perturbation ball. k_1 = 5, k_2 = 20 as described in 5.3.4.

- The MSD adversary used step sizes of $\alpha = (0.01, 0.2, 0.05)$ for the $(\ell_\infty, \ell_2, \ell_1)$ directions within a radius of $\epsilon = (0.3, 1.5, 12)$ for 100 iterations.

CIFAR10 adversary and training parameters We used a variation of the learning rate schedule from Smith [2018] to achieve superconvergence in 50 epochs, which is piecewise linear from 0 to 0.1 over the first 20 epochs, down to 0.005 over the next 20 epochs, and finally back down to 0 in the last 10 epochs.

- The ℓ_∞ adversary used a step size $\alpha = 0.003$ within a radius of $\epsilon = 0.03$ for 40 iterations.
- The ℓ_2 adversary used a step size $\alpha = 0.05$ within a radius of $\epsilon = 0.5$ for 50 iterations.
- The ℓ_1 adversary used a step size $\alpha = 0.1$ within a radius of $\epsilon = 12$ for 50 iterations. k_1 = 5, k_2 = 20 as described in 5.3.4.
- The MSD adversary used step sizes of $\alpha = (0.003, 0.05, 0.05)$ for the $(\ell_\infty, \ell_2, \ell_1)$ directions within a radius of $\epsilon = (0.03, 0.3, 12)$ for 50 iterations. Note that the MSD model trained for ℓ_2 radius of 0.3 is in fact robust to a higher radius of 0.5.

Attacks used for evaluation To evaluate the model, we incorporate the attacks from Schott et al. [2019] as well as our PGD based adversaries using projected steepest descent, however we provide a short description here. Note that we exclude attacks based on gradient estimation, since the gradient for the standard architectures used here are readily available.

- For ℓ_∞ attacks, although we find the ℓ_∞ PGD adversary to be quite effective, for completeness, we additionally use the Foolbox implementations of Fast Gradient Sign Method [Goodfellow et al., 2015], PGD adversary [Madry et al., 2017], and the Momentum Iterative Method [Dong et al., 2018].
- For ℓ_2 attacks, in addition to the ℓ_2 PGD adversary, we use the Foolbox implementations of the same PGD adversary, the Gaussian noise attack [Rauber et al., 2017], the boundary attack [Brendel et al., 2017], DeepFool [Moosavi-Dezfooli et al., 2016], the pointwise attack [Schott et al., 2019], DDN based attack [Rony et al., 2018], and C&W attack [Carlini and Wagner, 2017b].
- For ℓ_1 attacks, we use both the ℓ_1 PGD adversary as well as additional Foolbox implementations of ℓ_0 attacks at the same radius, namely the salt & pepper attack [Rauber et al., 2017] and the pointwise attack [Schott et al., 2019]. Note that an ℓ_1 adversary with radius ϵ is strictly stronger than an ℓ_0 adversary with the same radius, and so we choose to explicitly defend against ℓ_1 perturbations instead of the ℓ_0 perturbations considered by Schott et al. [2019].

We make **10 random restarts** for each of the evaluation results mentioned hereon for both MNIST and CIFAR10 [2]. We encourage future work in this area to incorporate the same, since the success of all attacks, specially decision based or gradient free ones, is observed to increase significantly over restarts.

[2] All attacks were run on a subset of the first 1000 test examples with 10 random restarts, with the exception of

Table 5.2: Summary of adversarial accuracy results for MNIST

	P_∞	P_2	P_1	B-ABS[3]	ABS[3]	Worst PGD	PGD Aug	MSD
Clean Accuracy	99.1%	99.4%	98.9%	99%	99%	98.9%	99.1%	98.2%
PGD-ℓ_∞	90.3%	0.4%	0.0%	-	-	68.4%	83.7%	63.7%
FGSM	94.9%	68.6%	6.4%	85%	34%	82.4%	90.9%	81.8%
PGD-Foolbox	92.1%	8.5%	0.1%	86%	13%	72.1%	85.7%	67.9%
MIM	92.3%	14.5%	0.1%	85%	17%	73.9%	87.3%	71.0%
ℓ_∞ attacks ($\epsilon = 0.3$)	**90.3**%	0.4%	0.0%	77%	8%	68.4%	83.7%	63.7%
PGD-ℓ_2	83.8%	87.0%	70.8%	-	-	85.3%	87.9%	84.2%
PGD-Foolbox	93.4%	89.7%	74.4%	63%	87%	86.9%	91.5%	86.9%
Gaussian Noise	98.9%	99.6%	98.0%	89%	98%	97.4%	99.0%	97.8%
Boundary Attack	52.6%	92.1%	83.0%	91%	83%	86.9%	79.1%	88.6%
DeepFool	95.1%	92.2%	76.5%	41%	83%	87.9%	93.5%	87.9%
Pointwise Attack	74.3%	97.4%	96.6%	87%	94%	92.7%	89.0%	95.1%
DDN	82.7%	87.0%	70.8%	-	-	85.1%	85.2%	84.3%
CWL2	88.2%	88.1%	75.5%	-	-	85.2%	87.5%	85.1%
ℓ_2 attacks ($\epsilon = 1.5$)	45.3%	**87.0**%	70.3%	39%	80%	82.1%	75.0%	82.6%
PGD-ℓ_1	51.8%	49.9%	71.8%	-	-	66.5%	57.4%	64.8%
Salt & Pepper	55.5%	96.3%	95.6%	96%	95%	86.4%	71.9%	92.2%
Pointwise Attack	2.4%	66.4%	85.2%	82%	78%	60.1%	17.1%	72.8%
ℓ_1 attacks ($\epsilon = 12$)	1.4%	43.4%	71.8%	**82**%	78%	54.6%	15.6%	62.3%
All attacks	1.4%	0.4%	0.0%	39%	8%	53.7%	15.6%	**58.7**%

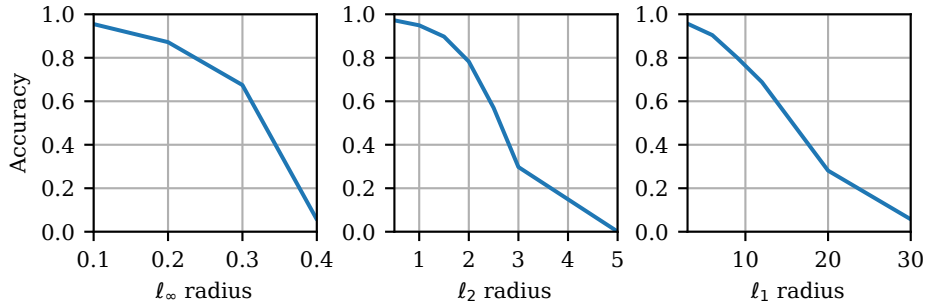


Figure 5.9: Robustness curves showing the adversarial accuracy for the MNIST model trained with MSD against ℓ_∞ (left), ℓ_2 (middle), and ℓ_1 (right) threat models over a range of epsilon.

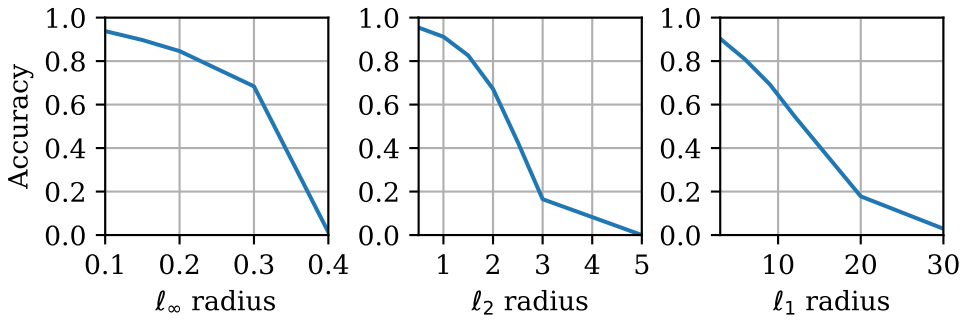


Figure 5.10: Robustness curves showing the adversarial accuracy for the MNIST model trained with the worst case generalization for adversarial training (Worst-PGD) against ℓ_∞ (left), ℓ_2 (middle), and ℓ_1 (right) threat models over a range of epsilon.

