Methods for robust training and evaluation of deep neural networks

Leslie Rice

CMU-CS-23-108

May 2023

Computer Science Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
J. Zico Kolter, Chair
Matt Fredrikson
Aditi Raghunathan
Nicholas Carlini (Google DeepMind)

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

This research was sponsored by the National Science Foundation under award number 1522054, Intel, Robert Bosch GMBH under award number 0087016732PCRPO0087023984, and the Defense Advanced Research Projects Agency under award number HR00112020006. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Keywords: adversarial robustness, deep learning

To Armen

Abstract

As machine learning systems are deployed in real-world, safety-critical applications, it becomes increasingly important to ensure these systems are robust and trustworthy. The study of machine learning robustness gained a significant amount of interest upon discovering the brittle nature of deep neural networks. Intrigue and concerns about this behavior have resulted in a significant body of work on adversarial robustness, which studies a model's performance on worst-case perturbed inputs, known as adversarial examples. In the first chapter of this thesis, we present improvements on adversarial training methods for developing empirically robust deep networks. First, we show that with certain modifications, adversarial training using the fast gradient sign method can result in models that are significantly more robust than previously thought possible, while retaining a much lower training cost compared to alternative adversarial training methods. We then discuss our findings on the harmful effects of overfitting that occur during adversarial training, and show that by using validation-based early stopping, the robust test performance of an adversarially trained model can be drastically improved.

An increasing interest in more natural, non-adversarial settings of robustness has led researchers to alternatively measure robustness in terms of a model's average performance on randomly sampled input corruptions, a notion which also underlies standard data augmentation strategies. In the second chapter of this thesis, we generalize the seemingly separate notions of average and worst-case robustness under a unifying framework that allows us to evaluate models on a wide spectrum of robustness levels. For practical use, we introduce a path sampling-based method for accurately approximating this intermediate robustness objective. We use this metric to analyze and compare deep networks in zero-shot and fine-tuned settings to better understand the effects of large-scale pre-training and fine-tuning on robustness. We show that we can also train models to intermediate levels of robustness using this objective, and further explore alternative, more efficient methods for training that bridge the gap between average and worst-case robustness.

Acknowledgments

I would like to begin by thanking my advisor Zico Kolter for these past five years. I have learned so much, and I am exceedingly grateful for the opportunity and research experience. I would additionaly like to thank Matt Fredrikson, Aditi Raghunathan, and Nicholas Carlini for taking the time to serve on my thesis committee, and for providing valuable feedback on this work. I would also like to thank my research collaborators who contributed to this work. Especially, I'd like to thank Eric Wong, who worked with me on the research presented in Chapter 2 of this thesis, and who also acted as a mentor to me during my first few years as a graduate student. I also would like to acknowledge Huan Zhang and Anna Bair for their significant contribution to the work presented in Chapter 3 of this thesis. Additionally, I'd like to thank all of the current and former members of the Locus Lab, whose camaraderie and kindness I've greatly valued. I'd like to also acknowledge Gaurav Manek for managing the lab's GPU cluster for some time, without whom much of this work would not have been possible. Thanks to Ann Stetser and Sara Golembiewski for their administrative assistance to the Locus Lab, and thanks to Deborah Cavlovich and Jenn Landefeld for all their work in making the graduate program run so smoothly. I'd also like to thank my internship hosts, Wan-Yi Lin at the Bosch Center for AI, and Cyrus Rashtchian and Da-Cheng Juan at Google Research, for providing wonderful internship experiences during my time in graduate school.

I'd also like to thank the people who helped me get into graduate school in the first place – thanks to Alison Norman for encouraging me to get involved in research at UT, Robert van de Geijn for being my undergraduate research advisor, and Jianyu Huang for his research mentorship. Importantly, none of this would have been possible without the friends and family who have supported me during this endeavor. Thanks to my parents, who gave me access to a great education that led me to where I am today, and for their unfaltering love and support. Thanks to my brother Matthew and future sister-in-law Clara for their friendship, and for their optimism that helps remind me of the important things in life. Thanks to the Berberian family for their support and encouragement these past several years. Finally, I would like to thank my husband Armen for being there through the highs and lows of graduate school, for moving to California for a summer so I could do an internship at Google, and most importantly for bringing so much happiness to my life.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

As machine learning methods continue to be successful in a wide range of applications, the problem of learning robust machine learning models remains an important area of research. As machine learning becomes more integrated in safety and security focused applications, the robustness of machine learning models to noisy or perturbed inputs becomes a greater concern. The classical definition of robustness in machine learning, known as adversarial robustness, traces its roots back to robust optimization, a field in optimization theory that allows for the study of worst-case uncertainty [6]. In particular, this view of robustness has been studied extensively in the context of image classification using deep neural networks, after it was shown that such networks are quite brittle in the presence of small, worst-case noise [72]. Despite a decade having passed since the discovery of the susceptibility of deep neural networks to adversarial examples, training adversarially robust models still remains quite challenging, typically resulting in an increased training cost compared to standard training and degraded performance on clean (unperturbed) inputs. And while innovations in deep learning research have resulted in vastly improved standard performance on image classification tasks, we have only seen comparatively incremental gains in adversarial robustness, the largest of which have resulted from training with additional data. The challenges of adversarial learning and the seemingly inevitable trade-off between standard and robust performance are some factors motivating alternative, less conservative notions of robustness.

In this thesis, we begin by discussing our contributions to the field of adversarial robustness, including improvements to adversarial training algorithms and analyses of differences between adversarial learning and standard training of deep neural networks. We then discuss our contributions to alternative measures of robustness, proposing a novel robustness objective that enables measuring robustness across a spectrum of stringency levels up to worst-case robustness. We further show how this intermediate robustness metric can be used to evaluate and compare the robustness of models trained in a variety of manners, and show how this metric can be used to train deep networks to fine-grained levels of robustness.

1.1 Contributions

1.1.1 Identifying and mitigating pitfalls of adversarial training

Adversarial training is considered to be the most successful approach for training models to be empirically robust to worst-case input perturbations. These methods typically use some form of gradient-based attack to generate adversarial examples, and then train the model on these examples. In Chapter 2, we highlight our contributions to the field of adversarial robustness, specifically regarding methods for adversarial training. We identify different failure modes that can occur in existing methods for adversarial training, and show how robust performance can be improved by avoiding these pitfalls. In Section 2.2, we revisit the usage of the fast gradient sign method (FGSM) [30] for adversarial training, which generates adversarial examples using a single gradient step. While efficient, this method was largely ignored after appearing to be ineffective at producing a model robust to a multi-step adversary like projected gradient descent (PGD) [47]. We find that, on the contrary, training against an FGSM adversary can result in a fairly robust model against PGD attacks so long as during training, we avoid a failure mode we call "catastrophic overfitting" [79]. We further provide mechanisms to better avoid this pitfall during training without increasing the algorithm's computational complexity. Equipped with a robust and efficient adversarial training algorithm, we show that we can further improve the speed of adversarial training by combining FGSM with standard techniques for fast training of deep neural networks, such as cyclic learning rate schedules and mixed precision arithmetic.

While catastrophic overfitting is specific to FGSM adversarial training, in Section 2.3 we identify a separate form of overfitting, which we call "robust overfitting" that we discover negatively affects all adversarial training methods [63]. While fitting exactly to the training set is common when training standard deep neural networks, typically resulting in improved standard test accuracy, we show that doing so in adversarial training results in significantly worse robust test accuracy. We further show that while most regularization techniques fail to mitigate the harmful effect of robust overfitting, simply early stopping adversarial training can result in a significant boost in robust test performance. In fact, with early stopping, we show that PGD adversarial training remains competitive, or performs better, than supposed algorithmic improvements introduced subsequently for training robust models.

1.1.2 Robustness between the average and worst case

While adversarial robustness remains an important topic of study, it is often criticized for being an over-conservative objective to consider. For example, adversarial training often results in a model that performs worse on unperturbed inputs than a model trained normally using standard empirical risk minimization. Furthermore, the difficulty of computing the worst-case perturbation subject to some constraints becomes increasingly challenging when considering more realistic and ill-defined types of input corruptions. As an alternative to adversarial robustness, researchers sometimes measure the robustness of a model in terms of average performance over more "naturally-occurring" corruptions, e.g. fixed or randomly sampled, as opposed to adversarial corruptions. Data augmentation can be equated to this average-case robustness objective, typically involving training on random input transformations. However, a criticism of average-

case robustness is that it is not robust enough, as this objective largely ignores low probability perturbations that incur a large loss for the model.

Average-case and worst-case robustness objectives have largely been studied separately, over different perturbation sets. In Chapter 3, we introduce a new robustness objective that generalizes these two notions of robustness under the same framework [64]. Our proposed objective allows for interpolating between average-case and worst-case robustness, effectively measuring *intermediate* robustness. We construct this objective via the observation that worst and average case robustness each can be expressed as a (functional) q-norm over perturbation space, and this formulation furthermore allows for intermediate notions of robustness. As computing these proposed robustness objectives involves approximating a high-dimensional integral, we additionally present an approach for accurate estimation using the Markov chain Monte Carlo-based path sampling method [28]. We show that this metric is not only useful for evaluation purposes, but also can be used as a training objective to train models to a specific desired robustness level.

We additionally show the usefulness of the intermediate robustness metric for comparing models trained without any specific form of robust training. Specifically, we evaluate the robustness of CLIP (Contrastive Language-Image Pre-training) [61], a class of foundation model known for its impressive zero-shot performance on a variety of tasks and distribution shifts. To better understand how pre-training on vast, diverse sets of data, and subsequently fine-tuning on a downstream task, affect robustness beyond the average case, we compare the intermediate robustness of the zero-shot and fine-tuned CLIP models to those of the same architecture trained from scratch. Lastly, we more deeply dive into training for intermediate robustness. We discuss techniques for improved training using the intermediate robustness objective estimated via path sampling, and additionally compare to alternative robust training methods to determine the most efficient way of training for intermediate robustness.

Chapter 2

Adversarial robustness

2.1 Background

Neural networks have been shown to be be highly susceptible to *adversarial examples*, which are inputs to the model that have been intentionally perturbed to cause the model to output an incorrect prediction. Adversarial examples are commonly studied in the image classification setting, and are inputs corrupted with additive noise that is typically constrained to be imperceptible to the human eye. Adversarial examples are constructed by finding a bounded perturbation of the input that maximizes the loss of the model, where the loss simply measures how different the model's prediction is from the ground truth label. Szegedy et al. [72] first discovered this brittle nature of neural networks, finding adversarial perturbations to be extremely effective at changing a model's output, even when such perturbations are quite small. Since their existence was discovered, adversarial examples have been studied extensively, due to the security risk they pose when using machine learning in safety-critical scenarios.

More formally, considering a model h parameterized by θ, and loss function ℓ, we can define an adversarial example for some input x with ground truth label y as the perturbed input $x' = x + \delta^*$, where the perturbation δ^* is chosen to maximize the model's loss, such that

$$\delta^* = \arg\max_{\delta \in \Delta(x)} \ell(h_\theta(x + \delta), y),$$

while being constrained to some set Δ. Typically, the set of allowable perturbations Δ is chosen to be an ℓ_p-norm ball with a small enough radius ϵ such that the perturbation is imperceptible to a human observer, i.e.

$$\Delta = \{\delta : ||\delta||_p \leq \epsilon\}.$$

Following the discovery of adversarial examples, Goodfellow et al. [30] subsequently developed a method called the fast gradient sign method (FGSM) for efficiently generating adversarial examples, and proposed training on these examples to improve a model's adversarial robustness, a technique which is known as adversarial training. FGSM generates adversarial examples with a single gradient step to maximize the loss, such that

$$\delta^* = \epsilon \, \text{sign} \nabla_x \ell(h_\theta(x), y).$$

Adversarial training can be more formally described by the following robust optimization problem,

$$\min_{\theta} \sum_{x,y \in D} \max_{\delta \in \Delta} \ell(h_\theta(x+\delta), y),$$

where the inner maximization can be approximated by an adversarial attack such as FGSM. The full adversarial training procedure using FGSM is shown in Algorithm 1.

Algorithm 1 FGSM adversarial training for T epochs, given some radius ϵ, and a dataset of size M for a network h_θ.

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

FGSM was eventually determined to be a relatively weak attack as more effective approximations to the inner maximization problem were developed. FGSM was initially enhanced by adding a randomization step, and referred to as R+FGSM [76]. The basic iterative method then further improved upon the one-step approach by taking multiple, smaller gradient steps, projecting the updated perturbation back to the perturbation set after each step [43]. This iterative adversarial attack was further strengthened by repeating the basic iterative method multiple times with a different random initialization of the perturbation, and was also shown to be particularly effective when used in the adversarial training procedure [47]. These improvements form the basis of what is widely understood today as adversarial training against a projected gradient descent (PGD) adversary, and the resulting method is recognized as an effective approach to learning empirically robust networks. The full adversarial training procedure using PGD is shown in Algorithm 2.

Algorithm 2 PGD adversarial training for T epochs on a dataset of size M for a network h parameterized by θ. The perturbation set is an ℓ_∞ ball with radius ϵ, and we use step size α.

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

Further proposed improvements to both the PGD adversary and the training procedure include incorporating momentum into the adversary [20], leveraging matrix estimation [84], logit pairing [55], and feature denoising [83]. Most notably, Zhang et al. [91] proposed the method TRADES for adversarial training that balances the trade-off between standard and robust errors, and achieves state-of-the-art performance on several benchmarks, minimizing the objective

$$\min_{\theta} \sum_{x,y \in D} \Big[\ell(h_\theta(x), y) + \beta \max_{\delta \in \Delta} \ell(h_\theta(x), h_\theta(x+\delta)) \Big].$$

Because multi-step adversarial training methods are significantly more time consuming than standard training, several works have focused on improving the efficiency of adversarial training by reducing the computational complexity of calculating gradients and reducing the number of attack iterations [67, 90]. While most adversarial training studies consider additive perturbations bounded by some ℓ_p-norm, separate works have also expanded the general PGD adversarial training algorithm to different threat models including image transformations [22, 82], different distance metrics [78], and multiple threat models [48, 75].

Other adversarial defenses that have been proposed were not always successful, such as distillation [13, 60] and detection of adversarial examples [11, 12, 25, 50, 73], which eventually were defeated by stronger attacks. Adversarial examples were also believed to be ineffective in the real world across different viewpoints [46] until proven otherwise [4], and a large number of adversarial defenses were shown to be relying on obfuscated gradients and ultimately rendered ineffective [3], including thermometer encoding [10] and various preprocessing techniques [32, 70].

2.2 Methods for fast adversarial training

While effective, PGD adversarial training comes at a non-trivial computational cost due to the multiple gradient steps required to generate the adversarial examples, often increasing training time by an order of magnitude over standard training. In response to this difficulty, there was a surge in work that tried to to reduce the complexity of generating an adversarial example while retaining the strength of the PGD attack [67, 90]. While these works present reasonable improvements to the runtime of adversarial training, they are still significantly slower than standard training, which has been greatly accelerated due to competitions for optimizing both the speed and cost of training [16].

We address this problem of improving the efficiency of adversarial training by revisiting the original adversarial attack used in adversarial training, namely the single-step, fast gradient sign method. As discussed in Section 2.1, FGSM adversarial training was previously shown to be completely ineffective in producing models robust to a stronger attack such as PGD. Contrary to this prior belief, we find that several slight modifications to the FGSM attack generation can result in significantly improved robustness against a PGD adversary. Specifically, by simply randomly initializing the perturbation before the single gradient step update, and using a larger step size than the typical choice of radius ϵ, the resulting model is significantly more robust to PGD attacks than one trained according to the original FGSM adversarial training method. We find that these modifications help avoid a failure mode of FGSM adversarial training that we

term "catastrophic overfitting", where the model's robust performance against a PGD adversary suddenly drops during training. We further show that this modified version of 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, cyclic learning rates [69] and mixed-precision training [52].

The end result is that, with these approaches, we are able to train (empirically) robust classifiers far faster than in previous work. For example, we can train an image classifier on the CIFAR-10 dataset to 45% robust test accuracy against ℓ_∞-norm bounded perturbations with radius $\epsilon = 8/255$ in just 6 minutes; previous works reported times of 80 hours for the same level attained using the original PGD-based training [47] and 10 hours using free adversarial training [67], one of the proposed faster alternatives to PGD. Similarly, we can train an an ImageNet classifier to 43% top-1 robust test accuracy against ℓ_∞-norm bounded perturbations with radius $\epsilon = 2/255$ (again matching previous results at the time) in 12 hours of training, compared to 50 hours using free adversarial training, according to Shafahi et al. [67]. Both of these times roughly match the comparable time for quickly training a standard non-robust model to reasonable accuracy. We extensively evaluate these FGSM-trained models against strong PGD-based attacks, and show that they obtain robustness close or equal to that originally reported by Madry et al. [47] using slower PGD-based training.

2.2.1 Related work

As discussed in Section 2.1, the fast gradient sign method (FGSM) was the original method for constructing adversarial examples, and is quite efficient due to its use of a single gradient step. This method was used to perturb the inputs to the model before performing backpropagation as an early form of adversarial training, as shown in Algorithm 1. The FGSM attack was enhanced by adding a randomization step, which was referred to as R+FGSM [76], where the perturbation is initialized on the surface of a hypercube with radius $\epsilon/2$, such that $\delta = \frac{\epsilon}{2}\mathcal{N}(0, 1)$, where ϵ is the radius of the norm ball that comprises the allowable perturbations, and then takes a single gradient step of size $\epsilon/2$. However, both FGSM and R+FGSM were shown to be ineffective for training models to be robust to the stronger PGD attack.

Despite the eventual defeat of other adversarial defenses, adversarial training with a PGD adversary remains empirically robust to this day. However, running a strong PGD adversary within an inner loop of training is expensive, and some earlier work in this topic found that taking larger but fewer steps did not always significantly change the resulting robustness of a network [77]. Several prior works have studied how to train for adversarial robustness at a reduced cost compared to projected gradient descent. One such work shows that when performing a multi-step PGD adversary, it is possible to cut out redundant calculations during backpropagation when computing adversarial examples for additional speedup [90].

Alternatively, to combat the increased computational overhead of the PGD defense, some recent work has looked at regressing the k-step PGD adversary to a variation of its single-step FGSM predecessor called "free" adversarial training, which can be computed with little overhead over standard training by using a single backwards pass to simultaneously update both the model weights and also the input perturbation [67]. This method takes FGSM steps with full step sizes $\alpha = \epsilon$ followed by updating the model weights for N iterations on the same mini-batch (also re-

Algorithm 3 Free adversarial training for T epochs, N mini-batch replays, and a dataset of size M for a network h_θ. The threat model is an ℓ_∞-norm ball with radius ϵ.

$\delta = 0$
// Iterate T/N times to account for mini-batch 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
      $\nabla_\delta, \nabla_\theta = \nabla \ell(h_\theta(x_i + \delta), y_i)$
      $\delta = \delta + \epsilon \cdot \text{sign}(\nabla_\delta)$
      $\delta = \max(\min(\delta, \epsilon), -\epsilon)$
      $\theta = \theta - \nabla_\theta$ *// Update model weights with some optimizer, e.g. SGD*
    end for
  end for
end for

ferred to as "mini-batch replays"). A 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 FGSM adversarial training needs two backwards passes to compute gradients separately for the perturbation and the model weights. The algorithm for free adversarial training is summarized in Algorithm 3, where we highlight the fact that the perturbations are not reset between mini-batches. To account for the additional computational cost of mini-batch 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, Shafahi et al. [67] still need to run over 200 epochs in over 10 hours to learn a robust CIFAR-10 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.

There has also been increasing study on speeding up non-adversarial training of deep networks. For example, top performing training methods from the DAWNBench competition [16] are able to train CIFAR-10 and ImageNet architectures to standard benchmark metrics in mere minutes and hours respectively, using only a modest amount of computational resources. Some general techniques such as cyclic learning rates [69] and half-precision computations [52] have been quite successful in the top ranking submissions, and we show these techniques can also be useful for adversarial training.

Our high-performance modifications to FGSM are similar to that of R+FGSM and free adversarial training in that the perturbation is not initialized to zero. However, we ultimately find that FGSM adversarial training with our choice of initialization and step size results in significantly better robustness against a PGD adversary when compared to R+FGSM. While free adversarial training can result in similar robustness levels as our modified FGSM training, it is less amenable to the reduced epoch, cyclic learning rate schedule, likely due to training with mini-batch replays.

Algorithm 4 Uniform R+FGSM adversarial training for T epochs on a dataset of size M for a network h parameterized by θ. The perturbation set is an ℓ_∞ ball with radius ϵ.

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

2.2.2 Revisiting the fast gradient sign method

We find that minor modifications to the fast gradient sign method can result in significant improvements in robustness against a PGD adversary. Those modifications include uniform random perturbation initialization and a step size larger than ϵ. In this section, we discuss our intuition for these modifications, and best practices for implementing these modifications. Algorithm 4 outlines our complete modified FGSM training method, which we call Uniform R+FGSM.

Perturbation initialization Despite being quite similar to FGSM adversarial training, free adversarial training has been shown to be empirically robust against PGD attacks in contrast to the original FGSM adversarial training. 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 example is a reasonable starting point for the next example. As a result, we hypothesize that the main benefit comes from starting from a non-zero initial perturbation. In light of this difference, we test simply randomly initializing the perturbation, according to a uniform random distribution between $-\epsilon$ and ϵ for an ℓ_∞-norm ball perturbation set with radius ϵ, before applying the FGSM attack. We find that this simple adjustment to FGSM adversarial training can be used as an effective empirical defense. Crucially, we find that starting from a non-zero initial perturbation is the primary driver for success, regardless of the actual initialization, so long as the initialization is not restricted beyond the ϵ ball constraints. In fact, both starting with the previous mini-batch's perturbation or initializing from a uniformly random perturbation allow FGSM adversarial training to succeed at being significantly more robust to full-strength PGD adversarial attacks than previously believed. While randomized initialization for FGSM is not a new idea and was previously studied by Tramèr et al. [76], crucially, Tramèr et al. [76] use a different, more restricted random initialization and step size, which does not result in models robust to full-strength PGD adversaries.

[1] As reported by Shafahi et al. [67] using a different network architecture and an adversary with 20 steps and 10 restarts, which is strictly weaker than the adversary used in this work.

[2] As reported by Madry et al. [47] 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 work

Table 2.1: Standard and robust test accuracy of models trained on CIFAR-10 according to different adversarial training methods, and their corresponding training times. The threat model considered is the ℓ_∞-norm ball with radius $\epsilon = 8/255$, and the PGD test accuracy is calculated using 50 steps, step size $\alpha = 2/255$, and 10 random restarts.

	Test accuracy (%)		
Train method	Standard	PGD-50	Train 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
+ $\alpha = 10/255$	83.81	46.06	12.17
+ $\alpha = 16/255$	86.05	0.00	12.06
+ early stopping	70.93	40.38	8.81
Free $(m = 8)$ [67][1]	85.96	46.33	785
+ DAWNBench	78.38	46.18	20.91
PGD-7 [47][2]	87.30	45.80	4965.71
+ DAWNBench	82.46	50.69	68.8

To test the effect of initialization in FGSM adversarial training, we train several models, using the PreActResNet18 architecture, to be robust to ℓ_∞-norm ball perturbations at a radius $\epsilon = 8/255$ on CIFAR-10, 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 2.1, are consistent with the literature, and show that the model trained with zero-initialization (i.e. the original FGSM specification) is not robust against a PGD adversary. However, surprisingly, simply using a uniform random or previous mini-batch initialization instead of a zero initialization actually results in reasonable robustness levels (with uniform random initialization performing slightly better) that are comparable to both the free and the original PGD adversarial training methods.

Step size We recognize 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 too weak. We find that increasing the step size by a factor of 1.25 to $\alpha = 10/255$ further improves the robustness of the model so that it is on par with the best reported result using free adversarial training, as shown in Table 2.1. However, while increasing the step size to $\alpha = 10/255$ improves the robustness of the final model, we simultaneously find that forcing the resulting perturbation to lie on the boundary with a step size of $\alpha = 2\epsilon$ results in a model that is *not* robust to PGD adversarial attacks *at all*. This result is also shown in Table 2.1, where using $\alpha = 16/255$ results in 0% PGD accuracy. We show additional results of FGSM adversarial training with different step sizes in Figure 2.1, where we plot the mean and standard error of the robust test accuracy for models trained for 30 epochs over 3 random seeds, varying the step size from $\alpha = 1/255$ to

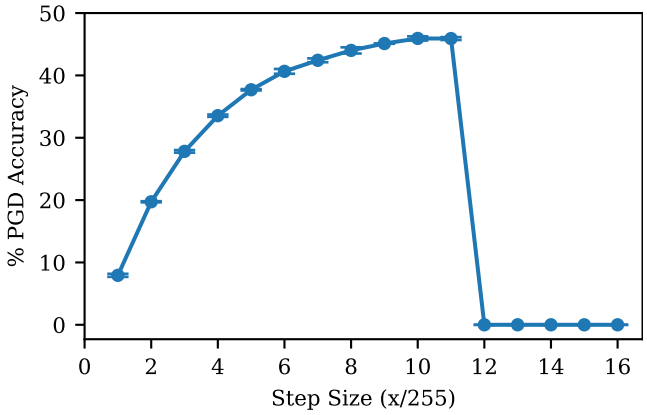


Figure 2.1: Robust test accuracy of models trained using Uniform R+FGSM adversarial training with different step sizes, where we consider ℓ_∞-norm ball perturbations with $\epsilon = 8/255$.

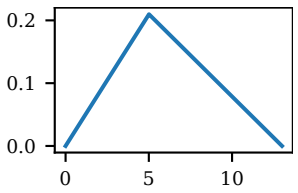
$\alpha = 16/255$. We find that we get improved robust performance as we increase the step size up to $\alpha = 10/255$. Beyond this, we see no further benefit, or worse, find that the model lacks any robustness to a PGD adversary. We discuss this failure mode in more detail in Section 2.2.4.

2.2.3 Incorporating standard training acceleration techniques

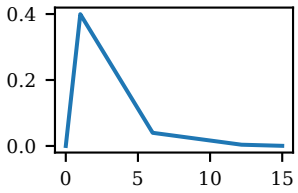
Given modifications to FGSM that result in successfully training robust models, we further find that FGSM can outperform free adversarial training in terms of efficiency when combining adversarial training with standard training speedup techniques. Ultimately, while these speedup techniques can be applied to any adversarial training technique, we find that free adversarial training is less amenable to these techniques. The acceleration techniques we consider have been largely publicized by the DAWNBench competitions, at which top submissions have shown that CIFAR-10 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 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 schedule Introduced by Smith [68] 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 [69]. A simple cyclic learning rate schedule increases the learning rate linearly from zero to a maximum learning rate and back down to zero (examples can be found in Figure 2.2). Using a cyclic learning rate allows CIFAR-10 architectures to converge to benchmark accuracies in tens of epochs instead of hundreds.

Mixed-precision arithmetic When GPU architectures have 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 [52]. This can drastically reduce the memory utilization, and when tensor cores are available, also reduce run time.



(a) CIFAR-10



(b) ImageNet

Figure 2.2: Cyclic learning rates used for FGSM adversarial training on CIFAR-10 and ImageNet over epochs. The ImageNet cyclic schedule is decayed further by a factor of 10 in the second and third phases.

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 run time 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.

2.2.4 Catastrophic overfitting

The failure modes of starting from a zero-initialized perturbation and using too large of a step size may explain why previous attempts at FGSM adversarial training failed. Many of the variations of FGSM adversarial training which have been found to not succeed all fail similarly: the model will very rapidly (over the span of a couple of epochs) appear to overfit to the FGSM adversarial examples. What was previously a reasonably robust model will quickly transform into a non-robust model which suffers 0% robust accuracy with respect to a PGD adversary. We find that in these scenarios, an interesting phenomenon occurs, where at some point during training, the FGSM train error suddenly decreases, while at the same time, the PGD test error suddenly increases. This phenomenon, which we call catastrophic overfitting, can be seen in Figure 2.3 which plots the learning curves for vanilla FGSM adversarial training from zero-initialization.

We find that this occurrence of this phenomenon can be detected on the training set as well, i.e. if we measure the PGD error on the training set we see similar curves to those in Figure 2.3, suggesting that the model overfits to the FGSM attack. Due to the rapid deterioration of robust performance, FGSM adversarial training can be salvaged to some degree with a simple early-stopping scheme by measuring PGD accuracy on a small mini-batch of training data, with minimal computational cost. In practice, we find that this can be a simple as a single mini-batch with a 5-step PGD adversary, which can be quickly checked at the end of the epoch. If robust accuracy with respect to this adversary suddenly drops, then the model has catastrophically overfit. Using a PGD adversary on a training mini-batch to detect catastrophic overfitting, we can early stop to avoid catastrophic overfitting and achieve a reasonable amount of robust performance. By using early stopping to catch the model at its peak performance before overfitting, FGSM adversarial training with larger step sizes can actually achieve some degree of robust accuracy, as shown in Figure 2.5. The recovered results for some of these failure modes are also shown

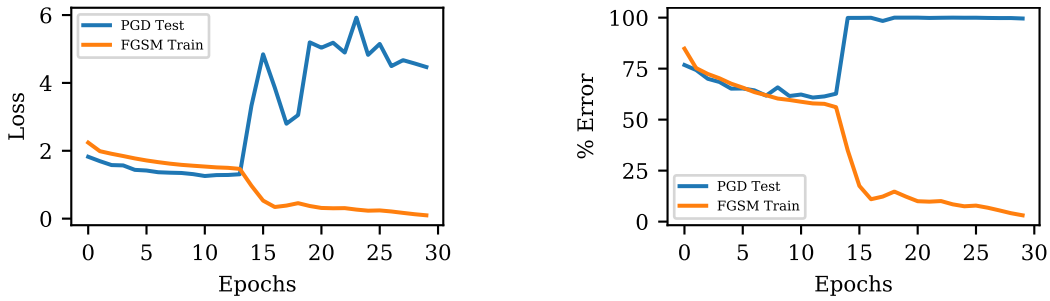


Figure 2.3: Learning curves for FGSM adversarial training plotting the training loss and error rates incurred by an FGSM and PGD adversary when trained with zero-initialization FGSM at $\epsilon = 8/255$, depicting the catastrophic overfitting where PGD performance suddenly degrades while the model overfits to the FGSM attack.

in Table 2.1. While too large of a step size can cause catastrophic overfitting in FGSM adversarial training, other design decisions, including specific learning rate schedules or numbers of training epochs, can also make catastrophic overfitting more likely to occur. Our modified version of FGSM adversarial training (with uniform random initialization and adjusted step size) successfully avoids catastrophic overfitting entirely when combined with the cyclic learning rate schedule, achieving its best robust performance with only 30 epochs of training. However, by training longer, or using a different learning rate schedule, even our Uniform R+FGSM algorithm can result in catastrophic overfitting.

We initially hypothesize that one of the reasons for this failure may lie in the lack of diversity in adversarial examples generated by these FGSM adversaries. For example, as opposed to PGD or Uniform R+FGSM, using FGSM with a zero initialization or using the random initialization scheme from Tramèr et al. [76] will result in adversarial examples whose features have been perturbed by $\{-\epsilon, 0, \epsilon\}$. We attempt to test this hypothesis by running a PGD adversarial attack on models which have catastrophically overfitted, and find that 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 2.4. However, follow-up work to ours further tests this hypothesis by training a model using PGD adversarial training but projecting the perturbations onto $\{-\epsilon, \epsilon\}$, and does not observe any catastrophic overfitting [2]. Andriushchenko and Flammarion [2] instead suggest that the overfitting results from the nature of the single-step approximation to the inner maximization rather than the diversity of perturbations.

2.2.5 Experiments

To demonstrate the effectiveness of Uniform R+FGSM adversarial training with the acceleration techniques described in Section 2.2.3, we run a number of experiments on MNIST, CIFAR-10, and ImageNet benchmarks. All CIFAR-10 experiments in this section are run on a single GeForce RTX 2080 Ti using the PreActResNet18 architecture [34], and all ImageNet experiments are run on a single machine with four GeForce RTX 2080 Tis using the ResNet50 architecture [33]. Our code for reproducing all experiments in this section and the corresponding

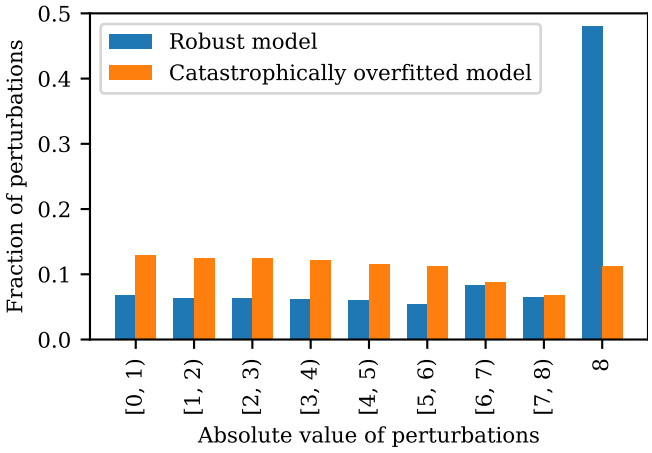


Figure 2.4: Histogram of the resulting perturbations from a PGD adversary for each feature for a successfully trained robust model and a catastrophically overfitted model on CIFAR-10.

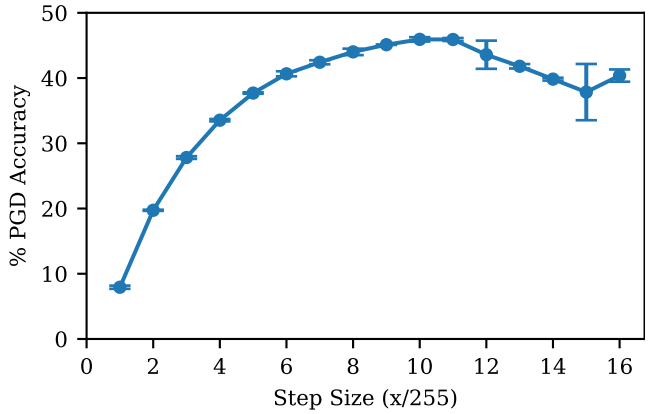


Figure 2.5: Robust test performance of FGSM adversarial training over different step sizes for $\epsilon = 8/255$ with early stopping to avoid catastrophic overfitting.

Table 2.2: Robust test accuracy (%) of Uniform R+FGSM and PGD adversarial training on MNIST.

		PGD		Verified
Method	Standard	$\epsilon = 0.1$	$\epsilon = 0.3$	$\epsilon = 0.1$
PGD	99.20	97.66	89.90	96.7
Uniform R+FGSM	99.20	97.53	88.77	96.8

trained model weights are made publicly available[3].

Implementation details All experiments using FGSM adversarial training are carried out with (uniform) random initial starting points and step size $\alpha = 1.25\epsilon$ as described in Section 2.2.2. 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. [67] but further strengthened with random restarts). Speedup with mixed-precision is incorporated with the Apex `amp` package at the `O1` optimization level for ImageNet experiments and `O2` without loss scaling for CIFAR-10 experiments.[4] For all methods, we use a batch size of 128, and SGD optimizer with momentum 0.9 and weight decay $5 \cdot 10^{-4}$. We report the average results over 3 random seeds. For FGSM and PGD adversarial training, we use a maximum learning rate of 0.2, and for free adversarial training we use a maximum learning rate of 0.04. For Table 2.1, we train FGSM for 30 epochs, PGD for 40 epochs, and free for 96 epochs. For runs using early-stopping, we use a 5-step PGD adversary with 1 restart on a single training mini-batch to detect overfitting to the FGSM adversaries, as described in Section 2.2.4.

MNIST

Verified performance on MNIST 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 [74]. 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. [74]. We use both PGD and Uniform R+FGSM adversarial training at $\epsilon = 0.3$, where the PGD adversary for training uses 40 iterations with step size 0.01 as done by Madry et al. [47]. The exact verification results can be seen in Table 2.2, 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]

[3] `https://github.com/locuslab/fast_adversarial`

[4] Since CIFAR-10 does not suffer from loss scaling problems, we find 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.

Table 2.3: Ablation study showing the performance of R+FGSM from Tramèr et al. [76] and our proposed FGSM modifications, over 10 random seeds, on MNIST.

Method	Step size	Initialization	Robust accuracy (%)
R+FGSM [76]	0.15	Hypercube(0.15)	34.58 ± 36.06
R+FGSM (+ full step size)	0.30	Hypercube(0.15)	26.53 ± 32.48
R+FGSM (+ uniform init.)	0.15	Uniform(0.3)	72.92 ± 10.40
Uniform + full (ours)	0.30	Uniform(0.3)	86.21 ± 00.75

Comparison to R+FGSM on MNIST We now show the improvement of our modified version of FGSM (Uniform R+FGSM) to R+FGSM, a randomized version of FGSM adversarial training proposed by Tramèr et al. [76]. As described in Section 2.2.1, R+FGSM differs from our method in two ways. First, it initializes the perturbation on the surface of a hypercube with radius $\epsilon/2$, such that $\delta = \frac{\epsilon}{2}\mathcal{N}(0,1)$. Second, it uses a step size of $\epsilon/2$. 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 2.3.

We find that using a uniform initialization adds the greatest marginal improvement to the original R+FGSM attack, while using a full step size does not appear to help on its own. Implementing both of these improvements results in the form of FGSM adversarial training presented in this work. Additionally, we note that R+FGSM as done by Tramèr et al. [76] 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.

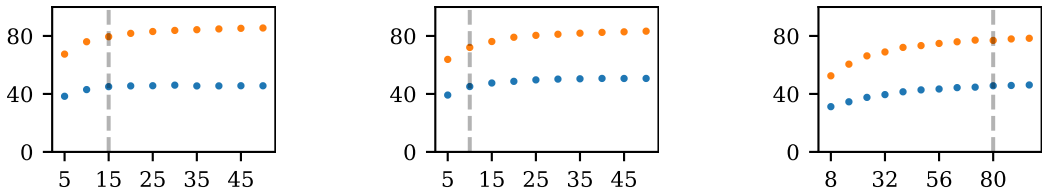
CIFAR-10

We begin our CIFAR-10 experiments by using cyclic learning rate schedules and mixed-precision arithmetic from Section 2.2.3 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 [69].