5.4.2 Robustness to ℓ_∞, ℓ_2, and ℓ_1 on MNIST

We first present results on the MNIST dataset, which are summarized in Table 5.2. All attacks were run on a subset of the first 1000 test examples with 10 random restarts, with the exception of Boundary Attack, which by default makes 25 trials per iteration, and DDN attack, which does not benefit from restarts owing to a deterministic starting point. Note that the results for B-ABS and ABS models are from Schott et al. [2019], which uses gradient estimation techniques whenever a gradient is needed, and the robustness against all attacks for B-ABS and ABS is an upper bound based on the reported results. Further, these models are not evaluated with restarts, pushing the reported results even higher than actual.

While considered an "easy" dataset, we note that the previous state-of-the-art result for multiple threat models on MNIST (and our primary comparison) is only able to defend against two out of three threat models at a time [Schott et al., 2019] using comparatively complex variational

Boundary Attack, which by default makes 25 trials per iteration and DDN based Attack which does not benefit from the same owing to a deterministic initialization of δ.

[3]Results are from Schott et al. [2019], which used an ℓ_0 threat model of the same radius and evaluated against ℓ_0 attacks. So the reported number here is an upper bound on the ℓ_1 adversarial accuracy. Further, they evaluate their model without restarts and the adversarial accuracy against all attacks is an upper bound based on the reported accuracies for the individual threat models. Finally, all ABS results were computed using numerical gradient estimation, since gradients are not readily available.

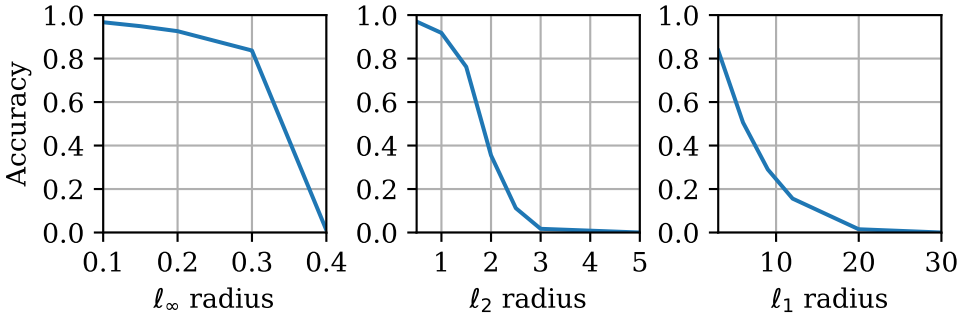


Figure 5.11: Robustness curves showing the adversarial accuracy for the MNIST model trained with the data augmentation generalization for adversarial training (PGD-Aug) against ℓ_∞ (left), ℓ_2 (middle), and ℓ_1 (right) threat models over a range of epsilon.

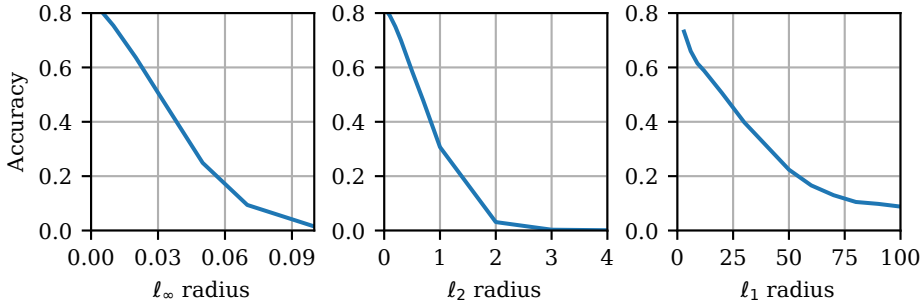


Figure 5.12: Robustness curves showing the adversarial accuracy for the CIFAR10 model trained with MSD against ℓ_∞ (left), ℓ_2 (middle), and ℓ_1 (right) threat models over a range of epsilon.

autoencoder architectures. In contrast, we see that both simple generalizations of adversarial training are able to achieve competitive results on standard models, notably being able to defend against all three threat models simultaneously, while the model trained with MSD performs even better, achieving error rates of 63.7%, 82.6%, and 62.3% for ℓ_∞, ℓ_2, and ℓ_1 perturbations with radius $\epsilon = 0.3$, 1.5, and 12. A complete robustness curve over a range of epsilons for the MSD model over each threat model can be found in Figure 5.9, and robustness curves for the worst case and the data augmentation method are in Figures 5.10 and 5.11.

5.4.3 Robustness to ℓ_∞, ℓ_2, and ℓ_1 on CIFAR10

Next, we present results on the CIFAR10 dataset, which are summarized in Table 5.3. All attacks were run on a subset of the first 1000 test examples with 10 random restarts, with the exception of Boundary Attack, which by default makes 25 trials per iteration, and DDN attack, which does not benefit from restarts owing to a deterministic starting point. Further note that salt & pepper and pointwise attacks in the ℓ_1 section are technically ℓ_0 attacks, but produce perturbations in the ℓ_1 ball.

Our MSD model achieves $(47.6\%, 64.3\%, 53.4\%)$ adversarial accuracy for $(\ell_\infty, \ell_2, \ell_1)$ perturbations of size $\epsilon = (0.03, 0.5, 12)$, reaching an overall adversarial adversarial accuracy of 46.1% over all threat models. Interestingly, note that the P_1 model trained against an ℓ_1 PGD adversary

Table 5.3: Summary of adversarial accuracy results for CIFAR10

	P_∞	P_2	P_1	Worst-PGD	PGD-Aug	MSD
Clean accuracy	83.3%	90.2%	73.3%	81.0%	84.6%	81.7%
PGD-ℓ_∞	50.3%	48.4%	29.8%	44.9%	42.8%	49.8%
FGSM	57.4%	43.4%	12.7%	54.9%	51.9%	55.0%
PGD-Foolbox	52.3%	28.5%	0.6%	48.9%	44.6%	49.8%
MIM	52.7%	30.4%	0.7%	49.9%	46.1%	50.6%
ℓ_∞ attacks ($\epsilon = 0.03$)	50.7%	28.3%	0.2%	44.9%	42.5%	47.6%
PGD-ℓ_2	59.0%	62.1%	28.9%	64.1%	66.9%	66.0%
PGD-Foolbox	61.6%	64.1%	4.9%	65.0%	68.0%	66.4%
Gaussian Noise	82.2%	89.8%	62.3%	81.3%	84.3%	81.8%
Boundary Attack	65.5%	67.9%	2.3%	64.4%	69.2%	67.9%
DeepFool	62.2%	67.3%	0.9%	64.4%	67.4%	65.7%
Pointwise Attack	80.4%	88.6%	46.2%	78.9%	83.8%	81.4%
DDN	60.0%	63.5%	0.1%	64.5%	67.7%	66.2%
CWL2	62.0%	71.6%	0.1%	66.9%	71.5%	68.7%
ℓ_2 attacks ($\epsilon = 0.05$)	57.3%	61.6%	0.0%	61.7%	65.0%	64.3%
PGD-ℓ_1	16.5%	49.2%	69.1%	39.5%	54.0%	53.4%
Salt & Pepper	63.4%	74.2%	35.5%	75.2%	80.7%	75.6%
Pointwise Attack	49.6%	62.4%	8.4%	63.3%	77.0%	72.8%
ℓ_1 attacks ($\epsilon = 12$)	16.0%	46.6%	7.9%	39.4%	54.0%	53.4%
All attacks	15.6%	27.5%	0.0%	34.9%	40.6%	**46.1**%