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. While all the adversarial training methods benefit greatly from the cyclic learning rate schedule, we show in Figure 2.6 that both FGSM and PGD adversarial training require much fewer epochs than free adversarial training, and consequently reap the greatest speedups.

[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`.



(a) Uniform R+FGSM (b) PGD (c) Free ($m = 8$)

Figure 2.6: Performance of models trained on CIFAR-10 at $\epsilon = 8/255$ using cyclic learning rate schedules and mixed precision arithmetic, 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 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 (PGD) accuracy. The vertical dotted line indicates the minimum number of epochs needed to train a model to 45% robust accuracy.

Table 2.4: Time to train a robust CIFAR-10 classifier to 45% robust accuracy using various adversarial training methods with the DAWNBench techniques of cyclic learning rate schedules and mixed-precision arithmetic, showing significant speedups for all forms of adversarial training.

Method	Epochs	Seconds/epoch	Total time (minutes)
PGD-7 + DAWNBench	10	104.94	17.49
Free ($m = 8$) + DAWNBench	80	13.08	17.44
Uniform R+FGSM + DAWNBench	15	25.36	6.34
PGD-7 [47][6]	205	1456.22	4965.71
Free ($m = 8$) [67][7]	205	197.77	674.39

Using the minimum number of epochs needed for each training method to reach a baseline of 45% robust test accuracy, we report the total training time in Table 2.4. We find that while all adversarial training methods benefit from the DAWNBench improvements, FGSM adversarial training is the fastest, capable of learning a robust CIFAR-10 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.

ImageNet

Finally, we apply all of the same techniques (Uniform R+FGSM adversarial training, mixed-precision, and cyclic learning rate schedules) 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

Table 2.5: Standard and robust accuracy of models trained using Uniform R+FGSM and free adversarial training on ImageNet at $\epsilon = 2/255$ and $\epsilon = 4/255$, as well as their corresponding training times.

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

Table 2.6: Time to train a robust ImageNet classifier using Uniform R+FGSM and free adversarial training methods.

Method	Precision	Epochs	Min/epoch	Time (hrs)
Uniform R+FGSM (phase 1)	single	6	22.65	2.27
Uniform R+FGSM (phase 2)	single	6	65.97	6.60
Uniform R+FGSM (phase 3)	single	3	114.45	5.72
Uniform R+FGSM	single	15		14.59
Free	single	92	34.04	52.20
Uniform R+FGSM (phase 1)	mixed	6	20.07	2.01
Uniform R+FGSM (phase 2)	mixed	6	53.39	5.34
Uniform R+FGSM (phase 3)	mixed	3	95.93	4.80
Uniform R+FGSM	mixed	15		12.14
Free	mixed	92	25.28	38.76

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 2.5, 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. [67].

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 2.6.[8] We compare to the best performing variation of free adversarial training which which uses $m = 4$ mini-batch replays over 92 epochs of training (scaled down accordingly to 23 passes over the data).

[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 2.7: Free adversarial training with DAWNBench on ImageNet, considering ℓ_∞-norm ball perturbations with radius $\epsilon = 4/255$.

Method	Step size	Epochs	Accuracy (%) Standard	PGD+1	PGD+10
Free $(m = 3)$ + DAWNBench	$4/255$	15	49.87	22.78	22.18
Free $(m = 3)$ + DAWNBench	$5/255$	15	50.48	22.88	22.25
Free $(m = 3)$ + DAWNBench	$4/255$	30	49.87	28.17	27.08
Free $(m = 3)$ + DAWNBench	$5/255$	30	50.48	28.73	27.81
Free $(m = 4)$	$4/255$	92	60.42	31.22	31.08
Uniform R+FGSM + DB	$5/255$	15	55.45	30.28	30.18

Comparison to free adversarial training + DAWNBench Free adversarial training can also be enhanced with mixed-precision arithmetic, which reduces the runtime by 25%, but directly combining free adversarial training with the other fast techniques used in FGSM adversarial training for ImageNet (cyclic learning rate schedules, progressive resizing, and batch-norm regularization) results in reduced performance. Since ImageNet is too large to run a comprehensive search over the various parameters as was done for CIFAR-10 in Table 2.4, 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 $m = 3$ mini-batch replays, 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. The results are shown in Table 2.7.

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 work, 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 CIFAR-10, 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 slower than one epoch of free training, a fairer comparison is to give free adversarial training a larger number of training epochs. Even with double the number of training epochs, we find that the final robust performance is better, but does not quite reach the original robust performance of free adversarial training.

2.2.6 Discussion

Our findings show that FGSM adversarial training, when used with uniform random initialization and an adapted step size, can result in much stronger robustness levels than vanilla FGSM adversarial training, while being significantly faster than alternative methods of adversarial training.

While a single iteration of FGSM adversarial training is computationally more expensive than that of free adversarial training, training using FGSM converges significantly faster, especially when using a cyclic learning rate schedule. As a result, we are able to learn adversarially robust classifiers for CIFAR-10 in minutes and for ImageNet in hours, even faster than free adversarial training but with comparable levels of robustness. Leveraging these significant reductions in time to train robust models can enable future work to iterate even faster, and accelerate research in learning models which are resistant to adversarial attacks.

2.3 Robust overfitting in adversarial training

In standard training of deep networks, it is common practice to use overparameterized networks and train for as long as possible. There are numerous studies that show, both theoretically and empirically, that such practices surprisingly do not unduly harm the generalization performance of the classifier [89]. Deep learning models can often be trained to zero training error, effectively memorizing the training set, seemingly without causing any detrimental effects on the generalization performance. This phenomenon has been widely studied both from the theoretical [58] and empirical perspectives [5], and remains such a hallmark of deep learning practice that it is often taken for granted.

In this section, we empirically study this phenomenon in the setting of adversarial training, finding that overfitting to the training set in this setting does in fact harm robust performance to a very large degree. This is shown, for instance, in Figure 2.7 for adversarial training on CIFAR-10, where the robust test error decreases immediately after the first learning rate decay, and only increases beyond this point. We show that this phenomenon, which we refer to as "robust overfitting", can be observed on multiple datasets beyond CIFAR-10, including SVHN, CIFAR-100, and ImageNet.

Based upon this observed effect, we show that the performance gains of many algorithmic improvements upon PGD-based adversarial training [55, 83, 84, 91] can be matched by simply early stopping during training. 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 robust performance as state-of-the-art methods, a result which we evaluate externally [17]. For example, vanilla PGD-based adversarial training [47] can achieve 56.8% robust test accuracy against a PGD adversary, considering ℓ_∞-norm ball perturbations with radius $\epsilon = 8/255$, on CIFAR-10 when training is stopped early, on par with the 56.6% robust test accuracy reported by TRADES [91] against the same adversary. We find that this phenomenon is not unique to ℓ_∞-norm ball perturbations and is also seen in adversarial training for robustness towards ℓ_2-norm ball perturbations. For instance, early stopping a model trained on CIFAR-10 against an ℓ_2 adversary with radius $128/255$ can increase the robust test accuracy from 68.9% to 71.6%.

We further study various empirical properties of overfitting for adversarially robust training and how they relate to standard training. Since the effects of such overfitting appear closely tied to the learning rate schedule, we begin by investigating how changes to the learning rate schedule affect the prevalence of robust overfitting and its impacts on model performance. We next explore how known connections between the hypothesis class size and generalization in

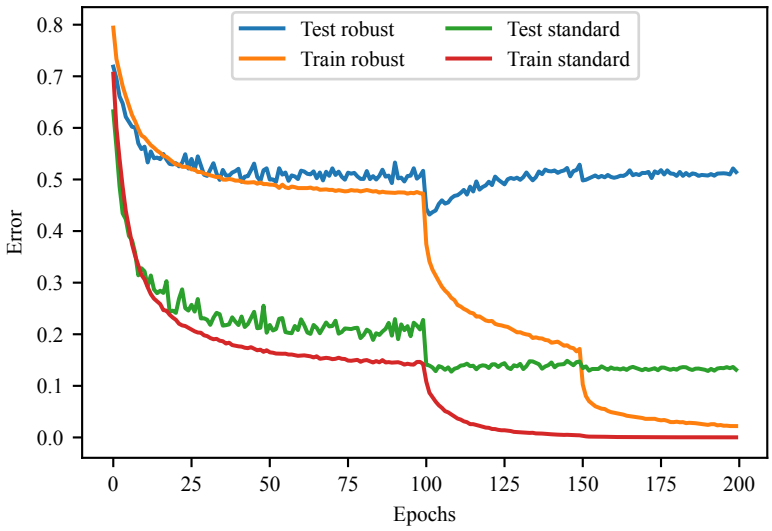


Figure 2.7: The learning curves for a robustly trained model replicating the experiment done by Madry et al. [47] on CIFAR-10, plotting accuracy. The curves demonstrate *robust overfitting*; shortly after the first learning rate decay the model momentarily attains 56.8% robust accuracy, and is actually more robust than the model at the end of training, which only attains 48.6% robust test accuracy against a 10-step PGD adversary for ℓ_∞ radius of $\epsilon = 8/255$. The learning rate is decayed at 100 and 150 epochs.

deep networks translate to the robust setting, and show that the "double descent" generalization curves seen in standard training [5] also hold for robust training [56]. However, although this is used as a justification for the lack of overfitting in the standard setting, surprisingly, changing the hypothesis class size does not actually mitigate the robust overfitting that is observed during training.

Our final contribution is to investigate several techniques for preventing robust overfitting. We first explore the effects of classic statistical approaches for combating overfitting beyond early stopping, namely explicit ℓ_1 and ℓ_2 regularization. We then study more modern approaches using data augmentation, including cutout [19], mixup [92], and semisupervised learning methods, which are known to empirically reduce overfitting in deep networks. Ultimately, while these methods can mitigate 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. In fact, even combining regularization methods with early stopping tends to not significantly improve on early stopping alone. We find that the one exception is data augmentation with semi-supervised learning, where although the test performance can vary wildly even when training has converged, at select epochs it is possible to find a model with improved robust performance over simple early stopping. The code for reproducing all the experiments in this section, along with pre-trained model weights and training logs, is made publicly available.[9][10]

[9] `https://github.com/locuslab/robust_overfitting`

[10] Since there are over 75 models trained in this work, we selected a subset of pretrained models to release (e.g. those which are for Wide ResNets since those take the most time to train, and can achieve the best performance.)

2.3.1 Related work

Highly relevant to work presented in this section are those that study the general problem of overfitting in machine learning. Both regularization and early stopping have been well-studied in classical statistical settings to reduce overfitting and improve generalization. Although ℓ_2 regularization (also known as weight decay) is commonly used for training deep networks [41], 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 [54]. Indeed, it is now known that the standard bias-variance trade-off from classical statistical learning theory fails to explain why deep networks can generalize so well [89]. 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 [5, 56]. The generalization gap for robust deep networks has also been studied from a learning theoretic perspective in the context of data complexity [66] and Rademacher complexity [86].

Also relevant to this work are methods specific to deep learning that empirically reduce overfitting and improve performance of deep networks. For example, dropout is a commonly used stochastic regularization technique that randomly drops units and their connections from the network during training [71] 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 [19] 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 [92] 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 [1, 14, 88].

2.3.2 The effect of overfitting in adversarial training

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 test loss continues to decrease with the training loss. On the contrary, for the setting of adversarially robust training we discover that 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 in Figure 2.7. 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.

In order to better understand the scope of robust overfitting, we train a number of models on

[11]Note that the TRADES repository does not provide default training parameters 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)

Table 2.8: Performance of adversarial training over a variety of datasets, training algorithms, and perturbation threat models, showing the occurrence of robust overfitting.

				Robust test accuracy (%)			Standard test accuracy (%)		
Dataset	Method	Norm	ϵ	Final	Best	Diff.	Final	Best	Diff.
SVHN	PGD	ℓ_∞	$8/255$	54.4 ± 0.40	61.0	-6.6	90.0 ± 0.15	89.8	0.2
		ℓ_2	$128/255$	73.6 ± 0.27	74.8	-1.2	93.0 ± 0.23	92.8	0.2
CIFAR-10	PGD	ℓ_∞	$8/255$	48.6 ± 0.41	56.8	-8.2	86.6 ± 0.19	86.1	0.5
		ℓ_2	$128/255$	68.9 ± 0.46	71.6	-2.7	89.0 ± 0.08	88.7	0.3
	FGSM	ℓ_∞	$8/255$	40.2 ± 0.09	46.3	-6.1	87.6 ± 0.21	86.4	1.2
		ℓ_2	$128/255$	68.4 ± 0.18	70.8	-2.4	90.1 ± 0.16	89.5	0.6
	TRADES	ℓ_∞	$8/255$	49.4 ± 0.31	55.0	-5.6	85.0 ± 0.24	84.1	0.9
		ℓ_2 [11]	$128/255$	41.8 ± 0.66	46.4	-4.6	66.1 ± 0.95	84.3	-18.2
CIFAR-100	PGD	ℓ_∞	$8/255$	21.4 ± 0.39	28.1	-6.7	54.1 ± 0.23	52.7	1.4
		ℓ_2	$128/255$	37.5 ± 0.09	43.2	-5.7	60.1 ± 0.22	62.5	-2.4
ImageNet	PGD	ℓ_∞	$4/255$	14.5 ± 8.87	37.3	-22.8	49.5 ± 14.32	63.0	-13.5
		ℓ_2	$76/255$	5.2 ± 1.16	37.0	-31.8	36.8 ± 6.80	59.9	-23.1

different datasets (SVHN, CIFAR-10, CIFAR-100, and ImageNet), using different adversarial training methods (PGD, FGSM, and TRADES), and with different perturbation threat models (ℓ_∞ and ℓ_2). We observe the learning curves and compare the best and final checkpoint of each training run. We find that robust overfitting occurs across all of these datasets, algorithmic approaches, and perturbation threat models, indicating that it is a general property of the adversarial training formulation and not specific to a particular problem, as can be seen in Table 2.8. The final accuracy is an average over the final 5 epochs of when the model has converged, whereas the best accuracy is the highest test accuracy of all model checkpoints during training. 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 a decrease of 8.2% robust accuracy for CIFAR-10 and 22.8% robust accuracy for ImageNet against an ℓ_∞ adversary, to highlight a few. Robust overfitting is also not specific to PGD-based adversarial training, and affects fast adversarial training methods such FGSM adversarial training, as well as top performing algorithms for adversarially robust training such as TRADES [91]. We discuss these results, as well as experimental details, below.

Implementation details We default to using the PreActResNet18 model architecture, with the exception of Wide ResNet with width factor 10 for ℓ_∞ adversaries on CIFAR-10 (for a proper comparison to what is reported for TRADES), and ResNet50 for ImageNet. For CIFAR-10 and CIFAR-100, we train with the SGD optimizer using a batch size of 128, a piecewise 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 fine-tune the pretrained model from Engstrom et al. [23] and continue training with the exact same parameters with a learning rate of 0.001. We consider the ℓ_∞ threat model

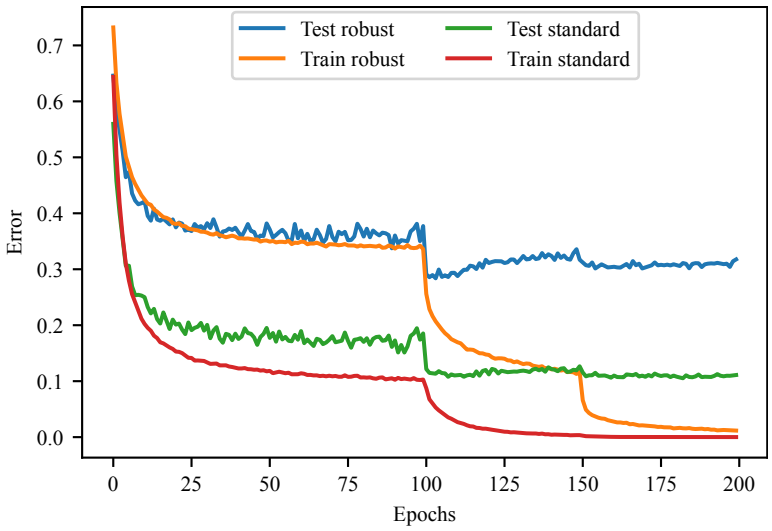


Figure 2.8: Learning curves when training using PGD for robustness to ℓ_2-norm ball perturbations of radius $128/255$ for CIFAR-10.

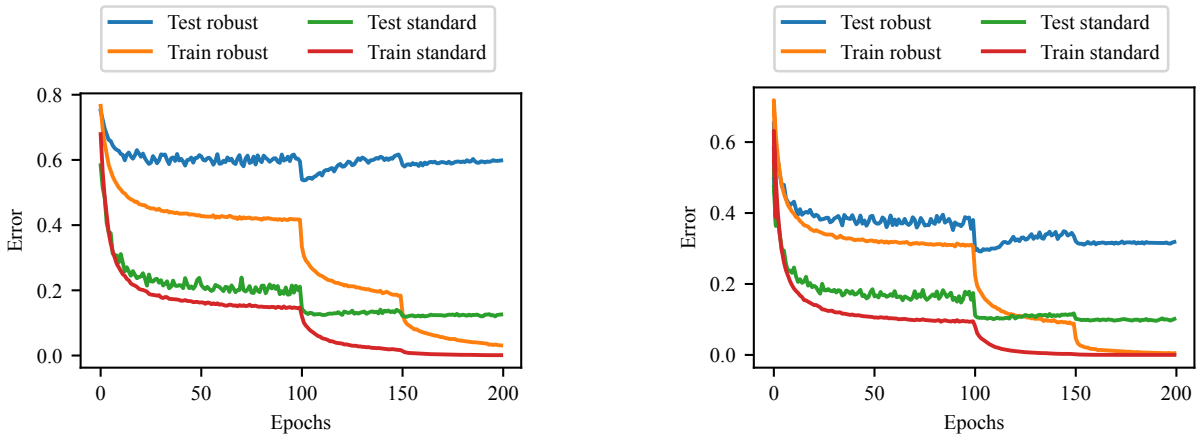
with radius $8/255$, with the PGD adversary taking 10 steps of size $2/255$ on all datasets except for ImageNet, which uses an adversary with 5 steps of size $0.9/255$ within a ball of radius $4/255$. We consider the ℓ_2 threat model with radius $128/255$, with the PGD adversary taking 10 steps of size $15/255$ on all datasets except for ImageNet, which uses an adversary with 7 steps of size 0.5 within a ball of radius 3.

Robust overfitting of various adversarial training methods on CIFAR-10

We first show that robust overfitting is not specific to one adversarial training algorithm, but appears to be a universal phenomenon. We run several experiments on CIFAR-10 using PGD, Uniform R+FGSM, and TRADES adversarial training, and show the occurrence of robust overfitting in each setting below.

PGD We have already shown in Figure 2.7 that robust overfitting occurs when using PGD adversarial training for ℓ_∞-norm ball perturbations. We also show in Figure 2.8 that robust overfitting occurs for PGD when training for robustness towards ℓ_2-norm ball perturbations.

Uniform R+FGSM For Uniform R+FGSM training on CIFAR-10, 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_∞ adversary to $7/255$ to avoid catastrophic overfitting, but still see robust overfitting. The convergence curves showing that robust overfitting still occurs in both the ℓ_∞ and ℓ_2 settings are shown in Figure 2.9. We emphasize catastrophic overfitting 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



(a) ℓ_∞ (b) ℓ_2

Figure 2.9: Learning curves for adversarially training a CIFAR-10 classifier with a Uniform R+FGSM adversary against different ℓ_p-norm ball threat models.

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 2.9, suggesting that this is related to the generalization properties of adversarially robust training rather than the strength of the adversary.

TRADES For TRADES we use the publicly released implementation of both the defense and attack from Zhang et al. [91] 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 as TRADES learning rate),[12] and the standard learning rate schedule used by Madry et al. [47] for PGD adversarial training, which decays at 100 epochs and 150 epochs (denoted as Madry learning rate). We additionally explore both the PreActResNet18 architecture that we use extensively in this work, as well as the Wide ResNet architecture which TRADES uses. The corresponding learning curves for each combination of learning rate schedule and model architecture can be found in Figure 2.10 for the ℓ_∞ threat model. We note that in three of the four cases, we see a clear instance of robust overfitting. Only the TRADES learning rate schedule on the smaller, PreActResNet18 model doesn't indicate any degradation in robust test set performance. This is likely due the shortened learning rate schedule, which implicitly early stops training, combined with the regularization induced by a smaller architecture having less representational power. 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 schedule does (and also achieves a slightly better best checkpoint).

[12]This is the learning rate schedule described in the paper by Zhang et al. [91]. 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.

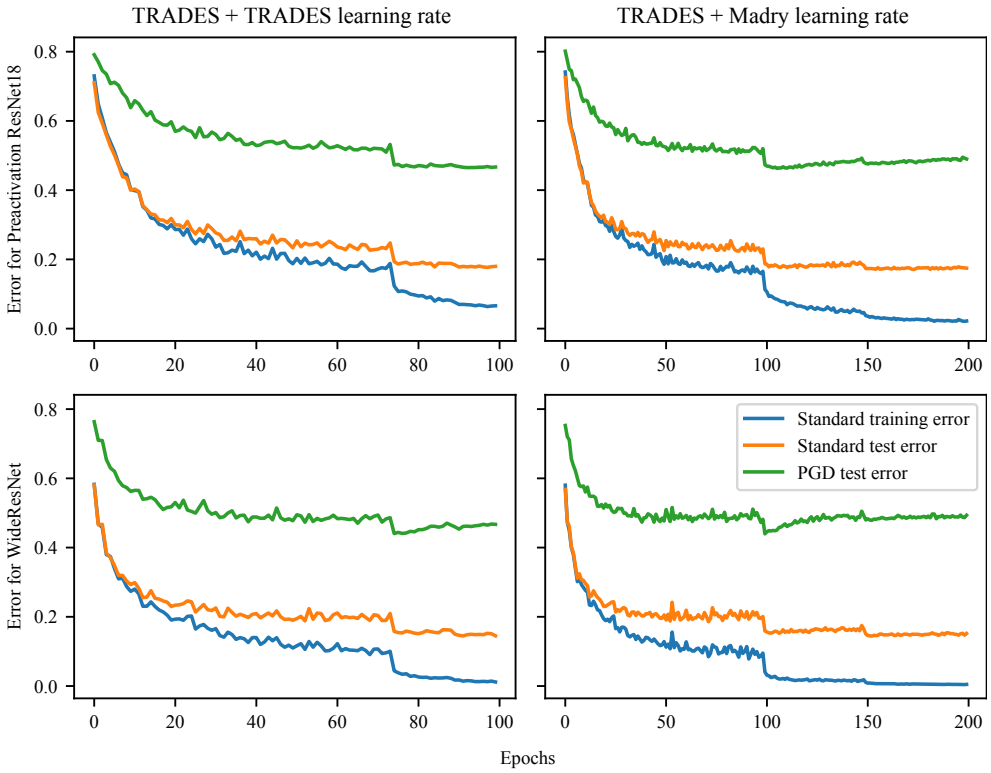
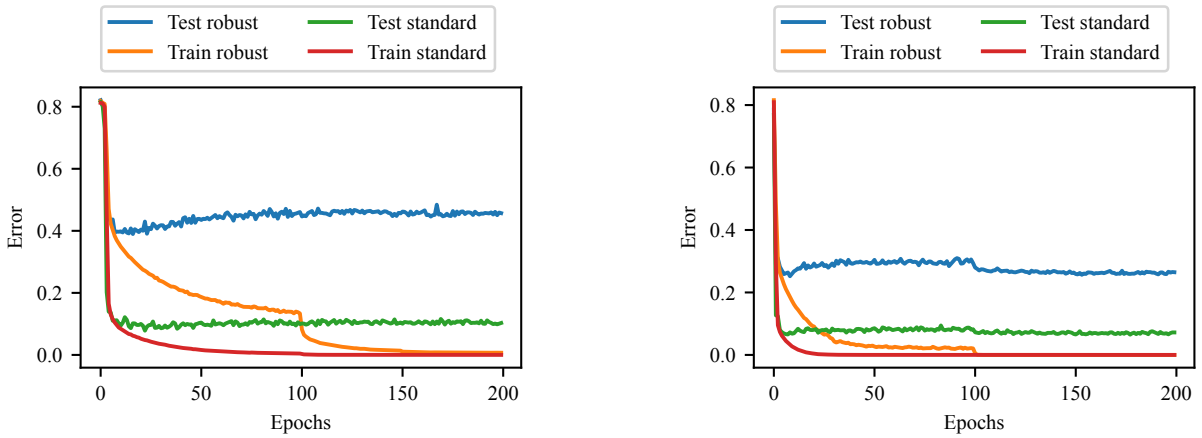


Figure 2.10: Learning curves when training using TRADES for robustness to ℓ_∞ perturbations of radius $8/255$ on combinations of different learning rate schedules and architectures for CIFAR-10.



(a) ℓ_∞ (b) ℓ_2

Figure 2.11: Learning curves for adversarially training an SVHN classifier with a PGD adversary against different ℓ_p-norm ball threat models.

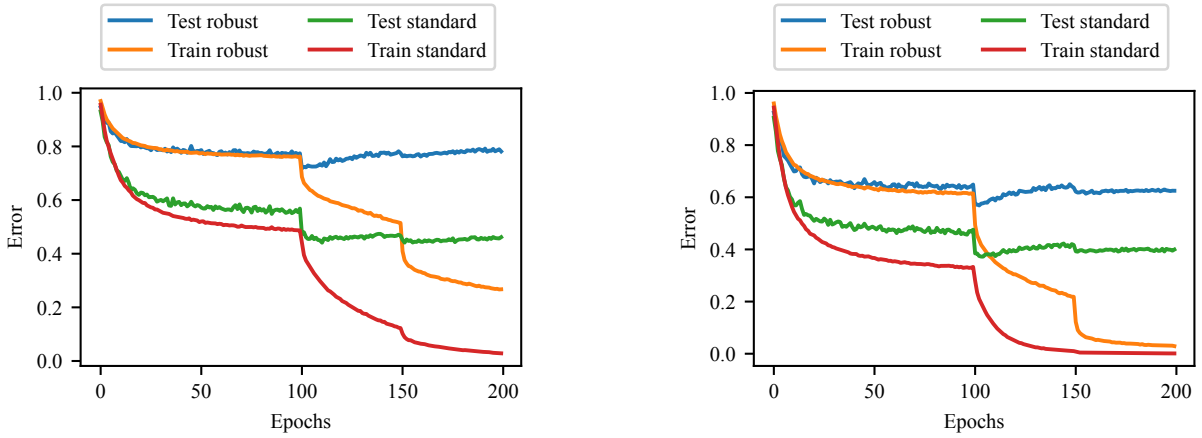
Occurrence of robust overfitting on additional datasets

We next show that robust overfitting during adversarial training is not specific to one dataset, but occurs on multiple datasets. We run several experiments on SVHN, CIFAR-100, and ImageNet, and show the occurrence of robust overfitting in each setting below.

SVHN Figure 2.11 contains the convergence plots for the PGD-based adversarial training experiments on SVHN for ℓ_∞ and ℓ_2 perturbations. 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 61.0% robust accuracy, which is a 6.6% improvement over the 54.4% robust accuracy achieved at the end of training.

CIFAR-100 Figure 2.12 contains the convergence plots for the PGD-based adversarial training experiments on CIFAR-100 for ℓ_∞ and ℓ_2 perturbations respectively. We find that robust overfitting on this dataset reflects the CIFAR-10 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 28.1% robust accuracy, which is a 6.7% improvement over the 21.4% robust accuracy achieved at the end of training.

ImageNet Figure 2.13 contains the convergence plots for our continuation of PGD-based adversarial training experiments on ImageNet for ℓ_∞ and ℓ_2 perturbations respectively. Using training logs provided by Engstrom et al. [23], we know the pretrained ℓ_2 robust ImageNet model has 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



(a) ℓ_∞ (b) ℓ_2

Figure 2.12: Learning curves for adversarially training a CIFAR-100 classifier with a PGD adversary against different ℓ_p-norm ball threat models.

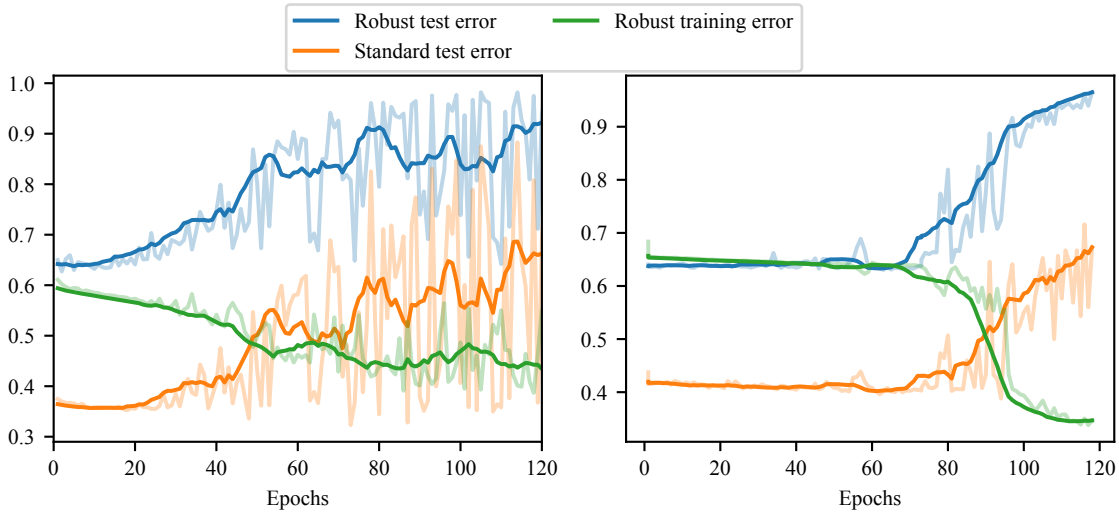


Figure 2.13: Continuation of training released pre-trained ImageNet models for ℓ_∞ (left) and ℓ_2 (right). The number of epochs indicate the number of additional epochs the pre-trained models were trained for.

long it was trained and under what schedule, however the pretrained model checkpoint indicates that the model has been trained for at least one epoch at learning rate 0.001, so we continue training from this point on. The ℓ_∞ pre-trained model appears to have not yet converged for the checkpointed learning rate, and so further training without any form of learning rate decay is able to gradually deteriorate the performance of the model. The ℓ_2 pre-trained model seems 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. 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 2.8, where ImageNet has the greatest variation in final accuracy (both robust and standard). Training the models further can in fact improve the performance of the pretrained model slightly at specific checkpoints (e.g. from 33.6% initial robust test accuracy up to 37.3% robust test accuracy at the best checkpoint for ℓ_∞), however eventually the ImageNet models suffer greatly from robust overfitting, with an average decrease of 22.8% robust accuracy for the ℓ_∞ model and 31.8% robust accuracy for the ℓ_2 model.

2.3.3 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 CIFAR-10, as shown in Figure 2.14. We consider the following types of learning rate schedules:

1. **Piecewise decay:** This schedule 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 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 [68]. We adopt the version 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.

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 accuracy during training is strictly worse than the best robust test accuracy during training with the discrete piecewise decay schedule. The cyclic learning rate is the exception here, which has two

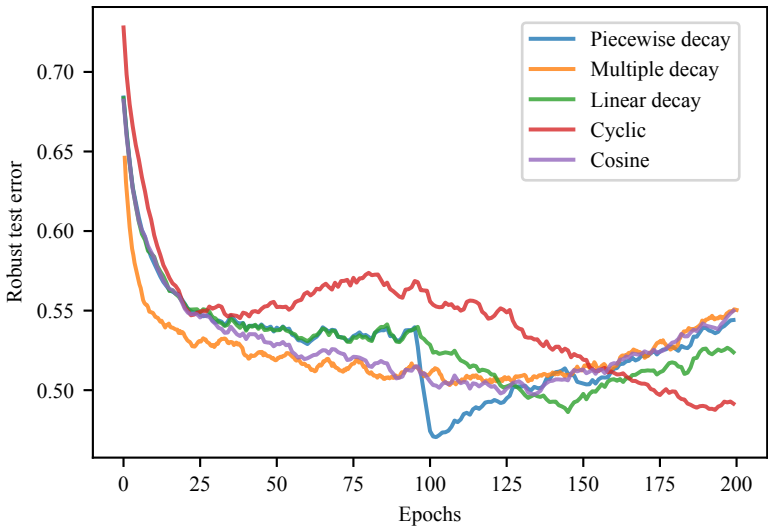


Figure 2.14: Robust test error over training epochs for various learning rate schedules on CIFAR-10. None of the alternative smoother learning rate schedules can achieve a peak performance competitive with the standard piecewise decay learning rate, indicating that the peak performance is obtained by 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.

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 find 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 accuracy for both the best checkpoint and the final converged model.

Tuning the piecewise decay schedule Since the piecewise decay schedule appears 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 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 accuracy of the best checkpoint by 0.5%, as seen in Table 2.9. Note that robust overfitting still occurs in these tuned learning rate schedules as seen in Figures 2.15, 2.16, and 2.17, which show the learning curves for each one of the models shown in Table 2.9.

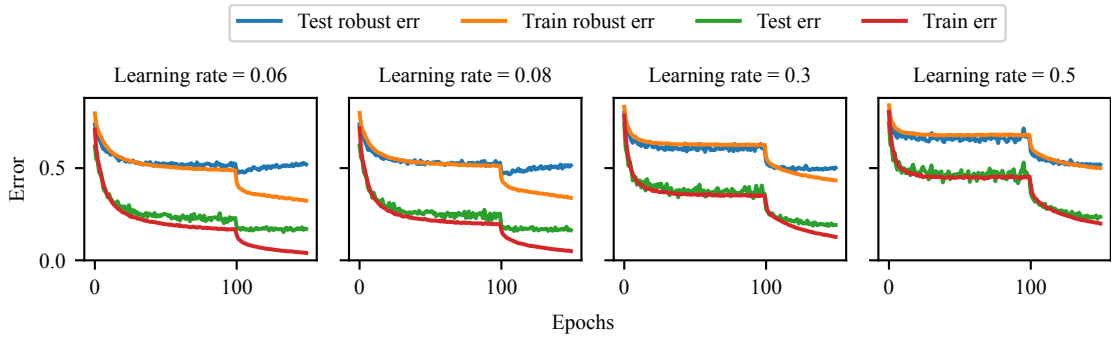


Figure 2.15: Learning curves for a piecewise decay schedule with a modified starting learning rate.

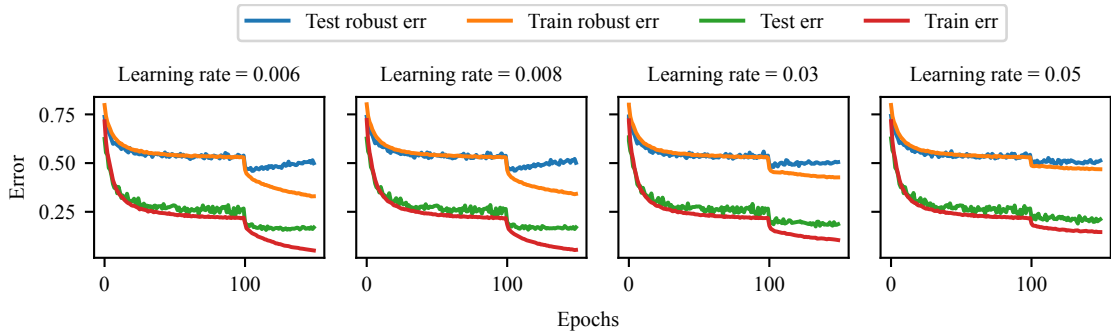


Figure 2.16: Learning curves for a piecewise decay schedule with a modified ending learning rate.

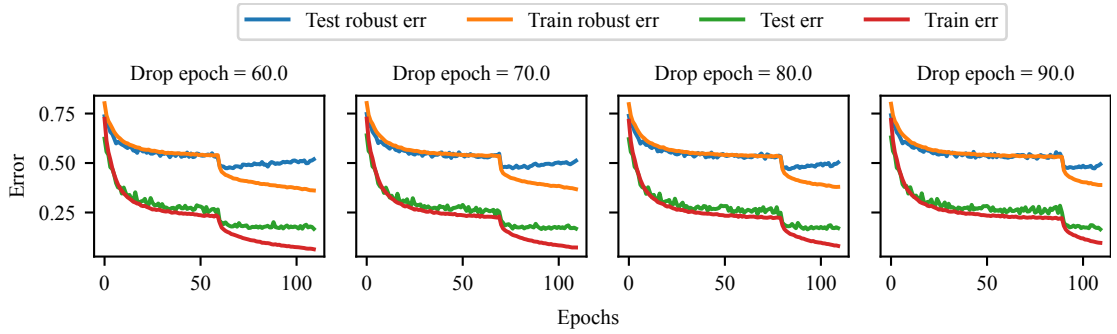


Figure 2.17: Learning curves for a piecewise decay schedule with a modified epoch at which the decay takes effect.

Table 2.9: Tuning experiments using stochastic gradient descent to optimize the best robust test accuracy obtained from the piecewise decay schedule for a PreActResNet18 on CIFAR-10.

Decay epoch	Start LR	End LR	PGD acc. (%)
100	0.10	0.01	53.3
60	0.10	0.010	52.6
70			52.7
80			53.1
90			52.7
100	0.06	0.010	52.6
	0.08		53.3
	0.30		51.3
	0.50		49.0
100	0.10	0.006	54.0
		0.008	53.9
		0.030	52.2
		0.050	50.7

2.3.4 Early stopping to mitigate robust overfitting

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 publicly released code for TRADES uses the best robust performance on the test set from an earlier checkpoint in order to achieve their top reported result of 56.6% robust accuracy against an ℓ_∞ PGD adversary with $\epsilon = 8/255$ on CIFAR-10, a number which was 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. Specifically, the robust test accuracy of the model at the checkpoint with the best performance on the test set is 55.9% whereas the robust test accuracy of the model at the end of training has decreased to 49.4%.[13]

Surprisingly, when we early stop vanilla PGD 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. [91] for TRADES, the model checkpoint from PGD training with the best performance on the test set achieves 57.7% robust test

[13]We use the public implementation of TRADES available at `https://github.com/yaodongyu/TRADES` and simply run it to completion using the same learning rate decay schedule used by Madry et al. [47].