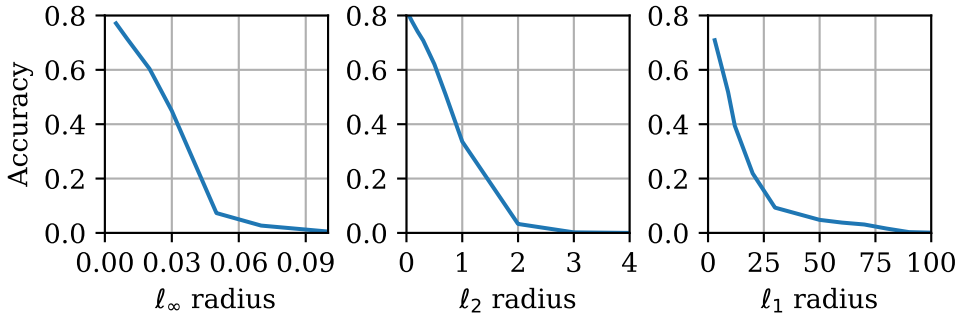


Figure 5.13: Robustness curves showing the adversarial accuracy for the CIFAR10 model trained with the worst case generalization for adversarial training (Worst-PGD) against ℓ_∞ (left), ℓ_2 (middle), and ℓ_1 (right) threat models over a range of epsilon.

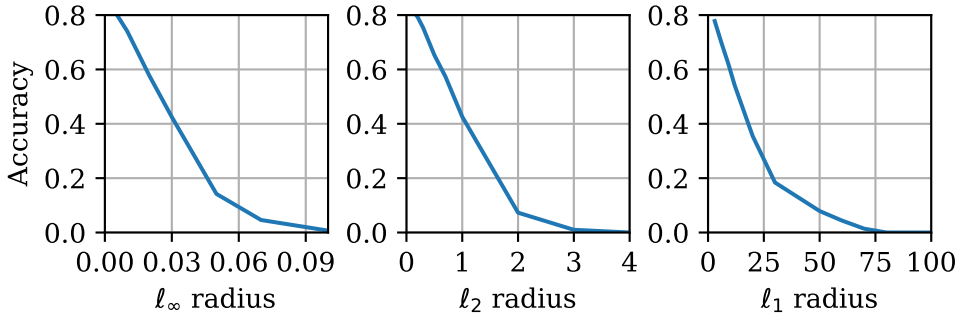


Figure 5.14: Robustness curves showing the adversarial accuracy for the CIFAR10 model trained with the data augmentation generalization for adversarial training (PGD-Aug) against ℓ_∞ (left), ℓ_2 (middle), and ℓ_1 (right) threat models over a range of epsilon.

Table 5.4: Comparison with contemporary work on MNIST (higher is better). Results for all models except MSD are taken as is from Tramèr and Boneh [2019]

	Vanilla	Adv_∞	Adv_1	Adv_2	Adv_{avg}	Adv_{max}	**MSD**
Clean accuracy	99.4%	99.1%	98.9%	98.5%	97.3%	97.2%	98.2%
ℓ_∞ attacks ($\epsilon = 0.3$)	0.0%	91.1%	0.0%	0.4%	76.7%	71.7%	63.7%
ℓ_2 attacks ($\epsilon = 2.0$)	12.4%	12.1%	50.6%	71.8%	58.3%	56.0%	67.4%
ℓ_1 attacks ($\epsilon = 10$)	8.5%	11.3%	78.5%	68.0%	53.9%	62.6%	70.0%
All attacks	0.0%	6.8%	0.0%	0.4%	49.9%	52.4%	**60.9%**

is not very robust when evaluated against other attacks, even though it can defend reasonably well against the ℓ_1 PGD attack in isolation. A complete robustness curve over a range of epsilons for the MSD model over each threat model can be found in Figure 5.12, and robustness curves for the worst case and the data augmentation method are in Figures 5.13 and 5.14.

5.4.4 Comparison with Tramèr and Boneh [2019]

In this section we compare the results of our trained MSD model with that of Tramèr and Boneh [2019], who study the theoretical and empirical trade-offs of adversarial robustness in various settings when defending against multiple adversaries. Training methods presented by them in their comparisons, namely Adv_{avg} and Adv_{max} closely resemble the naive approaches discussed in this paper: PGD-Aug and Worst-PGD respectively. We use the results as is from their work, and additionally compare the position of our MSD models at the revised thresholds used by Tramèr and Boneh [2019] without specially retraining them.

The results of Tables 5.4 and 5.5 show that the relative advantage of MSD over naive techniques does hold up. While we do make a comparison to the most relevant concurrent work for completeness, the following differences can bias the robust accuracies reported for the MSD models to relatively lower than expected (and correspondingly, the robust accuracies reported for the other models are relatively higher than expected):

1. **Use of random restarts:** We observe in our experiments that using up to 10 restarts for all our attacks leads to a decrease in model accuracy from 5 to 10% across all models.

Table 5.5: Comparison with contemporary work on CIFAR10 (higher is better). Results for all models except MSD are taken as is from Tramèr and Boneh [2019]

	Vanilla	Adv_∞	Adv_1	Adv_{avg}	Adv_{max}	**MSD**
Clean accuracy	95.7%	92.0%	90.8%	91.1%	91.2%	82.1%
ℓ_∞ attacks ($\epsilon = \frac{4}{255}$)	0.0%	71.0%	53.4%	64.1%	65.7%	65.6%
ℓ_1 attacks ($\epsilon = \frac{2000}{255}$)	0.0%	16.4%	66.2%	60.8%	62.5%	62.0%
All attacks	0.0%	16.4%	53.1%	59.4%	61.1%	**61.7%**

Tramèr and Boneh do not mention restarting their attacks for these models and so the results for models apart from MSD in Tables 5.4, 5.5 could potentially be lowered with random restarts.

2. **Different training and testing thresholds:** The MSD model for the MNIST dataset was trained at ϵ = (0.3, 1.5, 12) for the ℓ_∞, ℓ_2, ℓ_1 perturbation balls respectively, while Tramèr and Boneh [2019] tested at ϵ = (0.3, 2.0, 10). This may lower the robust accuracy at these thresholds for the MSD model, since it was not trained for that particular threshold. Likewise, the MSD model for CIFAR10 was also trained at ϵ = (0.03, 0.05, 12) for the ℓ_∞, ℓ_2, ℓ_1 perturbation balls respectively, while Tramèr and Boneh [2019] tested at ϵ = ($\frac{4}{255}$, 0, $\frac{2000}{255}$).
3. **Different perturbation models:** For the CIFAR10 results in Table 5.5, Adv_{avg} & Adv_{max} models are trained and tested only for ℓ_1 and ℓ_∞ adversarial perturbations, whereas the MSD model is robust to the union of ℓ_1, ℓ_2 and ℓ_∞, achieving a much harder task.
4. **Larger Suite of Attacks Used:** The attacks used by Tramèr and Boneh are PGD, EAD [Chen et al., 2017] and Pointwise Attack [Schott et al., 2019] for ℓ_1; PGD, C&W [Carlini and Wagner, 2017b] and Boundary Attack [Brendel et al., 2017] for ℓ_2; and PGD for ℓ_∞ adversaries. We use a more expansive suite of attacks as seen in Table 5.3. Some of the attacks like DDN, which proved to be strong adversaries in most cases, were not considered by Tramèr and Boneh [2019] and thus were only used to attack the MSD models in Tables 5.4 and 5.5.