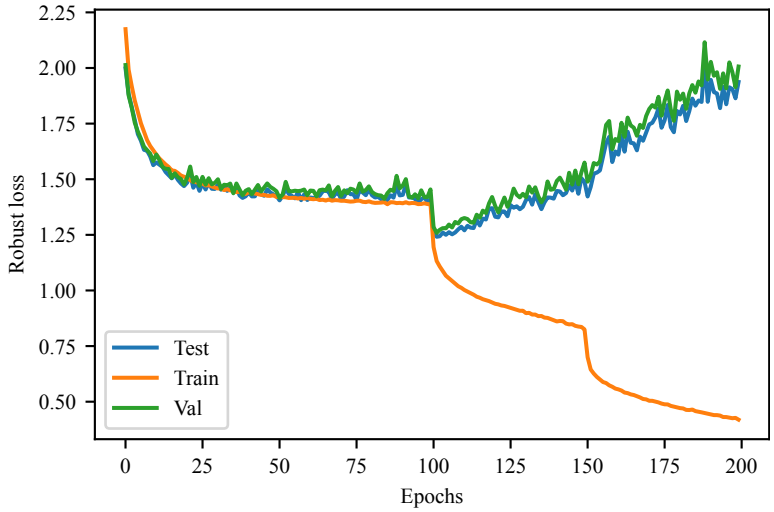


Figure 2.18: Learning curves for a CIFAR-10 PreActResNet18 model trained with a hold-out validation set of 1,000 examples. We find that the hold-out validation set is enough to reflect the test set performance, and stopping based on the validation set is able to prevent overfitting and recover 53.1% robust test accuracy, in comparison to 53.3% achieved by the best-performing model checkpoint.

accuracy, which is actually slightly better than the best reported result for TRADES from Zhang et al. [91].[14]

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 CIFAR-10 training set for validation purposes, we use validation-based early stopping to achieve 53.1% robust accuracy on the test set, in comparison to the 53.3% robust accuracy achieved by the best-performing model checkpoint for a PreActResNet18 architecture. The resulting validation curve during training closely matches the testing curve as seen in Figure 2.18, 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 [62]).

2.3.5 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* [5]. 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 work; since training for longer can also be viewed as increasing

[14]We found that our implementation of the PGD adversary to be slightly more effective, increasing the robust test accuracy of the TRADES model and the PGD trained model to 55.0% and 56.8% respectively.

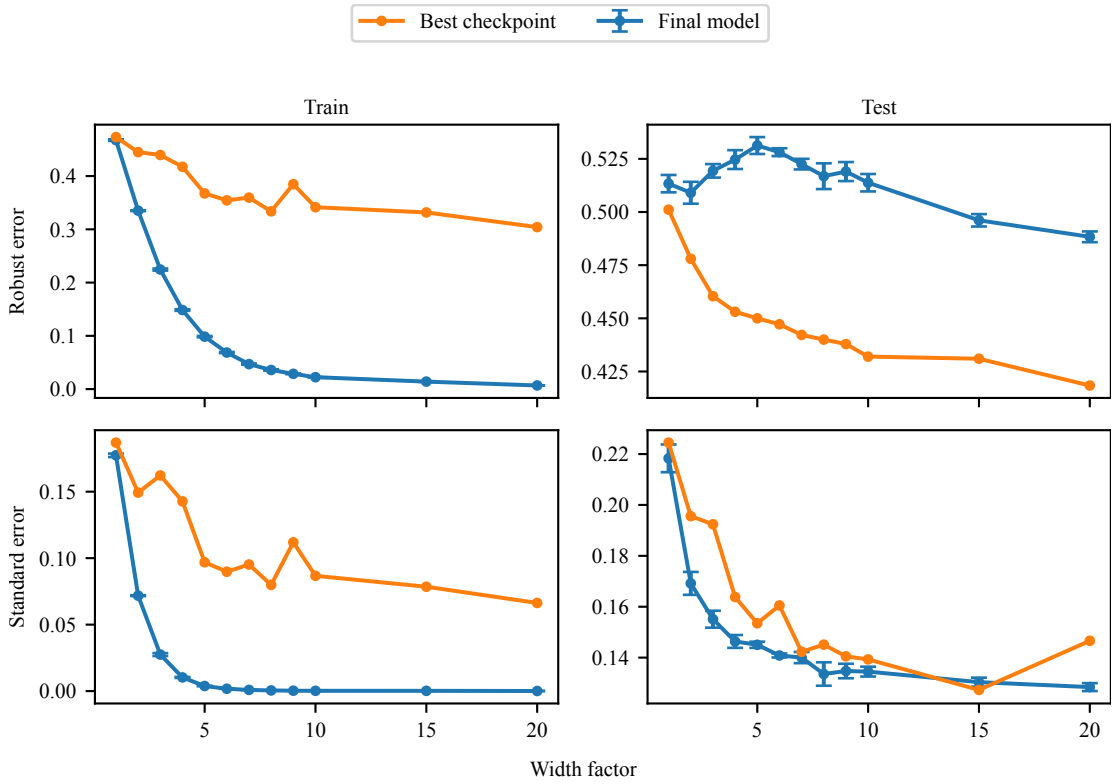


Figure 2.19: Standard and robust error on the train and test set across Wide ResNets with varying width factors depicting double descent for adversarially robust generalization, where hypothesis class complexity is controlled by varying the width factor.

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. [56] for the ℓ_2 robust setting, and so in this section we show that this generally holds also in the ℓ_∞ robust setting.

We explore these properties by training multiple adversarially robust Wide ResNets [87] with depth 28 and varying widths to control model complexity. All models were trained with the same training parameters described in Section 2.3.2. We record the final model performance as the average performance over the last 5 epochs with the corresponding width factor from training until convergence, and save the checkpoint with the highest robust test accuracy measured during training as the "best" checkpoint.

In Figure 2.19, we see that no matter how large the model architecture is, robust overfitting still results in a significant gap between the best and final robust test performance. This can also be visualized by looking at the learning curves for each width, shown in Figure 2.20. Notably, we also see that adversarially robust training still produces the double descent generalization curve,

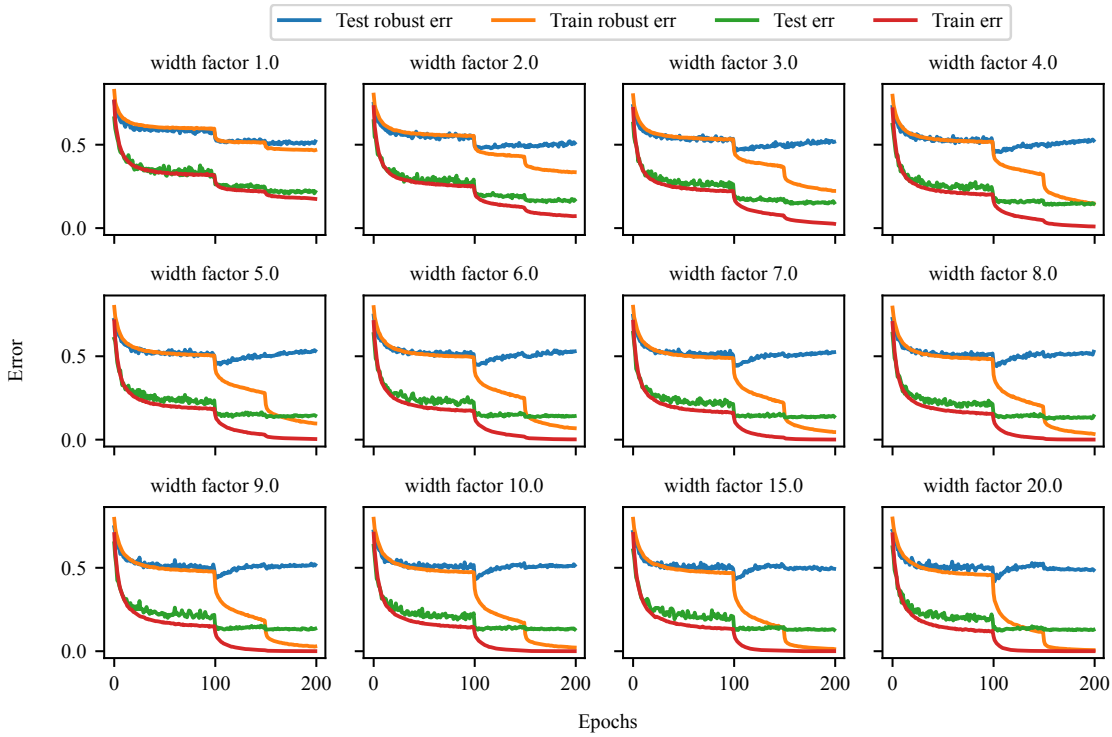


Figure 2.20: Learning curves for training Wide ResNets with different width factors.

as the robust test error increases and then decreases again with architecture size, suggesting that the double descent and robust overfitting are separate phenomenon. Even the lowest robust test error achieved during training continues to decrease with increased model complexity, suggesting that larger architecture sizes are still beneficial for adversarially robust training despite robust 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 51.2% robust test accuracy at the end of training and 58.2% robust test accuracy at the best checkpoint. This marks a further improvement over the more typical choice of width factor 10 which achieves 48.6% robust test accuracy at the end of training and 56.8% robust test accuracy at the best checkpoint.

2.3.6 Exploring alternative methods to prevent robust overfitting

We also 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 CIFAR-10 using classical and modern regularization techniques, yet ultimately find that no technique performs as well in isolation as early stopping, as shown in Table 2.10. Unless otherwise stated, we begin each experiment with the standard setup for ℓ_∞ PGD-based adversarial

Table 2.10: Performance of adversarially robust training over a variety of regularization techniques for PGD-based adversarial training on CIFAR-10 for ℓ_∞ with radius $8/255$.

	PGD accuracy (%)			Standard accuracy (%)		
Regularization method	Final	Best	Diff	Final	Best	Diff
Early stopping w/ val	**53.1**	53.3	**−0.2**	81.8	81.8	0.0
ℓ_1 regularization	47.0 ± 0.39	51.4	−4.4	84.1 ± 0.13	84.6	−0.5
ℓ_2 regularization	48.6 ± 0.73	53.6	−5.0	84.3 ± 0.21	85.1	−0.8
Cutout	51.2 ± 0.79	53.3	−2.1	83.2 ± 0.21	83.6	−0.4
Mixup	50.9 ± 1.32	53.7	−2.8	76.7 ± 3.04	81.0	−4.3
Semi-supervised	52.9	59.8	−6.9	77.0 ± 3.82	82.8	−5.8

training with a 10-step adversary with step size $2/255$ using a PreActResNet18 [34].

Explicit 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.

We explore the effects of using ℓ_1 and ℓ_2 regularization when training robust networks on robust overfitting, and sweep across a range of hyperparameter values. [15]. Although explicit regularization does improve the performance to some degree, on its own, it is still not as effective as early stopping, with the best explicit regularizer achieving 44.8% robust test accuracy with ℓ_2 regularization and parameter $\lambda = 5 \cdot 10^{-3}$. Additionally, neither of these regularization techniques can completely remove the detrimental effects of robust overfitting without drastically over-regularizing the model. We include additional details about each regularization method below.

ℓ_1 regularization Figure 2.21 shows the training and test performance of models using various degrees of ℓ_1 regularization. We perform a search over regularization parameters $\lambda = \{5 \cdot 10^{-6}, 5 \cdot 10^{-5}, 5 \cdot 10^{-4}, 5 \cdot 10^{-3}\}$, and find 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 2.22 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

[15] 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.

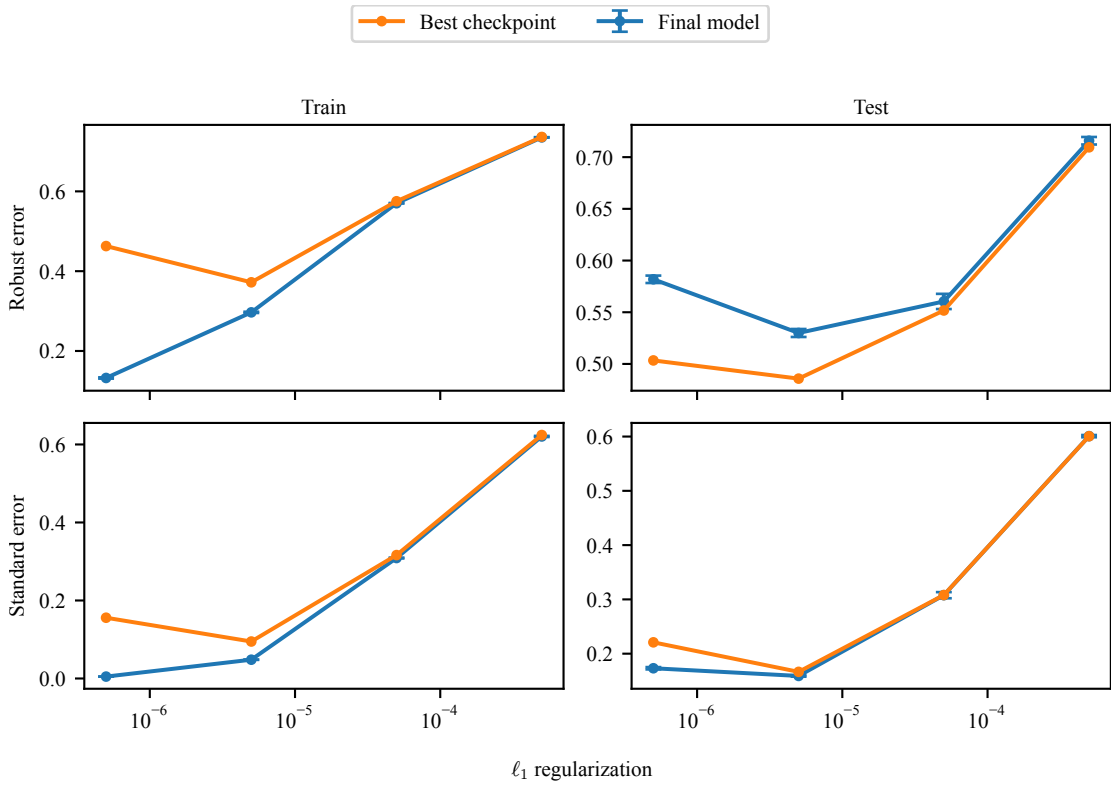


Figure 2.21: Standard and robust performance on the train and test set using varying degrees of ℓ_1 regularization.

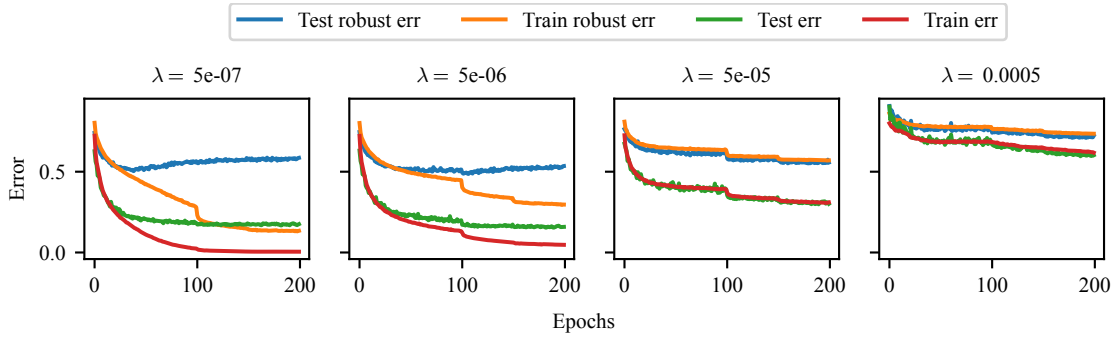


Figure 2.22: Learning curves for adversarial training using ℓ_1 regularization.

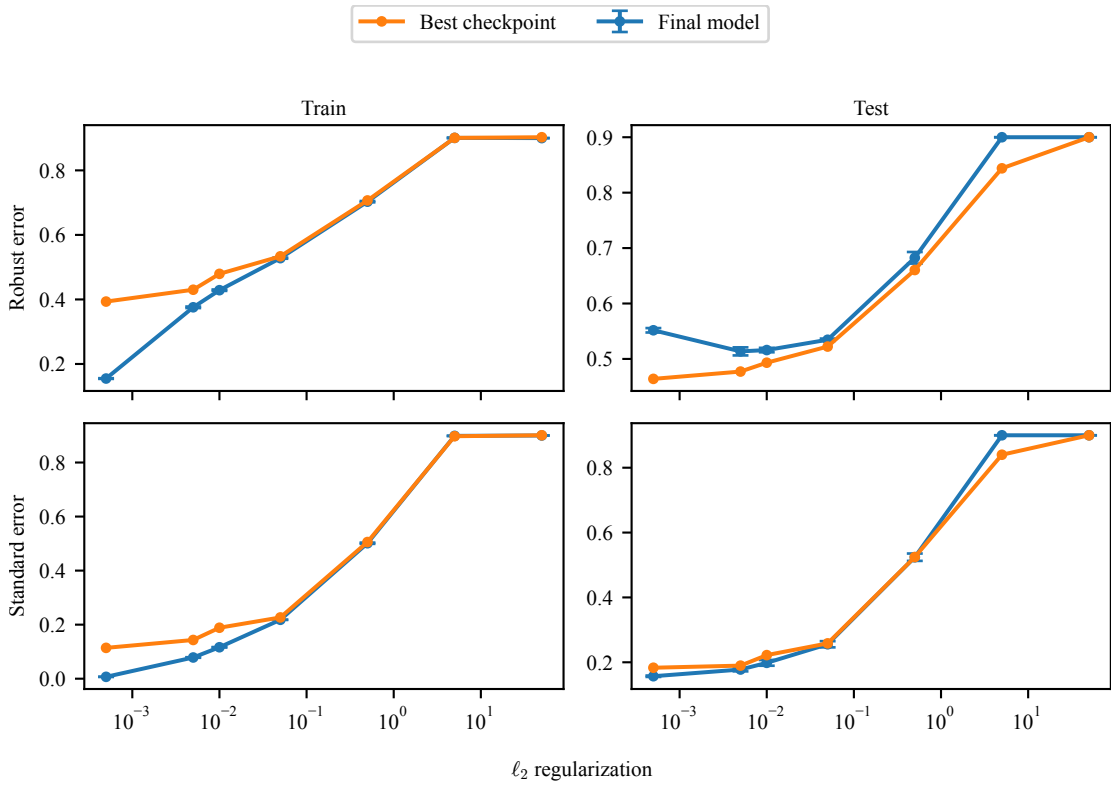


Figure 2.23: Standard and robust performance on the train and test set using varying degrees of ℓ_2 regularization.