5.5 Discussion

In this chapter, we have expanded the possible threat models for adversarial examples in two main directions. First, we proposed a general threat model for adversarial examples based on the Wasserstein distance, a metric that captures a kind of perturbation that is fundamentally different from traditional ℓ_p perturbations. To generate these examples, we derived an algorithm for fast, approximate projection onto the Wasserstein ball that can use local transport plans for even more speedup on images. We successfully attacked standard networks, showing that these adversarial examples are structurally perturbed according to the content of the image, and demonstrated the empirical effectiveness of adversarial training. Finally, we observed that networks trained to be provably robust against ℓ_∞ attacks are more robust than standard networks against Wasserstein attacks, however we show that the current state of provable defenses is insufficient to directly

apply to the Wasserstein ball due to their reliance on interval bounds. This is a key roadblock to the development of provable defenses against not just Wasserstein attacks, but also to improve the robustness of classifiers to other attacks that do not naturally convert to interval bounds (e.g. ℓ_0 or ℓ_1 attacks).

Second, we showed how to generalize adversarial training to a union of multiple perturbation models, and demonstrated that it can perform quite effectively. While simple generalizations of adversarial training can work to some degree, we improve upon this with multi steepest descent, which incorporates the different perturbation models directly into the direction of steepest descent. MSD based adversarial training procedure is able to outperform past approaches, demonstrating that adversarial training can in fact learn networks that are robust to multiple perturbation models (as long as they are included in the threat model) while being scalable beyond MNIST and using standard architectures. While having meaningful, individual threat models is certainly important, this work makes progress towards the ultimate goal: a classifier which is truly robust to all kinds of adversarial attacks.

Chapter 6

Conclusion

In this dissertation, we studied the defenses against adversarial examples using the perspective of robust optimization, tackling a diverse set of problems. In the process, we presented a number of new algorithms while revisiting old algorithms, developing efficient methods and discovering new insights with major implications for the field of adversarial robustness.

In Chapter 3 we presented a framework for developing duality based certificates which can be optimized to guarantee the robustness of a deep network to adversarial examples. The approach was motivated by dual feasible solutions of linear programming relaxations, which enable computationally fast bounds that brought provable defenses first to convolutional networks, then to modern deep learning architectures with convolutions and residual connections. Efficiency of the approach was maintained by leveraging Cauchy random projections to estimate computationally expensive quantities. By virtue of being a provable defense, the approach is unbreakable and not prone to overestimated measures of robustness unlike countless past approaches, and is ready for application to small to medium scale problems where robustness is critical, capable of trading off certified robustness and clean accuracy with model cascades.

Despite being so widely used, the more empirical defense of adversarial training is still not fully understood. In Chapter 4 we overturn the longstanding presumption that single step FGSM adversarial training cannot learn a robust classifier, correcting for previous approaches and identifying a catastrophic overfitting behavior which explains why past approaches failed. We also identify a more general robust overfitting phenomenon which occurs throughout all adversarial training approaches across different settings, where unlike in the standard setting, overfitting can drastically impair the generalization performance of an adversarially robust network. The implications of this finding are quite dire for the field, as all algorithmic improvements over PGD adversarial training can simply be matched by early stopping vanilla PGD adversarial training, and so no algorithmic progress has been made since PGD adversarial training. Both of these properties of robust training have further implications on the difficulty and runtime of robust training: by using weaker adversaries with less steps and early stopping the training process, one can learn adversarially robust deep networks orders of magnitude faster than before, and on par with training times for the standard setting.

Since measuring progress in adversarial examples for both empirical and provable defenses rely so heavily on having well-defined threat models, in Chapter 5 we further proposed two additional threat models to expand the scope of adversarial robustness to more settings. We

established the Wasserstein threat model to capture inherent structure within images as an alternative to completely unstructured ℓ_p noise, and developed efficient methods for projecting onto the Wasserstein ball, generating Wasserstein adversarial examples, and ultimately applying an adversarial training defense on the Wasserstein threat model. We additionally explored methods for learning models which are adversarially robust to the union of multiple threat models as a step towards developing models which are robust in more human-like ways, improving benchmarks in robustness to unions of ℓ_p threat models.

Although we focus on adversarial defenses in this dissertation, the problems encountered and the methods used spanned a wide range of topics such as robust optimization, convex relaxations, linear programming, duality, optimal transport, generalization, and overfitting in deep learning. Through this wide variety of problems, the work in this dissertation has advanced the state of both provable and empirical defenses for robust deep learning.

6.1 Open problems

We conclude this thesis by posing a number of open questions for the field of adversarial robustness. Answering these questions will progress our understanding of deep learning, allowing us to more concretely characterize what has historically been a complex black box. Solving some of these problems is a necessary prerequisite before deep learning can be safely or even legally deployed in higher-stakes scenarios that require robust performance.

6.1.1 Adversarial training, provable defenses, and generalization

Although work has looked at studying and explaining the generalization gap of robust training [Raghunathan et al., 2019, Schmidt et al., 2018], there is a significant gap between the empirical robustness of adversarial training and certified robustness. Is this is a theoretical limitation of the provable defenses based on the relaxations being used, or can this gap can be closed further?

Furthermore, provable defenses produce vacuous results on models that weren't trained to minimize their specific bound (including adversarially trained networks). Is it possible to produce a non-vacuous bound for adversarially trained networks? This is currently only possible for small, fully connected networks with SDP verifiers [Raghunathan et al., 2018b]. Does a provable defense which can certify other models exist?

Although adversarial training can achieve 0% robust error on the training set, current approaches in provable defenses are unable to do so. How can we improve the effective hypothesis complexity of provable defenses to better (robustly) fit the training set, while still maintaining reasonable bounds? Recent results in universal approximation of even the extremely loose interval bound propagation approach suggests that this is theoretically possible, albeit potentially extremely costly [Baader et al., 2019].

The generalization gap for robust metrics (e.g. adversarial accuracy) is significantly worse than the generalization gap for standard networks. This goes contrary to what one would expect: training a network to ignore inhuman, imperceptible perturbations should only push the network towards more human-recognizable features, which ought to be more generalizable here. What is the limitation here? Is the dataset simply insufficient and doesn't contain features which are

robust and generalize well? Or do the more human-recognizable features exist in the dataset but simply don't generalize?

6.1.2 Real world attacks, threat models, and specifications

Most real attacks on machine learning systems in the wild generally don't follow a clean mathematical formulation present in the imperceptible, ℓ_p setting. How can we bridge this gap and create meaningful threat models which reflect perturbations that show up in the wild that we want our systems to be robust to?

As machine learning models get used in more applications, what kind of specifications can we give with a model to convince a customer that the model does what they want? Will consumers care about imperceptible ℓ_p-norm robustness, given how effective adversarial attacks can be in the real world while not being captured by small ℓ_p perturbations? What sorts of properties and guarantees do we want our systems to have beyond robustness to imperceptible noise, and how can we measure them confidently?

6.1.3 Adversarial robustness as a way to encode priors into deep networks

Although adversarial examples were originally motivated as imperceptible noise which break deep networks, the min-max formulation and training approaches can more generally be seen as a way to encode and certify invariants into our models. Although for the adversarial examples setting this invariance is output stability over ℓ_p regions around each example, the notion of encoding invariants into our models could be leveraged to tackle problems that go far beyond adversarial robustness (e.g. controller properties, Lyapunov stability, or changes under distribution shift).