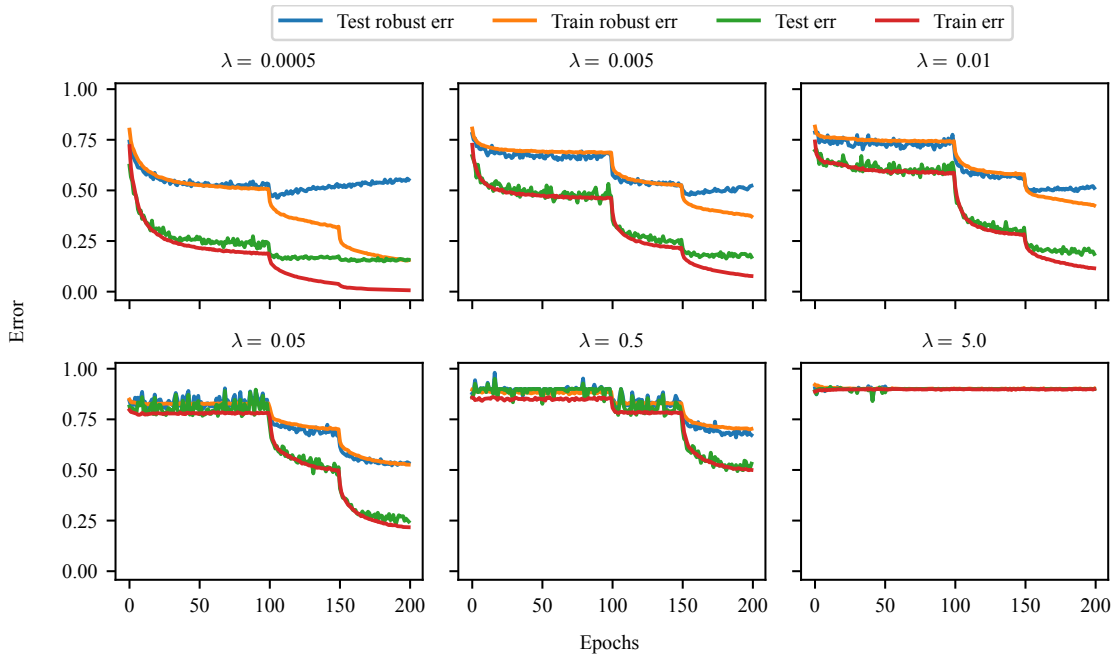


Figure 2.24: Learning curves for adversarial training using ℓ_2 regularization.

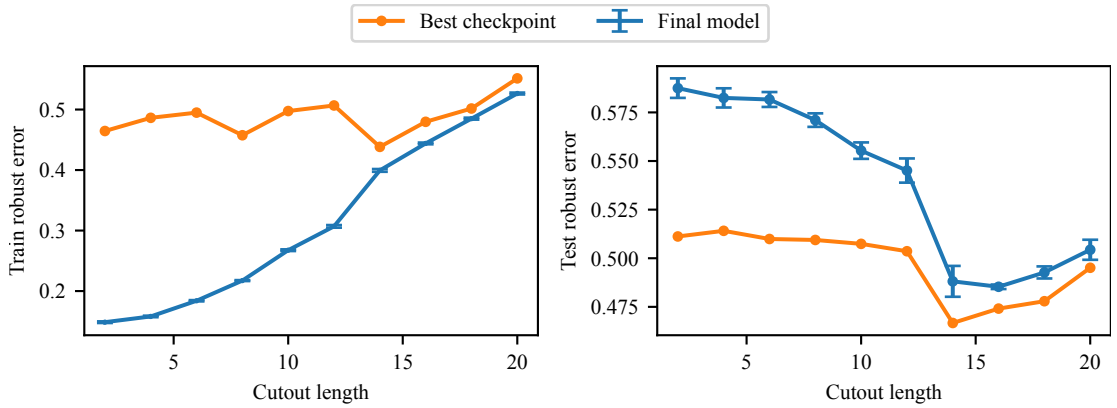


Figure 2.25: Standard and robust performance on the train and test set for varying cutout patch lengths.

options, to the extent that there is no generalization gap and the training and test curves actually appear to match.

ℓ_2 regularization Figure 2.23 shows the training and test performance of models using various degrees of ℓ_2 regularization. We perform 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 used value for weight decay in deep learning. We find that only the smallest choices for λ result in robust overfitting (e.g. $\lambda \leq 0.1$). However, inspecting the corresponding learning curves in Figure 2.24 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 accuracy.

Data augmentation for deep learning

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 to more recent approaches leveraging unlabeled data for semi-supervised learning, and some work has argued that robust deep learning requires more data than standard deep learning [66]. We scan a range of hyperparameters for both the cutout [19] and mixup [92] data augmentation techniques, 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, as seen in Figures 2.25 for cutout and Figures 2.27 for mixup. When trained to convergence, neither cutout nor mixup is as effective as early stopping.

Lastly, we additionally consider a semi-supervised data augmentation technique [1, 14, 88] which uses a standard classifier to label unlabeled data for use in robust training. Although there is a large gap between best and final robust performance, we find that this is primarily driven by high variance in the robust test accuracy during training rather than from robust overfitting, even when the model has converged. Due to this variance, the final model's average robust

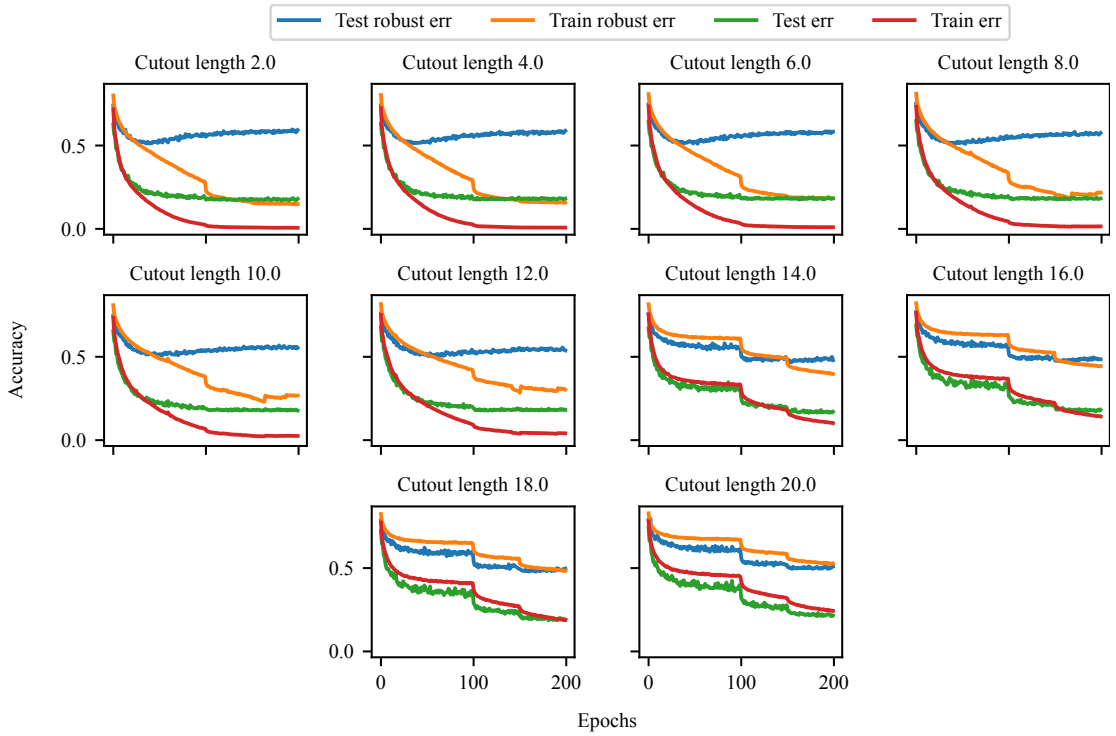


Figure 2.26: Learning curves for adversarial training using cutout data augmentation with different cutout patch lengths.

performance of 52.9% robust test accuracy 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, resulting in 59.8% robust test accuracy.[16] We include more details about each data augmentation technique below.

Cutout To analyze the effect of cutout on generalization, we range the cutout hyperparameter of patch length from 2 to 20. Figure 2.25 shows the training and test performance of models using varying choices of patch lengths. Additionally, for each hyperparameter choice, we plot the resulting learning curves in Figure 2.26. We find the optimal length of cutout patches to be 14, which on its 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.

Mixup When training using mixup, we vary the hyperparameter α from 0.2 to 2.0. The training and test performance of models using varying degrees of mixup can be found in Figure 2.27. The

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

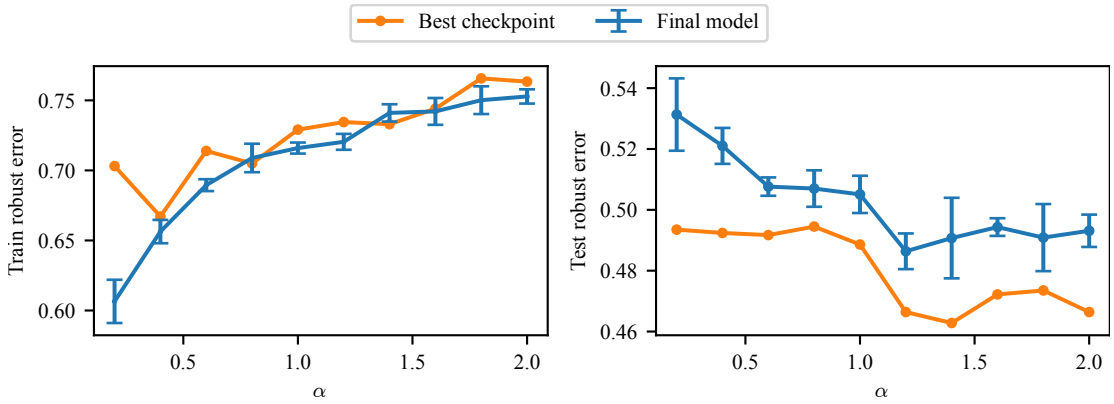


Figure 2.27: Standard and robust performance on the train and test set for varying degrees of mixup.

resulting learning curves for each choice of α can be found in Figure 2.28. 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 accuracy to steadily increase over training, indicating that mixup does stop robust overfitting to some degree (but does not obtain significantly better performance).

Semi-supervised learning For semi-supervised training, we use a batch size of 128 with equal parts labeled CIFAR-10 data and pseudo-labeled TinyImages data, as recommended by Carmon et al. [14]. Each epoch of training is now equivalent in computation to two epochs of standard adversarial training. Note that the PreActResNet18 is a smaller architecture than used by Carmon et al. [14], and so in our reproduction, the best checkpoint which achieves 59.8% accuracy is about 2% lower than 61.5%, which is what Carmon et al. [14] 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% higher accuracy than a PreActResNet18.

We observe that the semi-supervised approach does not exhibit severe robust overfitting, as the smoothed learning curves tend to be somewhat relatively flat and don't show significant decreases in robust test accuracy. 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 accuracy even when training accuracy is relatively flat and has converged. As a result, it is critical to still use the best checkpoint even without robust overfitting, in order to avoid the fluctuations in test performance induced by the augmented training data.

2.3.7 Discussion

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

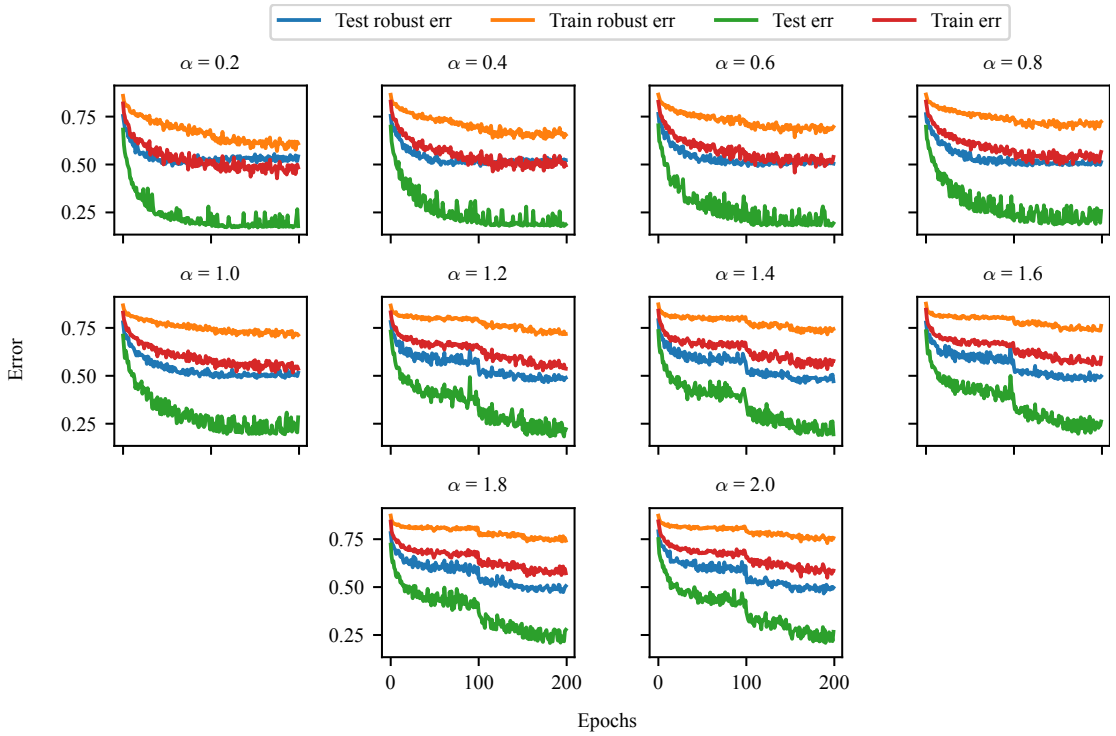


Figure 2.28: Learning curves for adversarial training using mixup with different choices of hyperparameter α.

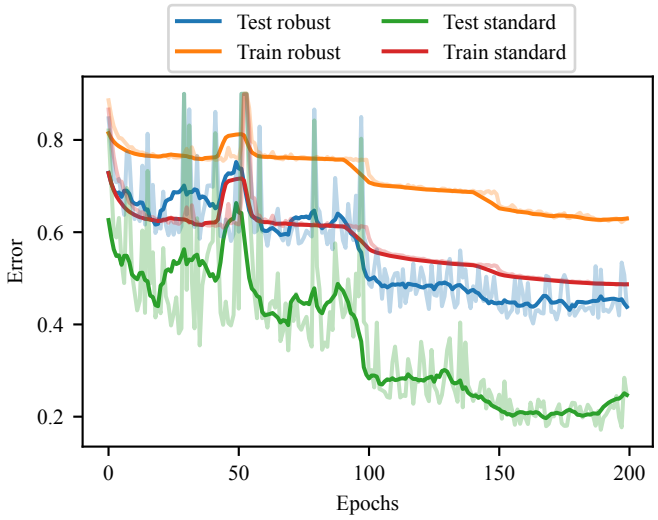


Figure 2.29: Learning curves for robust training with semi-supervised data augmentation, where we do not see a severe case of robust overfitting. When robust training accuracy has converged, there is a significant amount of variance in the robust test accuracy, so the average final model performance 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.

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 stopping.

Especially due to the prevalence of robust overfitting in adversarial training, we note the importance of using validation sets when performing model selection in this regime, and to analyze the learning curves of their models. This work exposes a key difference in generalization properties between standard and robust training, which is not fully explained by either classic statistics or modern deep learning, and re-establishes the competitiveness of the simplest adversarial training baseline.

Chapter 3

Robustness between the worst and average case

Evaluating the robustness of machine learning models can be broadly interpreted as evaluating their performance not just on a test set, but also evaluating the performance relative to some additional (possibly domain-specific) uncertainty or bounds on the problems. Although there are many formal definitions of robustness, most work in this area has focused on two particular settings. We focused one of these settings, adversarial robustness, in Chapter 2 of this thesis. As we discussed in Chapter 2, in the "classical" sense of robustness, we can consider evaluating the classifier in terms of its *worst-case* loss under some perturbation set applied to the inputs, i.e., we could evaluate (via finite sample approximation)

$$\mathbf{E}_{x,y\sim\mathcal{D}}\left[\max_{\delta\in\Delta(x)} \ell(h(x+\delta),y)\right] \tag{3.1}$$

where $\mathcal{D}$ denotes a distribution over x, y pairs, h denotes the hypothesis function, ℓ denotes a loss, and $\Delta(x)$ denotes some (input-dependent) *uncertainty region*. This formulation underlies adversarial examples and also motivates the classical adversarial training approaches discussed in Chapter 2. However, substantial work has also been done in evaluating the setting of robustness to *random* perturbations, i.e., evaluating a classifier via the loss

$$\mathbf{E}_{x,y\sim\mathcal{D}}\left[\mathbf{E}_{\delta\sim\mathcal{P}(x)}[\ell(h(x+\delta),y)]\right] \tag{3.2}$$

where now $\mathcal{P}(x)$ denotes some (again, input-dependent) *distribution* over possible perturbations. This formulation underlies common data augmentation strategies in deep learning, as well as most formulations of "natural" robustness [35] (even if not always written in this formal manner).

Until now, worst-case and average-case robustness have typically been seen as largely separate notions. We believe there is inherent value in generalizing these two notions to place them in a unified framework. The main criticism of worst-case robustness is that it focuses "too much" on the worst case, while the criticism of average-case is that it is "not robust enough". It seems very likely that what people actually want in terms of robustness is precisely something in the middle, between these two extremes.

In this chapter, we first present a natural generalization of robustness objectives between the worst and average case, and show how these objectives can be used for evaluating, and training

for, intermediate robustness. We further extend initial work presented in the beginning of this section, by applying the intermediate robustness objective for evaluating the foundation model CLIP, with the goal of understanding the robustness gains from large-scale pre-training, and the trade-offs that result from fine-tuning on a downstream task. We then present improvements for intermediate-q robustness training, and explore alternative, more efficient, methods of training for intermediate robustness that could perform similarly at a reduced computational cost.

3.1 Intermediate-q robustness

In this section, we advocate for a more fine-grained spectrum of robustness definitions, which naturally interpolates between both these two extremes. In particular, we argue that robustness to random perturbations and worst-case robustness can be naturally interpreted as (functional) ℓ_q norms of the loss function evaluated over the perturbation distribution (which can be a uniform distribution in the case of traditional adversarial loss). In particular, the random setting corresponds to the choice of $q = 1$ and the adversarial setting corresponds to the choice of $q = \infty$. We believe that it is also valuable and informative to consider the performance of classifiers in a wide range in between these two extremes, i.e., the performance of "intermediate-q" robustness. However, evaluating this intermediate-q robustness is non-trivial, owing to the fact that it requires computing a high dimensional integral over the perturbation space. Thus, our main technical contribution in this section is the proposal of a simple approach for evaluating the relevant robustness norms, using the Markov chain Monte Carlo (MCMC) based path sampling technique. Despite their seeming complexity, in this particular case the eventual estimators take the very simple form of a geometric mean computed over samples from an annealed distribution over the perturbation region.

We evaluate our approach on networks trained via standard training, data augmentation, and adversarial training. In all the cases our proposed approach shows a clear trade-off between different levels of robustness that would missed by solely considering just the random or adversarial perturbation setting. Furthermore, we show that our path sampling estimator based on Hamiltonian Monte Carlo (HMC) for intermediate-q robustness is vastly superior to naive estimates produced e.g. by Monte Carlo sampling. We also highlight the ability to *train* networks using these estimators to create classifiers more robust to these intermediate notions of robustness. The code for reproducing experiments in this section is made publicly available[1].

3.1.1 Related work

When considering robustness to more "realistic" perturbations, the focus is often shifted from robustness to the worst-case perturbation within some set to robustness to random perturbations [27, 35, 36, 37, 85]. It was been further studied whether such corruptions can universally improve robustness to real-world distribution shifts, such as geographic location, or camera hardware [38]. While there have been significant research efforts on both sides of worst-case and average-case robustness, there have been much fewer efforts focusing on robustness in between these two

[1] `https://github.com/locuslab/intermediate_robustness`

extremes. While some work such as Meunier et al. [51] could be interpreted as an interpolation between random and adversarial noise, as could tilted empirical minimization [44, 45], a modification of empirical risk minimization allowing for the influence of outliers to be increased or decreased, generally the field faces a divide between studies on adversarial robustness and robustness to random data perturbations.

Because our definition of intermediate-q robustness involves computation of a high dimensional integral over a perturbation distribution, our work can also be related to methods for estimating an intractable partition function of a probability distribution. The problem of computing normalizing constants ties to the problem of computing differences in free energy in physics. Several Monte Carlo sampling-based approaches exist for approximately estimating normalizing constants. Importance sampling is one approach at estimating ratios of normalizing constants that relies on sampling from some alternative density that approximates the target density. An improvement over importance sampling, bridge sampling [7, 49] is a way of further reducing the approximation errors that involves sampling from more than one approximate density to reduce the distance between the target and proposal densities. Taking this one step further, path sampling [28], which originated under the name of thermodynamic integration in physics [59], is an estimator that can be viewed as using infinitely many bridge densities to more accurately approximate the partition function.

Sampling from these approximate densities is done using Markov chain Monte Carlo methods (MCMC). When the distribution is non-differentiable, a method such as random-walk Metropolis Hastings can be used. When the distribution is differentiable, Hamiltonian Monte Carlo methods can be used to improve the sample efficiency the estimator [8, 21, 57]. Hamiltonian Monte Carlo (HMC) is useful due to its avoidance of random walks, and is compatible with sampling in a constrained space through use of reflection [53].

3.1.2 Defining a general robustness objective

We first make the simple observation that there is a natural interpolation between the notions of adversarial robustness and robustness to random perturbations. Specifically, we note that both these notions can be expressed as function ℓ_q norms over the perturbation space[2]. Specifically, we define the following functional norm

Definition 1. *$f : \mathbb{R}^n \to \mathbb{R}$ and density $\mu : \mathbb{R}^n \to \mathbb{R}_+$, $\int \mu(x)dx = 1$, let $\|f\|_{\mu,q}$ be the q-norm of the function under this density*

$$\|f\|_{\mu,q} = \mathbf{E}_{x\sim\mu}\left[|f(x)|^q\right]^{1/q} = \left(\int |f(x)|^q \mu(x)dx\right)^{1/q} \tag{3.3}$$

Then random perturbation loss and adversarial perturbation loss simply correspond to two extremes of this functional norm over the perturbation δ, as formalized by the following proposition.

[2]We should emphasize that this use of ℓ_q norms is entirely orthogonal to the use of ℓ_p balls as perturbation regions, commonly done in adversarial robustness. The ℓ_q norms here can be applied to any perturbation region, as we will highlight.

Proposition 1. *Let δ be a random variable with density μ and consider the expectation*

$$\mathbf{E}_{x,y\sim D}\Big[\|\ell(h(x+\delta),y)\|_{\mu,q}\Big]. \tag{3.4}$$

Then (for a smooth loss ℓ) this corresponds to the expected loss on random samples from μ when $q = 1$, and to the expected adversarial loss over the domain of μ when $q = \infty$.

This proposition follows immediately from the fact that losses are non-negative, and the fact that the ℓ_∞ norm is given by the pointwise maximum of the function, assuming a smooth loss ℓ (i.e. not the zero-one loss). Note that the "traditional" adversarial loss actually arises more specifically when $q = \infty$ and μ is a uniform distribution over some norm ball (unrelated to the norm q). When we have $1 < q < \infty$, we enable a full spectrum of robustness measurements, which we refer to as *intermediate-q robustness*, that evaluates the performance of classifiers in a wide range in between these two extreme cases. Furthermore, we argue that there are naturally appealing properties of these intermediate-q robustness measures: whereas adversarial robustness may overestimate the risk of what effectively amount to "measure zero" regions of the perturbation space, random robustness may likewise fail to take into account smaller but non-negligible regions that *do* contain areas of high loss.

As an example, when considering δ to have a Gaussian distribution, adversarial loss ($q = \infty$) is not meaningful because the worst case perturbation can be arbitrarily far away from x. Meanwhile, random data augmentation with Gaussian noise ($q = 1$) is often insufficient for evaluating robustness because random Gaussian samples are rarely good adversarial examples. Our intermediate-q robustness allows us to consider a middle ground where the model is robust under a certain degree of adversarial noise (stronger than randomly sampled Gaussian noise) while the evaluation is not hindered by very rare events (e.g., an extremely low probability Gaussian sample far away from x).

3.1.3 Path sampling estimation of intermediate-q robustness

Of course, simply writing the loss in this manner is not particularly useful on its own. In most cases, the integral in Equation 3.3 cannot be computed exactly, and so we must resort to numerical approximation methods. Specifically, in order to estimate the q-norm robustness for an arbitrary nonlinear model h, loss ℓ, and (known) density μ, we consider the problem of computing the integral

$$Z := \left(\int \ell(h(x+\delta),y)^q \mu(\delta) d\delta \right)^{(1/q)}. \tag{3.5}$$

One could naively estimate this integral by using Monte Carlo sampling, and sample $\delta^{(1)}$, $\ldots$, $\delta^{(m)}$ randomly from density μ, and approximate the objective as the following,

$$\hat{Z}_{\text{MC}} = \Big(\frac{1}{m} \sum_{i=1}^{m} \ell(h(x+\delta^{(i)}),y)^q \Big)^{(1/q)}. \tag{3.6}$$

However, because the integral in (3.5) will be dominated by values with large loss $\ell(h(x+\delta),y)$, simply using Monte Carlo sampling will be insufficient to approximate this integral well for larger values of q, as random sampling will place too much weight on regions of low loss.

Instead, in this work we argue that it is beneficial to interpret the task at hand as one of evaluating the *partition function* of a particular unnormalized probability density. Specifically, we define an (unnormalized) density over the perturbation δ

$$\tilde{p}(\delta) = \ell(h(x+\delta), y)^q \mu(\delta). \tag{3.7}$$

Then clearly, just from construction, we see that the task of evaluating the partition function (the normalizing constant) of this distribution is exactly the same as that of computing the integral of interest.

The advantage of this perspective on the integral of interest, however, is that we can use a wealth of techniques developed for partition function estimation in order to better estimate this particular integral. Specifically, we argue to use the path sampling [28] approach, a Markov chain Monte Carlo based method, to approximate the desired partition function. In fact, we show that for the precise form of the integral in question, the eventual estimator produced by this method takes on a very simple form: it consists of a geometric mean over samples generated from a certain annealed distribution. This makes the estimator particularly simple to implement and even in some cases to train networks based upon, via standard automatic differentiation toolkits.

We state our main result on the path sampling formulation of the integral of interest via the following theorem.

Theorem 1. *Consider the task of approximately computing the following integral:*

$$Z := \left(\int \ell(h(x+\delta), y)^q \mu(\delta) d\delta \right)^{(1/q)}.$$

Let $\{t^{(1)}, t^{(2)}, \cdots, t^{(m)}\}$ *be* m *scalars corresponding to linearly interpolated values from* 0 *to* q*. For* $i = 1, \ldots, m$*, sample* $\delta^{(i)}$ *from the following unnormalized density*

$$\delta^{(i)} \sim p(\delta | t^{(i)}), \text{ where, } p(\delta|t) \propto \ell(h(x+\delta), y)^t \mu(\delta).$$

Then the following estimator, given by the geometric mean of the resulting samples

$$\hat{Z}_{PS} := \Big(\prod_{i=1}^{m} \ell(h(x+\delta^{(i)}), y) \Big)^{1/m}. \tag{3.8}$$

is a consistent estimator of our integral Z*. Practically, the* m *samples can be drawn using MCMC.*

Proof. For notation completeness, we define the density

$$p(\delta|t) = \frac{1}{z(t)} \tilde{p}(\delta|t) \tag{3.9}$$

where

$$\tilde{p}(\delta|t) = \ell(h(x+\delta), y)^t \mu(\delta). \tag{3.10}$$

and where $z(t)$ denotes the partition function of this distribution

$$z(t) = \int \tilde{p}(\delta|t) d\delta. \tag{3.11}$$

Taking the log of both sides of and differentiating with respect to t, we get:

$$\begin{aligned}
\frac{d}{dt}\log(z(t)) &= \frac{d}{dt}\log\int \tilde{p}(\delta|t)d\delta \\
&= \frac{\int \frac{d}{dt}\tilde{p}(\delta|t)d\delta}{\int \tilde{p}(\delta|t)d\delta} \\
&= \int \frac{\tilde{p}(\delta|t)}{z(t)}\frac{\frac{d}{dt}\tilde{p}(\delta|t)}{\tilde{p}(\delta|t)}d\delta \\
&= \mathbf{E}_{\delta\sim p(\delta|t)}\Big[\frac{d}{dt}\log\tilde{p}(\delta|t)\Big] \\
&= \mathbf{E}_{\delta\sim p(\delta|t)}\Big[\log\ell((h(x+\delta),y))\Big]
\end{aligned} \tag{3.12}$$

where we use the fact that

$$\log\tilde{p}(\delta|t) = t\log\ell((h(x+\delta),y)). \tag{3.13}$$

Then we can integrate (3.12) from 0 to q to get

$$\log\Big[\frac{z(q)}{z(0)}\Big] = \int_0^q \mathbf{E}_{\delta\sim p(\delta|t)}\Big[\log(\ell(h(x+\delta),y))\Big]dt \tag{3.14}$$

Given in our case $z(0) = 1$ and incorporating the exponent of $(1/q)$, we have

$$\log\Big[z(q)^{(1/q)}\Big] = \frac{1}{q}\int_0^q \mathbf{E}_{\delta\sim p(\delta|t)}\Big[\log(\ell(h(x+\delta),y))\Big]dt \tag{3.15}$$

Then the right hand side of Equation 3.15 can be interpreted as the expectation of $\log(\ell(h(x+\delta),y))$ over the joint distribution of (δ,t), where t is a random variable with a uniform distribution in $[0,q]$.

$$\log\Big[z(q)^{(1/q)}\Big] = \mathbf{E}_{t\sim U[0,q]}\Big[\mathbf{E}_{\delta\sim p(\delta|t)}\Big[\log(\ell(h(x+\delta),y))\Big]\Big] \tag{3.16}$$

Finally, sampling $(\delta^{(i)},t^{(i)})$ for $i=1,\ldots,m$ from this joint distribution $p(\delta,t)$ (which we can do by linearly interpolating $t^{(i)}$ between 0 and q and then sampling $\delta^{(i)}$ from $p(\delta|t^{(i)})$), we have the fact that

$$\hat{Z}_{\text{PS}} := \exp\Big(\frac{1}{m}\sum_{i=1}^m \log(\ell(h(x+\delta^{(i)}),y))\Big) = \Big(\prod_{i=1}^m \ell(h(x+\delta^{(i)}),y)\Big)^{1/m}, \tag{3.17}$$

is a consistent estimator of the desired integral. □

The key point of this result is that it allows us to approximate the desired integral just through the ability to *sample* from the distribution $p(\delta|t)$. While this is still a challenging task, sampling from unnormalized probability distributions is a well-studied problem, and we can apply MCMC sampling methods to this task. Further, while the sampling of $(\delta^{(i)},t^{(i)})$ can be done in different ways, we choose to linearly anneal $t^{(i)}$ from 0 to q, and then draw $\delta^{(i)} \sim \tilde{p}(\delta|t)$ using some MCMC sampler. This has the nice feature that it starts with sampling from an "easy" distribution (when $t=0$, the distribution over δ is simply given by μ), and gradually anneals to a more peaked distribution as t increases. The resulting algorithm for evaluating a network using this geometric mean estimator is shown in Algorithm 5.

Algorithm 5 Evaluating the intermediate-q robustness of a neural network function h using path sampling estimation with m MCMC samples with $x, y \sim D$ for some norm q.

Initialize $\delta^{(0)}$ randomly
for $i = 1 \dots m$ **do**
  Let $t^{(i)} := q \cdot \frac{i-1}{m-1}$
  Sample $\delta^{(i)} \sim p(\delta|t^{(i)})$ using MCMC from initial state $\delta^{(i-1)}$
end for
return $\left(\prod_{i=1}^{m} \ell(h(x + \delta^{(i)}), y) \right)^{1/m}$

3.1.4 Using Hamiltonian Monte Carlo to sample from the loss-based distribution

In order to generate the samples for the path sampling estimation (3.17) from the desired distribution $p(\delta, t)$, we use Markov chain Monte Carlo (MCMC) methods to sample δ from the unnormalized distribution $\tilde{p}(\delta|t)$. When the loss is a differentiable function of the perturbation distribution, we can take advantage of gradient-based methods to reduce random walk behavior in MCMC sampling and achieve more efficient sampling. Hamiltonian Monte Carlo (HMC) is one such gradient-based MCMC method that simulates Hamiltonian dynamics to improve sample efficiency in high-dimensional spaces. HMC is based on the Hamiltonian function $H(q, p) = U(q) + K(p)$, where q is a d-dimensional position vector (corresponding to the current sampling state), and p is a (fictitiously introduced) d-dimensional momentum vector. $U(q)$ is the "potential energy" of the system, which in our setting is just the negative log probability density of the distribution we want to sample from, $U(\delta) = -\log(\ell(x + \delta), y)^t \mu(\delta))$, and $K(p) = ||p||^2/(2\sigma^2)$ is the "kinetic energy" of the system. Putting this all together, the Hamiltonian function H is equal to the following:

$$H(\delta, p) = -t \log(\ell(h(x + \delta), y)) + \log \mu(\delta) + \frac{||p||^2}{\sigma^2} \tag{3.18}$$

In order to make use of Hamiltonian dynamics in practice, Hamiltonian's equations must be discretized. The Leapfrog method is one such discretization of Hamiltonian dynamics, using a small stepsize α to discretize time and numerically integrating the system of differential equations as follows:

$$\begin{aligned} p &= p + \alpha t \nabla_\delta \log(\ell(h(x + \delta), y))/2 \\ \delta &= \delta + \alpha p/\sigma^2 \\ p &= p + \alpha t \nabla_\delta \log(\ell(h(x + \delta), y))/2 \end{aligned} \tag{3.19}$$

The HMC algorithm begins by first sampling a new momentum vector $p \sim \mathcal{N}(0, \sigma^2)$, independent of the current state of δ. Then, the Leapfrog method in (3.19) is repeated for L steps to propose a new Markov state (δ', p'). This proposed state is then accepted with probability $\min[1, \exp(-\Delta H)]$, where $\Delta H = H(\delta', p') - H(\delta, p))$. If the state (δ', p') is not accepted, the next state is then set to (δ, p).

Given that in general, our perturbation distribution will likely be constrained in some manner, we need to modify the HMC algorithm so that our proposals remain within the boundaries of the perturbation distribution. Consider the case where our perturbation distribution is the ℓ_∞ norm ball with radius ϵ, so that each element of δ is constrained to be within $-\epsilon$ and ϵ. In order to enforce these constraints while preserving the dynamics, we incorporate what is known as *reflection* in HMC. In this case, after setting $\delta = \delta + \alpha p/\sigma^2$, we check if any $\delta'_i > \epsilon$ or $\delta'_i < -\epsilon$ for $i = 1, \ldots, n$. If so, we negate the corresponding momentum term, p_i, and if $\delta'_i > \epsilon$, we set $\delta_i = 2\epsilon - \delta'_i$, whereas if $\delta'_i < -\epsilon$, we set $\delta'_i = -2\epsilon - \delta'_i$. We repeat this reflection step until δ_i satisfies our constraints. One can think of this behavior as effectively simulating reflecting off a physical boundary.

We note that we can still easily use the path sampling estimator for non-differentiable perturbations by replacing the Hamilton Monte Carlo sampler with any other non-gradient based MCMC sampler, such as a random walk Metropolis. This is roughly similar to random sampling, but with a more subtle weighting on terms that involve higher loss.

3.1.5 Estimating the partition function during training

We further consider the possibility of *training* networks using the estimators we have discussed to achieve better intermediate robustness. However, this becomes less computationally feasible due to the number of samples required to get a good estimate of the objective, due to the $m \times L$ iterations of the path sampling estimator with Hamiltonian Monte Carlo, where L is the number of Leapfrog steps. Additionally, the step size α and number of Leapfrog steps L in HMC require careful tuning. However, for larger values of q, the path sampling estimator is essential to getting accurate estimates of the training objective, as random sampling will be much less likely to come across regions of the perturbation distribution with high loss. Path sampling draws samples from the unnormalized loss distribution using MCMC, and so with increased q, there will be higher weighting on samples that induce higher loss. The Hamiltonian Monte Carlo method has the additional benefit of following the gradient (along with some noise), and so for larger q, even with a small number of iterations, path sampling can have advantages over Monte Carlo sampling during training.

3.1.6 Experiments

In this section, we evaluate the intermediate-q robustness of models trained using standard training, data augmentation, and adversarial training on MNIST and CIFAR-10 towards ℓ_∞-norm ball perturbations. We show that our proposed intermediate-q robustness objective interpolates between the average loss over random perturbations, and worst-case loss over the perturbation set (i.e. the adversarial loss). We compare approximations of the functional q-norm of the cross entropy loss function evaluated over the perturbation distribution computed using Monte Carlo sampling (3.6) to that computed using path sampling (3.8) with Hamiltonian Monte Carlo (Section 3.1.4). We show that path sampling with HMC results in more accurate estimates for larger values of q, more closely approaching the adversarial loss. We additionally show initial results of training according to the intermediate-q robustness objective, which we improve upon in Section

Table 3.1: Evaluations of models according to standard, intermediate robust, and adversarial robust (PGD-100) losses on MNIST considering ℓ_∞-norm ball perturbations.