Bibliography

Hadi Abdullah, Washington Garcia, Christian Peeters, Patrick Traynor, Kevin RB Butler, and Joseph Wilson. Practical hidden voice attacks against speech and speaker recognition systems. *arXiv preprint arXiv:1904.05734*, 2019. 2.1.2

Jean-Baptiste Alayrac, Jonathan Uesato, Po-Sen Huang, Alhussein Fawzi, Robert Stanforth, and Pushmeet Kohli. Are labels required for improving adversarial robustness? In *Advances in Neural Information Processing Systems*, pages 12192–12202, 2019. 2.2.3, 4, 4.4.3

Alnur Ali, J Zico Kolter, and Ryan J Tibshirani. A continuous-time view of early stopping for least squares regression. *arXiv preprint arXiv:1810.10082*, 2018. 4

Jason Altschuler, Jonathan Weed, and Philippe Rigollet. Near-linear time approximation algorithms for optimal transport via sinkhorn iteration. In *Advances in Neural Information Processing Systems*, pages 1964–1974, 2017. 5

Moustafa Alzantot, Yash Sharma, Ahmed Elgohary, Bo-Jhang Ho, Mani Srivastava, and Kai-Wei Chang. Generating natural language adversarial examples. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2890–2896, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1316. URL https://www.aclweb.org/anthology/D18-1316. 2.1.1, 5

Maksym Andriushchenko, Francesco Croce, Nicolas Flammarion, and Matthias Hein. Square attack: a query-efficient black-box adversarial attack via random search. *arXiv preprint arXiv:1912.00049*, 2019. 2.1.2

Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra, and Jorge Luis Reyes-Ortiz. A public domain dataset for human activity recognition using smartphones. In *ESANN*, 2013. 3.3.3

Anish Athalye and Ilya Sutskever. Synthesizing robust adversarial examples. *arXiv preprint arXiv:1707.07397*, 2017. 2.1.2, 2.2.4

Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018*, July 2018a. URL https://arxiv.org/abs/1802.00420. 1.1.1, 2.1.2, 2.2.4, 3.5.1

Anish Athalye, Logan Engstrom, Andrew Ilyas, and Kevin Kwok. Synthesizing robust adversarial examples. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning*

Research, pages 284–293, Stockholmsmssan, Stockholm Sweden, 10–15 Jul 2018b. PMLR. URL http://proceedings.mlr.press/v80/athalye18b.html. 2.1.1, 5

Maximilian Baader, Matthew Mirman, and Martin Vechev. Universal approximation with certified networks. *arXiv preprint arXiv:1909.13846*, 2019. 6.1.1

Mislav Balunovic, Maximilian Baader, Gagandeep Singh, Timon Gehr, and Martin Vechev. Certifying geometric robustness of neural networks. In *Advances in Neural Information Processing Systems*, pages 15287–15297, 2019. 2.2.1

Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, 2019. 4, 4.3.10

Aharon Ben-Tal, Laurent El Ghaoui, and Arkadi Nemirovski. *Robust optimization*. Princeton University Press, 2009. 2.2

Avrim Blum, Travis Dick, Naren Manoj, and Hongyang Zhang. Random smoothing might be unable to certify ℓ_∞ robustness for high-dimensional images. *arXiv preprint arXiv:2002.03517*, 2020. 2.2.2

Wieland Brendel, Jonas Rauber, and Matthias Bethge. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. *arXiv preprint arXiv:1712.04248*, 2017. 2.1.2, 5.4.1, 4

Tom Brown, Dandelion Mane, Aurko Roy, Martin Abadi, and Justin Gilmer. Adversarial patch. 2017. 2.1.1

Jacob Buckman, Aurko Roy, Colin Raffel, and Ian Goodfellow. Thermometer encoding: One hot way to resist adversarial examples. 2018. 1.1.1, 2.2.4

Nicholas Carlini. Is ami (attacks meet interpretability) robust to adversarial examples? *arXiv preprint arXiv:1902.02322*, 2019. 2.2.4

Nicholas Carlini and David Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 3–14. ACM, 2017a. 2.2.4

Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *Security and Privacy (SP), 2017 IEEE Symposium on*, pages 39–57. IEEE, 2017b. 1.1.1, 2.1.2, 2.2.1, 2.2.4, 3, 5.4.1, 4

Nicholas Carlini and David Wagner. Audio adversarial examples: Targeted attacks on speech-to-text. In *2018 IEEE Security and Privacy Workshops (SPW)*, pages 1–7. IEEE, 2018. 2.1.1

Nicholas Carlini, Guy Katz, Clark Barrett, and David L Dill. Ground-truth adversarial examples. *arXiv preprint arXiv:1709.10207*, 2017. 1.1.1, 2.2.1, 3

Nicholas Carlini, Anish Athalye, Nicolas Papernot, Wieland Brendel, Jonas Rauber, Dimitris Tsipras, Ian Goodfellow, and Aleksander Madry. On evaluating adversarial robustness. *arXiv preprint arXiv:1902.06705*, 2019. 2.1.2, 2.2.4

Yair Carmon, Aditi Raghunathan, Ludwig Schmidt, Percy Liang, and John C Duchi. Unlabeled data improves adversarial robustness. *arXiv preprint arXiv:1905.13736*, 2019. 2.2.3, 4, 5,

4.4.3

Pin-Yu Chen, Yash Sharma, Huan Zhang, Jinfeng Yi, and Cho-Jui Hsieh. Ead: Elastic-net attacks to deep neural networks via adversarial examples, 2017. 4

Chih-Hong Cheng, Georg Nührenberg, and Harald Ruess. Maximum resilience of artificial neural networks. In *International Symposium on Automated Technology for Verification and Analysis*, pages 251–268. Springer, 2017. 1.1.1, 2.2.1, 3

Moustapha Cisse, Piotr Bojanowski, Edouard Grave, Yann Dauphin, and Nicolas Usunier. Parseval networks: Improving robustness to adversarial examples. In *International Conference on Machine Learning*, pages 854–863, 2017. 2.2.2, 3, 3.2.3

Jeremy M Cohen, Elan Rosenfeld, and J Zico Kolter. Certified adversarial robustness via randomized smoothing. *arXiv preprint arXiv:1902.02918*, 2019. 2.2.2

Cody Coleman, Deepak Narayanan, Daniel Kang, Tian Zhao, Jian Zhang, Luigi Nardi, Peter Bailis, Kunle Olukotun, Chris Ré, and Matei Zaharia. Dawnbench: An end-to-end deep learning benchmark and competition. *Training*, 100(101):102, 2017. 4, 4.1

Francesco Croce and Matthias Hein. Minimally distorted adversarial examples with a fast adaptive boundary attack. *arXiv preprint arXiv:1907.02044*, 2019a. 2.1.2

Francesco Croce and Matthias Hein. Provable robustness against all adversarial l_p-perturbations for $p \geq 1$. *CoRR*, abs/1905.11213, 2019b. URL http://arxiv.org/abs/1905.11213. 5

Francesco Croce and Matthias Hein. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. *arXiv preprint arXiv:2003.01690*, 2020. 2.1.2, 2.2.3

Francesco Croce, Maksym Andriushchenko, and Matthias Hein. Provable robustness of relu networks via maximization of linear regions. *CoRR*, abs/1810.07481, 2018. URL http://arxiv.org/abs/1810.07481. 2.2.2

Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 2292–2300. Curran Associates, Inc., 2013. URL http://papers.nips.cc/paper/4927-sinkhorn-distances-lightspeed-computation-of-optimal-transport.pdf. 1.1.3, 5, 5.1.2, 5.1.4

John M Danskin. The theory of max-min, with applications. *SIAM Journal on Applied Mathematics*, 14(4):641–664, 1966. 2.2

Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017. 4, 4.4.2

Yinpeng Dong, Fangzhou Liao, Tianyu Pang, Hang Su, Jun Zhu, Xiaolin Hu, and Jianguo Li. Boosting adversarial attacks with momentum. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. 2.1.2, 2.2.3, 5.4.1

Tianyu Du, Shouling Ji, Jinfeng Li, Qinchen Gu, Ting Wang, and Raheem Beyah. Sirenattack: Generating adversarial audio for end-to-end acoustic systems. *arXiv preprint arXiv:1901.07846*, 2019. 2.1.1

John Duchi, Shai Shalev-Shwartz, Yoram Singer, and Tushar Chandra. Efficient projections onto the l1-ball for learning in high dimensions. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 272–279, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-205-4. doi: 10.1145/1390156.1390191. URL http://doi.acm.org/10.1145/1390156.1390191. 5.3.3

Krishnamurthy Dvijotham, Robert Stanforth, Sven Gowal, Timothy Mann, and Pushmeet Kohli. A dual approach to scalable verification of deep networks. In *Proceedings of the Thirty-Fourth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-18)*, pages 162–171, Corvallis, Oregon, 2018a. AUAI Press. 5.1.6

Krishnamurthy Dvijotham, Robert Stanforth, Sven Gowal, Timothy Mann, and Pushmeet Kohli. A dual approach to scalable verification of deep networks. *arXiv preprint arXiv:1803.06567*, 2018b. 2.2.1, 3.4.1, 3.4.2, 3.5.1

Ruediger Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In *International Symposium on Automated Technology for Verification and Analysis*, 2017. 1.1.1, 2.2.1, 3, 3.1.1, 3.4.2

Logan Engstrom, Brandon Tran, Dimitris Tsipras, Ludwig Schmidt, and Aleksander Madry. A rotation and a translation suffice: Fooling cnns with simple transformations. *arXiv preprint arXiv:1712.02779*, 2017. 2.1.1, 2.2.3, 5

Logan Engstrom, Andrew Ilyas, Shibani Santurkar, and Dimitris Tsipras. Robustness (python library), 2019. URL https://github.com/MadryLab/robustness. 4.3.4, 4.3.4, 4.3.6, 4.3.9

Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust physical-world attacks on deep learning visual classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1625–1634, 2018. 2.1.1, 5

Mahyar Fazlyab, Manfred Morari, and George J Pappas. Safety verification and robustness analysis of neural networks via quadratic constraints and semidefinite programming. *arXiv preprint arXiv:1903.01287*, 2019. 2.2.1

Reuben Feinman, Ryan R Curtin, Saurabh Shintre, and Andrew B Gardner. Detecting adversarial samples from artifacts. *arXiv preprint arXiv:1703.00410*, 2017. 2.2.4

Werner Fenchel. On conjugate convex functions. *Canad. J. Math*, 1(73-77), 1949. 3.4.1

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001. 4

Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. AI^2: Safety and robustness certification of neural networks with abstract interpretation. In *IEEE Conference on Security and Privacy*, 2018. 2.2.1

Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015. URL http://arxiv.org/abs/1412.6572. 2.1, 2.1.1, 2.1.2, 2.2, 2.2.3, 3.3, 4, 5.4.1

Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan

Uesato, Relja Arandjelovic, Timothy A. Mann, and Pushmeet Kohli. On the effectiveness of interval bound propagation for training verifiably robust models. *CoRR*, abs/1810.12715, 2018. URL http://arxiv.org/abs/1810.12715. 2.2.2

Chuan Guo, Mayank Rana, Moustapha Cisse, and Laurens Van Der Maaten. Countering adversarial images using input transformations. *arXiv preprint arXiv:1711.00117*, 2017. 1.1.1, 2.2.4

Chuan Guo, Jacob R Gardner, Yurong You, Andrew Gordon Wilson, and Kilian Q Weinberger. Simple black-box adversarial attacks. *arXiv preprint arXiv:1905.07121*, 2019. 2.1.2

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016a. 4.2, 5.2

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016b. 4.4, 5.4.1

Matthias Hein and Maksym Andriushchenko. Formal guarantees on the robustness of a classifier against adversarial manipulation. In *Advances in Neural Information Processing Systems*. 2017. 2.2.2

Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. *arXiv preprint arXiv:1903.12261*, 2019. 2.1.1

Dan Hendrycks, Kevin Zhao, Steven Basart, Jacob Steinhardt, and Dawn Song. Natural adversarial examples. *arXiv preprint arXiv:1907.07174*, 2019. 2.1.1

Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks. In *International Conference on Computer Aided Verification*, pages 3–29. Springer, 2017. 1.1.1, 2.2.1, 3

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015. 3

Robin Jia, Aditi Raghunathan, Kerem Göksel, and Percy Liang. Certified robustness to adversarial word substitutions. *arXiv preprint arXiv:1909.00986*, 2019. 5

Daniel Kang, Yi Sun, Tom Brown, Dan Hendrycks, and Jacob Steinhardt. Transfer of adversarial robustness between perturbation types. *arXiv preprint arXiv:1905.01034*, 2019. 1.1.3, 5

Guy Katz, Clark Barrett, David Dill, Kyle Julian, and Mykel Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. *arXiv preprint arXiv:1702.01135*, 2017. 1.1.1, 2.2.1, 3, 3.1.1

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015. 3.2, 3.3.1

Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009. 3.5

Anders Krogh and John A Hertz. A simple weight decay can improve generalization. In *Advances in neural information processing systems*, pages 950–957, 1992. 4

Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. In

International Conference on Learning Representations, 2017a. 2.1.2

Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *ICLR Workshop*, 2017b. URL https://arxiv.org/abs/1607.02533. 2.1.2

Alexey Kurakin, Ian Goodfellow, Samy Bengio, Yinpeng Dong, Fangzhou Liao, Ming Liang, Tianyu Pang, Jun Zhu, Xiaolin Hu, Cihang Xie, et al. Adversarial attacks and defences competition. *arXiv preprint arXiv:1804.00097*, 2018. 2.2.4

Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998. 2.2.1

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 3.5

Mathias Lecuyer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, and Suman Jana. Certified robustness to adversarial examples with differential privacy. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 656–672. IEEE, 2019. 2.2.2

Juncheng B Li, Frank R Schmidt, and J Zico Kolter. Adversarial camera stickers: A physical camera attack on deep learning classifier. *arXiv preprint arXiv:1904.00759*, 2019a. 2.1.2

Ping Li, Trevor J Hastie, and Kenneth W Church. Nonlinear estimators and tail bounds for dimension reduction in l1 using cauchy random projections. *Journal of Machine Learning Research*, 8(Oct):2497–2532, 2007. 3.4.5, 3.4.6, 3.5.2, 3.5.2

Yandong Li, Lijun Li, Liqiang Wang, Tong Zhang, and Boqing Gong. Nattack: Learning the distributions of adversarial examples for an improved black-box attack on deep neural networks. *arXiv preprint arXiv:1905.00441*, 2019b. 2.1.2

Alessio Lomuscio and Lalit Maganti. An approach to reachability analysis for feed-forward relu neural networks. *arXiv preprint arXiv:1706.07351*, 2017. 1.1.1, 2.2.1, 3, 3.1.1

Jiajun Lu, Hussein Sibai, Evan Fabry, and David Forsyth. No need to worry about adversarial examples in object detection in autonomous vehicles. *arXiv preprint arXiv:1707.03501*, 2017. 2.2.4

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017. 1.1.2, 2.1.2, 2.2, 2.2.3, 3.3, 3.5.1, 4.1, 4.1, 3, 4.2.1, 4.6, 4.3, 4.3.8, 11, 4.4, 5, 5.4.1, 5.4.1

Pratyush Maini, Eric Wong, and J Zico Kolter. Adversarial robustness against the union of multiple perturbation models. *arXiv preprint arXiv:1909.04068*, 2019. 2.1.2, 2.2.3

Jan Hendrik Metzen, Tim Genewein, Volker Fischer, and Bastian Bischoff. On detecting adversarial perturbations. In *International Conference on Learning Representations*, 2017. 2.2.4

Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. Mixed precision training. *arXiv preprint arXiv:1710.03740*, 2017. 4, 4.1.5

Matthew Mirman, Timon Gehr, and Martin Vechev. Differentiable abstract interpretation for provably robust neural networks. In *International Conference on Machine Learning*, pages

3575–3583, 2018. 2.2.2

Matthew Mirman, Timon Gehr, and Martin Vechev. Robustness certification of generative models. *arXiv preprint arXiv:2004.14756*, 2020. 2.2.1

Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2574–2582, 2016. 5.4.1

Nelson Morgan and Hervé Bourlard. Generalization and parameter estimation in feedforward nets: Some experiments. In *Advances in neural information processing systems*, pages 630–637, 1990. 4

Marius Mosbach, Maksym Andriushchenko, Thomas Trost, Matthias Hein, and Dietrich Klakow. Logit pairing methods can fool gradient-based attacks. *arXiv preprint arXiv:1810.12042*, 2018. 2.2.3, 4

Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. *arXiv preprint arXiv:1912.02292*, 2019. 4, 4.3.10

Nina Narodytska and Shiva Prasad Kasiviswanathan. Simple black-box adversarial perturbations for deep networks. *arXiv preprint arXiv:1612.06299*, 2016. 2.1.2

Behnam Neyshabur, Srinadh Bhojanapalli, David McAllester, and Nati Srebro. Exploring generalization in deep learning. In *Advances in Neural Information Processing Systems*, pages 5947–5956, 2017. 4

Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *2016 IEEE European symposium on security and privacy (EuroS&P)*, pages 372–387. IEEE, 2016a. 2.1.2

Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *Security and Privacy (SP), 2016 IEEE Symposium on*, pages 582–597. IEEE, 2016b. 1.1.1, 2.2.4

Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against deep learning systems using adversarial examples. In *Proceedings of the 2017 ACM Asia Conference on Computer and Communications Security*, 2017. 2.1.2

Jonathan Peck, Joris Roels, Bart Goossens, and Yvan Saeys. Lower bounds on the robustness to adversarial perturbations. In *Advances in Neural Information Processing Systems*, pages 804–813. 2017. 2.2.2, 3

Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Certified defenses against adversarial examples. In *International Conference on Learning Representations*, 2018a. 1.1.1, 2.2.2, 3, 3.2, 3.3.1

Aditi Raghunathan, Jacob Steinhardt, and Percy S Liang. Semidefinite relaxations for certifying robustness to adversarial examples. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 10900–10910. Cur-

ran Associates, Inc., 2018b. URL http://papers.nips.cc/paper/8285-semidefinite-relaxations-for-certifying-robustness-to-adversarial-ex pdf. 1.1.1, 2.2.1, 6.1.1

Aditi Raghunathan, Sang Michael Xie, Fanny Yang, John C Duchi, and Percy Liang. Adversarial training can hurt generalization. *arXiv preprint arXiv:1906.06032*, 2019. 2.2.3, 6.1.1

Jonas Rauber, Wieland Brendel, and Matthias Bethge. Foolbox: A python toolbox to benchmark the robustness of machine learning models. *arXiv preprint arXiv:1707.04131*, 2017. 1, 5.4.1

Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do cifar-10 classifiers generalize to cifar-10? *arXiv preprint arXiv:1806.00451*, 2018. 4.3.9

Leslie Rice, Eric Wong, and J Zico Kolter. Overfitting in adversarially robust deep learning. *arXiv preprint arXiv:2002.11569*, 2020. 2.2.3

Jérôme Rony, Luiz G. Hafemann, Luiz S. Oliveira, Ismail Ben Ayed, Robert Sabourin, and Eric Granger. Decoupling direction and norm for efficient gradient-based L2 adversarial attacks and defenses. *CoRR*, abs/1811.09600, 2018. URL http://arxiv.org/abs/1811.09600. 5.4.1

Hadi Salman, Jerry Li, Ilya Razenshteyn, Pengchuan Zhang, Huan Zhang, Sebastien Bubeck, and Greg Yang. Provably robust deep learning via adversarially trained smoothed classifiers. In *Advances in Neural Information Processing Systems*, pages 11289–11300, 2019a. 2.2.2

Hadi Salman, Greg Yang, Huan Zhang, Cho-Jui Hsieh, and Pengchuan Zhang. A convex relaxation barrier to tight robustness verification of neural networks. In *Advances in Neural Information Processing Systems*, pages 9832–9842, 2019b. 2.2.2

Hadi Salman, Mingjie Sun, Greg Yang, Ashish Kapoor, and J Zico Kolter. Black-box smoothing: A provable defense for pretrained classifiers. *arXiv preprint arXiv:2003.01908*, 2020. 2.2.2

Ludwig Schmidt, Shibani Santurkar, Dimitris Tsipras, Kunal Talwar, and Aleksander Madry. Adversarially robust generalization requires more data. In *Advances in Neural Information Processing Systems*, pages 5014–5026, 2018. 1.1.2, 2.2.3, 4.4.3, 6.1.1

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*, 2019. URL https://openreview.net/forum?id=S1EHOsC9tX. 2.1.2, 5, 5.3.3, 5.4, 5.4.1, 5.4.1, 5.4.2, 3, 4

Ali Shafahi, Mahyar Najibi, Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S Davis, Gavin Taylor, and Tom Goldstein. Adversarial training for free! *arXiv preprint arXiv:1904.12843*, 2019. 2.2.3, 4.1, 4.1, 2, 4.2, 4.6, 4.2.3

Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K Reiter. Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1528–1540. ACM, 2016. 2.1.1, 2.1.2, 5

Mahmood Sharif, Lujo Bauer, and Michael K Reiter. On the suitability of lp-norms for creating and preventing adversarial examples. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 1605–1613, 2018. 2.1.1

Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. Fast and effective robustness certification. In *Advances in Neural Information Processing Systems*, pages 10802–10813, 2018a. 2.2.1

Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. Boosting robustness certification of neural networks. 2018b. 2.2.1

Aman Sinha, Hongseok Namkoong, and John Duchi. Certifiable distributional robustness with principled adversarial training. In *International Conference on Learning Representations*, 2018a. 2.2.2

Aman Sinha, Hongseok Namkoong, and John Duchi. Certifying some distributional robustness with principled adversarial training. 2018b. 5

Leslie N Smith. Cyclical learning rates for training neural networks. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 464–472. IEEE, 2017. 4.1.5, 4

Leslie N Smith. A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay. *arXiv preprint arXiv:1803.09820*, 2018. 5.4.1, 5.4.1

Leslie N Smith and Nicholay Topin. Super-convergence: Very fast training of residual networks using large learning rates. 2018. 4, 4.1.5, 4.2.2

Michael Snow et al. Monge's optimal transport distance for image classification. *arXiv preprint arXiv:1612.00181*, 2016. 5

Yang Song, Taesup Kim, Sebastian Nowozin, Stefano Ermon, and Nate Kushman. Pixeldefend: Leveraging generative models to understand and defend against adversarial examples. *arXiv preprint arXiv:1710.10766*, 2017. 1.1.1, 2.2.4

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014. 4

Otto Neall Strand. Theory and methods related to the singular-function expansion and landwebers iteration for integral equations of the first kind. *SIAM Journal on Numerical Analysis*, 11 (4):798–825, 1974. 4

David Stutz, Matthias Hein, and Bernt Schiele. Confidence-calibrated adversarial training and detection: More robust models generalizing beyond the attack used during training. *arXiv preprint arXiv:1910.06259*, 2019. 5

Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation*, 23(5):828–841, 2019. 2.1.2

Arun Suggala, Adarsh Prasad, and Pradeep K Ravikumar. Connecting optimization and regularization paths. In *Advances in Neural Information Processing Systems*, pages 10608–10619, 2018. 4

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, 2014. URL http://arxiv.org/abs/1312.6199. 2.1, 2.1.1, 2.1.2

Guanhong Tao, Shiqing Ma, Yingqi Liu, and Xiangyu Zhang. Attacks meet interpretability: Attribute-steered detection of adversarial samples. In *Advances in Neural Information Processing Systems*, pages 7717–7728, 2018. 2.2.4

Guillaume Tartavel, Gabriel Peyré, and Yann Gousseau. Wasserstein loss for image synthesis and restoration. *SIAM Journal on Imaging Sciences*, 9(4):1726–1755, 2016. 5

Vincent Tjeng and Russ Tedrake. Verifying neural networks with mixed integer programming. *CoRR*, abs/1711.07356, 2017. URL http://arxiv.org/abs/1711.07356. 1.1.1, 2.2.1, 3

Vincent Tjeng, Kai Y Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. 2018. 2.2.1, 4.2.1

Florian Tramèr and Dan Boneh. Adversarial training and robustness for multiple perturbations. *arXiv preprint arXiv:1904.13000*, 2019. (document), 2.2.3, 5, 5.3.1, 5.3.4, 5.4, 5.4.4, 5.5, 1, 2, 4

Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses. *arXiv preprint arXiv:1705.07204*, 2017. (document), 1.1.2, 2.2.3, 4, 4.1.1, 4.1.2, 4.3, 4.1.4, 1, 2, 4.1.4, 1

Florian Tramer, Nicholas Carlini, Wieland Brendel, and Aleksander Madry. On adaptive attacks to adversarial example defenses. *arXiv preprint arXiv:2002.08347*, 2020. 2.2.4

Jonathan Uesato, Brendan O'Donoghue, Pushmeet Kohli, and Aaron van den Oord. Adversarial risk and the dangers of evaluating against weak attacks. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5025–5034, Stockholmsmssan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL http://proceedings.mlr.press/v80/uesato18a.html. 1.1.1, 2.1.2, 2.2.4

Santosh S Vempala. *The random projection method*, volume 65. American Mathematical Soc., 2005. 3.4.5, 3.5.4

Jianyu Wang. Bilateral adversarial training: Towards fast training of more robust models against adversarial attacks. *arXiv preprint arXiv:1811.10716*, 2018. 2.2.3

Yuting Wei, Fanny Yang, and Martin J Wainwright. Early stopping for kernel boosting algorithms: A general analysis with localized complexities. In *Advances in Neural Information Processing Systems*, pages 6065–6075, 2017. 4

Tsui-Wei Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Duane Boning, Inderjit S Dhillon, and Luca Daniel. Towards fast computation of certified robustness for relu networks. *arXiv preprint arXiv:1804.09699*, 2018. 1.1.1, 2.2.2

Eric Wong and J Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. *arXiv preprint arXiv:1711.00851*, 2017. 1.1.1, 1.2, 2.2.2, 3.4.1, 3.4.3, 3.5, 3.5.1, 3.5.2, 5.2

Eric Wong, Frank Schmidt, Jan Hendrik Metzen, and J. Zico Kolter. Scaling provable adversarial defenses. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 8410–8419. Curran Associates, Inc., 2018. URL http://papers.nips.cc/paper/8060-scaling-provable-adversarial-defenses.pdf. 1.2, 2.2.2, 5.2.1, 5.2.2

Eric Wong, Frank R Schmidt, and J Zico Kolter. Wasserstein adversarial examples via projected sinkhorn iterations. *arXiv preprint arXiv:1902.07906*, 2019. 2.1.1, 2.2.3

Eric Wong, Leslie Rice, and J. Zico Kolter. Fast is better than free: Revisiting adversarial training. In *International Conference on Learning Representations*, 2020a. URL https://openreview.net/forum?id=BJx040EFvH. 2.2.3, 4.3.1, 9, 4, 4.3.7, 4.3.7, 4.3.7

Eric Wong, Tim Schneider, Joerg Schmitt, Frank R. Schmidt, and J. Zico Kolter. Neural network virtual sensors for fuel injection quantities with provable performance specifications. In *2020 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2020b. 1.1.1

Chaowei Xiao, Jun-Yan Zhu, Bo Li, Warren He, Mingyan Liu, and Dawn Song. Spatially transformed adversarial examples. *arXiv preprint arXiv:1801.02612*, 2018. 2.1.1, 2.2.3, 5

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017. 3.3.3

Kai Y. Xiao, Vincent Tjeng, Nur Muhammad (Mahi) Shafiullah, and Aleksander Madry. Training for faster adversarial robustness verification via inducing reLU stability. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=BJfIVjAcKm. 2.2.2

Cihang Xie, Yuxin Wu, Laurens van der Maaten, Alan L Yuille, and Kaiming He. Feature denoising for improving adversarial robustness. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 501–509, 2019. 1.1.2, 2.2.3, 4

Huan Xu, Constantine Caramanis, and Shie Mannor. Robustness and regularization of support vector machines. *Journal of Machine Learning Research*, 10(Jul):1485–1510, 2009. 2.2

Greg Yang, Tony Duan, Edward Hu, Hadi Salman, Ilya Razenshteyn, and Jerry Li. Randomized smoothing of all shapes and sizes. *arXiv preprint arXiv:2002.08118*, 2020. 2.2.2

Yuzhe Yang, Guo Zhang, Dina Katabi, and Zhi Xu. Me-net: Towards effective adversarial robustness with matrix estimation. *arXiv preprint arXiv:1905.11971*, 2019. 2.2.3, 4

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016. 3.5, 4.3.10

Runtian Zhai, Tianle Cai, Di He, Chen Dan, Kun He, John Hopcroft, and Liwei Wang. Adversarially robust generalization just requires more unlabeled data. *arXiv preprint arXiv:1906.00555*, 2019. 2.2.3, 4, 4.4.3

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016. 4

Dinghuai Zhang, Tianyuan Zhang, Yiping Lu, Zhanxing Zhu, and Bin Dong. You only propagate once: Painless adversarial training using maximal principle. *arXiv preprint arXiv:1905.00877*, 2019a. 2.2.3

Hongyang Zhang, Yaodong Yu, Jiantao Jiao, Eric P Xing, Laurent El Ghaoui, and Michael I Jordan. Theoretically principled trade-off between robustness and accuracy. *arXiv preprint arXiv:1901.08573*, 2019b. 2.2.3, 4, 4.3.1, 10, 4.3.9

Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017. 4, 4.4.2

Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. In *Advances in neural information processing systems*, pages 4939–4948, 2018. 1.1.1, 2.2.2

Huan Zhang, Hongge Chen, Chaowei Xiao, Bo Li, Duane Boning, and Cho-Jui Hsieh. Towards stable and efficient training of verifiably robust neural networks. *arXiv preprint arXiv:1906.06316*, 2019c. 2.2.2, 4.3.7