Train m.	Standard	$\hat{Z}_{\text{MC},1}$	$\hat{Z}_{\text{MC},10}$	$\hat{Z}_{\text{MC},10^2}$	$\hat{Z}_{\text{MC},10^3}$	$\hat{Z}_{\text{PS},1}$	$\hat{Z}_{\text{PS},10}$	$\hat{Z}_{\text{PS},10^2}$	$\hat{Z}_{\text{PS},10^3}$	PGD
Standard	0.028	0.043	0.140	0.251	0.268	0.043	0.160	1.420	4.456	11.649
	±0.001	±0.004	±0.023	±0.041	±0.045	±0.004	±0.028	±0.202	±0.495	±0.893
$\hat{Z}_{\text{MC},1}$	0.034	0.032	0.084	0.143	0.154	0.032	0.088	0.692	2.133	7.363
	±0.001	±0.000	±0.001	±0.002	±0.003	±0.000	±0.002	±0.026	±0.087	±0.435
$\hat{Z}_{\text{MC},10}$	0.027	0.026	0.058	0.098	0.105	0.026	0.058	0.412	1.336	3.722
	±0.001	±0.002	±0.002	±0.002	±0.002	±0.002	±0.001	±0.011	±0.050	±0.231
$\hat{Z}_{\text{MC},10^2}$	0.026	0.025	0.055	0.093	0.099	0.025	0.055	0.388	1.261	3.492
	±0.001	±0.001	±0.001	±0.001	±0.001	±0.001	±0.001	±0.015	±0.069	±0.291
$\hat{Z}_{\text{MC},10^3}$	0.026	0.025	0.055	0.093	0.100	0.025	0.055	0.390	1.268	3.488
	±0.002	±0.002	±0.001	±0.002	±0.001	±0.002	±0.001	±0.013	±0.059	±0.242
$\hat{Z}_{\text{PS},10}$	0.034	0.031	0.075	0.126	0.135	0.031	0.075	0.467	1.307	5.012
	±0.001	±0.001	±0.001	±0.002	±0.003	±0.001	±0.001	±0.014	±0.029	±0.353
$\hat{Z}_{\text{PS},10^2}$	0.030	0.028	0.060	0.099	0.107	0.028	0.058	0.304	0.816	2.613
	±0.003	±0.002	±0.006	±0.010	±0.011	±0.002	±0.006	±0.007	±0.027	±0.193
$\hat{Z}_{\text{PS},10^3}$	**0.024**	**0.024**	**0.047**	0.077	0.083	**0.024**	**0.045**	0.239	0.684	1.646
	±0.001	±0.001	±0.001	±0.001	±0.001	±0.001	±0.001	±0.008	±0.037	±0.095
PGD-50	0.032	0.039	0.051	**0.076**	**0.081**	0.039	0.048	**0.101**	**0.187**	**0.270**
	±0.005	±0.006	±0.007	±0.010	±0.011	±0.006	±0.007	±0.016	±0.040	±0.033

3.3. Lastly, we consider non-differentiable spatial transformations, and evaluate the intermediate-q robustness of models trained using standard training, data augmentation, and intermediate-q robust training on CIFAR-10 towards these transformations.

We refer to the intermediate-q robustness objective estimated via Monte Carlo sampling as $\hat{Z}_{\text{MC},q}$, and the objective estimated via path sampling as $\hat{Z}_{\text{PS},q}$. Each training and evaluation run is performed using a single Quadro RTX 8000 GPU. We report results from multiple training runs using 3 different random seeds.

Robustness over the ℓ_∞-norm ball

In order to easily translate the notion of intermediate-q robustness to that of adversarial robustness, we consider perturbations uniformly distributed within an ℓ_∞-norm ball with radius ϵ.

MNIST On MNIST, we train a set of models using standard training, data augmentation (corresponding to $\hat{Z}_{\text{MC},1}$), and PGD adversarial training. For the model trained with data augmentation, we train on $m = 50$ random perturbations per image. For PGD adversarial training, we use 50 PGD steps.We consider perturbations uniformly distributed within the ℓ_∞-norm ball with radius $\epsilon = 0.3$. Each model is trained using the Adam optimizer for 10 epochs with a starting learning rate of 0.001. We use a convolutional ReLU architecture with two convolutional layers with 32 and 64 channels and kernel sizes of 4×4, which are followed by a fully connected layer with 1024 units.

We compare the intermediate-q test robustness ($\hat{Z}_{\text{MC},q}$ and $\hat{Z}_{\text{PS},q}$) of these models to the standard test loss and the adversarial test loss computed using PGD with 100 steps in Table 3.1. $\hat{Z}_{\text{MC},q}$

Table 3.2: Robust accuracy of models trained on MNIST for perturbations in the ℓ_∞ ball of radius $\epsilon = 0.3$.

Train method	PGD-100 accuracy (%)
Standard	4.69
$\hat{Z}_{\text{MC},1}$	25.21
$\hat{Z}_{\text{MC},10}$	43.54
$\hat{Z}_{\text{MC},10^2}$	47.27
$\hat{Z}_{\text{MC},10^3}$	45.45
$\hat{Z}_{\text{PS},10}$	43.57
$\hat{Z}_{\text{PS},10^2}$	65.11
$\hat{Z}_{\text{PS},10^3}$	69.45
PGD-50	91.55

is computed with $m = 2000$ samples, and $\hat{Z}_{\text{PS},q}$ is computed with $m = 100$ samples, using HMC with $L = 20$ Leapfrog steps. Our evaluations show that the intermediate-q robustness objective ($\hat{Z}_{\text{MC},q}$ and $\hat{Z}_{\text{PS},q}$) naturally interpolates between the average loss over random perturbations ($\hat{Z}_{\text{MC},1}$) and the adversarial loss with increasing values of q. For larger values of q, specifically $q = 10^2$ and $q = 10^3$, the intermediate-q objective approximated via path sampling with HMC ($\hat{Z}_{\text{PS},q}$) is a more accurate estimation of the desired integral than that approximated via Monte Carlo estimation ($\hat{Z}_{\text{MC},q}$). This is inferred based on the fact that the value of $\hat{Z}_{\text{PS},q}$ is higher and more closely approaching the adversarial loss.

We additionally show the potential benefits of training using the intermediate-q robustness objective. We train according to either $\hat{Z}_{\text{MC},q}$ or $\hat{Z}_{\text{PS},q}$ for $q = \{10, 10^2, 10^3\}$, and evaluate the resulting models in Table 3.1. Because computing these estimates of the intermediate-q objective is computationally expensive, during training we use $m = 50$ samples to compute $\hat{Z}_{\text{MC},q}$, and $m = 25$ samples with $L = 2$ Leapfrog steps to compute $\hat{Z}_{\text{PS},q}$. We find that most models trained according to $\hat{Z}_{\text{PS},q}$ outperform those trained using $\hat{Z}_{\text{MC},q}$ for the same value of q. In other words, path sampling-based intermediate robust training results in a model with better intermediate robustness. This suggests that even with a reasonably small number of samples, path sampling can still result in a better estimate of the objective than Monte Carlo estimation.

We also include the test robust accuracy of each trained model (from the PGD-100 evaluation) in Table 3.2, which shows that training using these estimates with increasingly large q does indeed improve worst-case robust performance. We do not include standard accuracy here, because on MNIST, an adversarially trained model does not lose much in terms of standard performance, however on more challenging datasets, we would expect to see a similar decrease in standard accuracy as we increase q, and thus one could choose a value of q based on the desired trade-off between standard and robust accuracy.

CIFAR-10 We also evaluate models trained with standard training, data augmentation ($\hat{Z}_{\text{MC},1}$), and PGD adversarial training on CIFAR-10. For the model trained with data augmentation, we train on $m = 10$ random perturbations per image. For PGD experiments, we set the step size

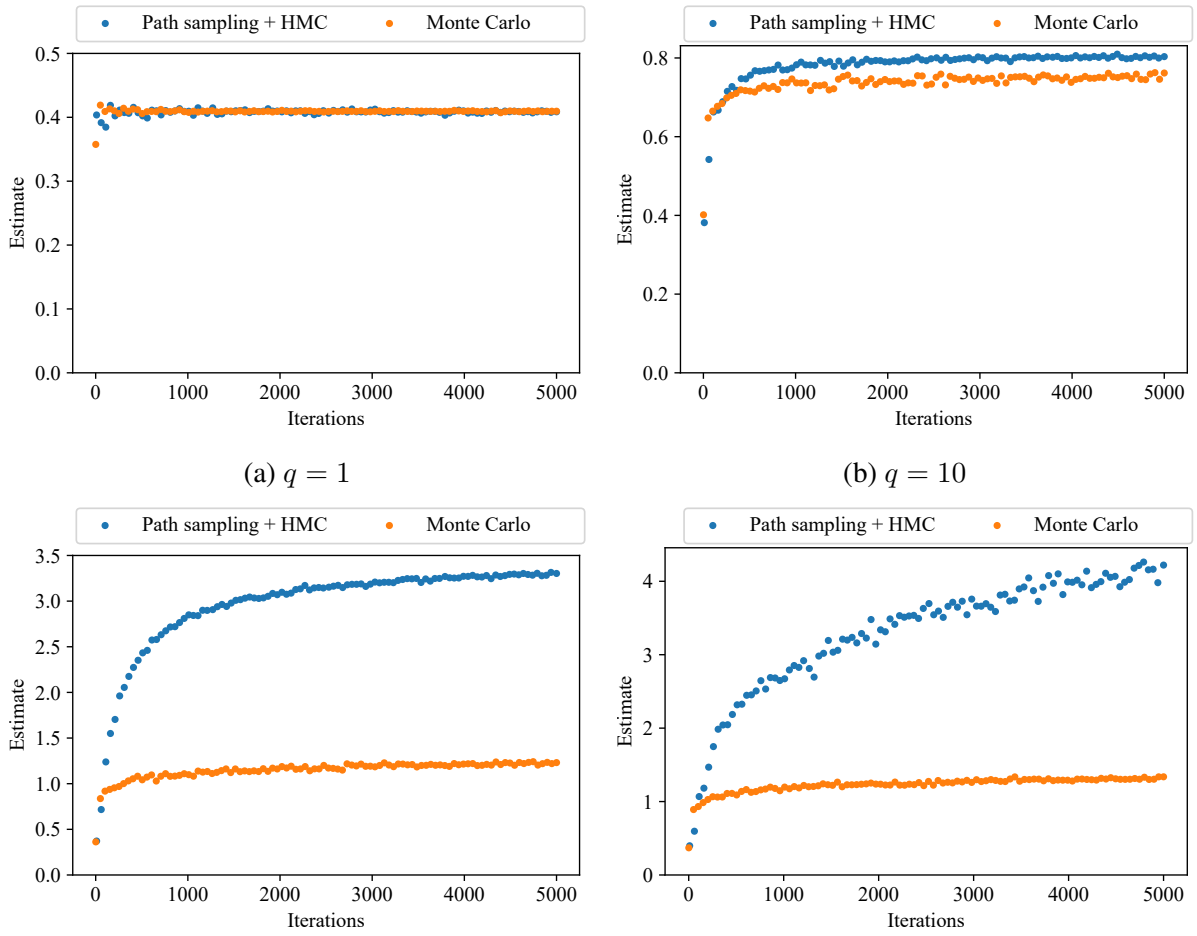
Table 3.3: Evaluations of models according to standard, intermediate robust, and adversarial robust (PGD-50) losses on CIFAR-10 considering ℓ_∞-norm ball perturbations.

Train m.	Standard	$\hat{Z}_{\text{MC},1}$	$\hat{Z}_{\text{MC},10}$	$\hat{Z}_{\text{MC},10^2}$	$\hat{Z}_{\text{MC},10^3}$	$\hat{Z}_{\text{PS},1}$	$\hat{Z}_{\text{PS},10}$	$\hat{Z}_{\text{PS},10^2}$	$\hat{Z}_{\text{PS},10^3}$	PGD
Standard	**0.382**	0.453	0.787	1.153	1.216	0.453	0.841	2.718	4.991	18.142
	±0.002	±0.006	±0.023	±0.037	±0.039	±0.006	±0.026	±0.090	±0.117	±0.448
$\hat{Z}_{\text{MC},1}$	0.400	0.405	0.532	0.717	0.756	0.405	0.546	1.490	3.140	14.240
	±0.005	±0.004	±0.006	±0.009	±0.010	±0.004	±0.006	±0.015	±0.017	±0.011
$\hat{Z}_{\text{MC},10}$	0.393	**0.398**	0.468	0.598	0.630	**0.398**	0.471	1.037	2.365	12.051
	±0.002	±0.003	±0.004	±0.005	±0.005	±0.003	±0.004	±0.013	±0.019	±0.036
$\hat{Z}_{\text{MC},10^2}$	0.399	0.402	**0.466**	**0.589**	**0.620**	0.402	**0.468**	0.980	2.269	12.084
	±0.003	±0.003	±0.003	±0.003	±0.004	±0.003	±0.003	±0.006	±0.020	±0.135
$\hat{Z}_{\text{MC},10^3}$	0.399	0.405	0.469	0.593	0.625	0.405	0.471	0.993	2.302	12.173
	±0.003	±0.003	±0.002	±0.003	±0.004	±0.003	±0.002	±0.005	±0.009	±0.128
PGD-10	0.731	0.733	0.734	0.743	0.761	0.733	0.734	**0.743**	**0.796**	**1.411**
	±0.005	±0.005	±0.005	±0.005	±0.005	±0.005	±0.005	±0.005	±0.003	±0.009

such that $\alpha = 2.5 \cdot \epsilon/m$, where m is the number of PGD steps, and we use early stopping for adversarial training based on the adversarial validation loss. For PGD adversarial training, we use 10 PGD steps. For experiments using Hamiltonian Monte Carlo, we use $\sigma = 0.1$, and set the step size such that $\alpha = \rho \cdot \sigma^2/L$. For $q = 1$ and $q = 10$, we set $\rho = 0.6$, for $q = 100$ we set $\rho = 0.4$, and for $q = 1000$, we set $\rho = 0.2$. Each model is trained with a PreActResNet18 architecture using the SGD optimizer for 200 epochs with a starting learning rate of 0.1 that is divided by 10 halfway and two thirds of the way through training, Nesterov momentum of 0.9, and weight decay 0.0005. We do not use random flip/crop data augmentation that is typically used for training CIFAR-10.

We evaluate the test performance of these models according to the standard cross entropy loss, intermediate-q robustness objectives, and the adversarial loss, computed using PGD with 50 steps. The results are shown in Table 3.3. $\hat{Z}_{\text{MC},q}$ is computed with $m = 500$ samples, and $\hat{Z}_{\text{PS},q}$ is computed with $m = 50$ samples, using HMC with $L = 10$ Leapfrog steps. The evaluation results on CIFAR-10 are consistent with those on MNIST. The intermediate-q robustness objective interpolates between the average and worst-case robustness objectives, and for larger values of q, the path sampling estimate of the intermediate-q robustness objective is a more accurate approximation as compared to the Monte Carlo estimate. The advantage of the path sampling estimator over the Monte Carlo estimator is further illustrated in Figure 3.1, where we show the convergence of their corresponding estimates of the functional q-norm of the loss over this same perturbation distribution given increasing number of samples. Specifically, we compare the approximated values $\hat{Z}_{\text{MC},q}$ and $\hat{Z}_{\text{PS},q}$ computed using the same number of "iterations", which corresponds to m for the Monte Carlo estimator, and $m \times L$ for the path sampling estimator, where $L = 10$. While for $q = 1$ the estimates converge to the same value, for $q = 100$, the estimates quickly diverge, with path sampling producing a much higher, more accurate estimate, approaching the adversarial loss.

We additionally train on CIFAR-10 using the intermediate-q objective, as shown in Table 3.3. However, the computational complexity given the number of samples required to get a reasonable estimate makes training more challenging compared to the smaller MNIST dataset. While for



(a) $q = 1$ (b) $q = 10$

(c) $q = 100$ (d) $q = 1000$

Figure 3.1: Convergence of path sampling and Monte Carlo estimations of the objective $\hat{Z}_q$ for different values of q on a single mini-batch of CIFAR-10 test data given an ℓ_∞-norm ball perturbation distribution.

models trained with $\hat{Z}_{\text{MC},q}$, we do see an improvement in intermediate-q robust performance from training with $q = 10$ vs. $q = 1$, training using values of q larger than 10 does not provide much, if any, additional benefit. Given the same number of iterations, we were not able to improve upon these results by training using the path sampling estimator, using $m = 5$ samples and $L = 2$ Leapfrog steps, suggesting that this number of samples is not large enough to get a good estimate of the objective. However, we show later, in Section 3.3, how using more samples and Leapfrog steps can improve training in this setting.

Robustness over (non-differentiable) spatial transformations

In order to show that the path sampling estimator can naturally be extended to non-differentiable perturbations, we consider a perturbation set consisting of parameterizations of spatial transformations on CIFAR-10. The parameters of the spatial transformations include horizontal flips,

Table 3.4: Evaluations of models according to standard, intermediate robust, and worst-case robust losses on CIFAR-10 considering spatial transformations

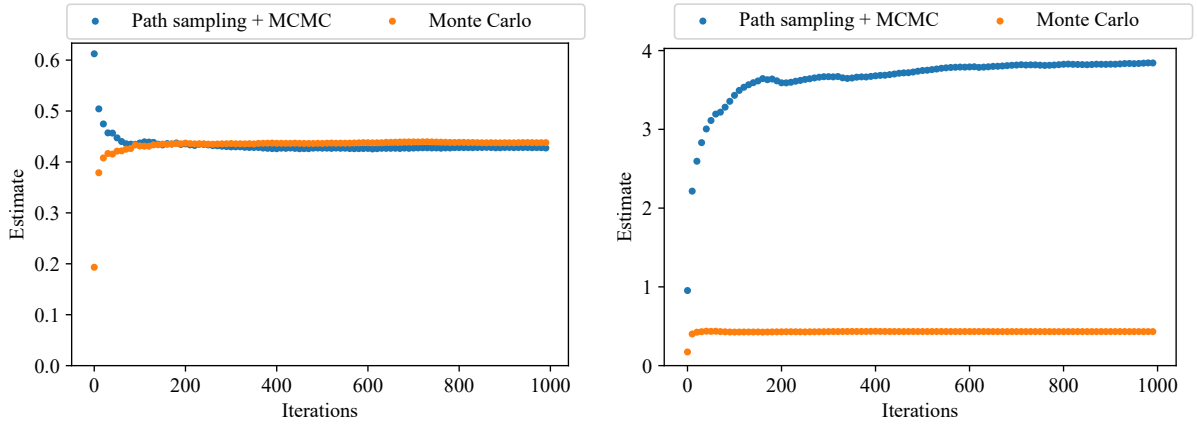
Train m.	Standard	$\hat{Z}_{\text{MC},1}$	$\hat{Z}_{\text{MC},10}$	$\hat{Z}_{\text{MC},10^2}$	$\hat{Z}_{\text{MC},10^3}$	$\hat{Z}_{\text{PS},1}$	$\hat{Z}_{\text{PS},10}$	$\hat{Z}_{\text{PS},10^2}$	$\hat{Z}_{\text{PS},10^3}$	Adv.
Standard	0.186	0.450	2.268	3.687	3.865	0.444	2.450	4.636	4.889	5.625
	±0.007	±0.017	±0.067	±0.095	±0.114	±0.017	±0.068	±0.113	±0.111	±0.134
$\hat{Z}_{\text{MC},1}$	**0.154**	**0.191**	**0.800**	**1.246**	**1.300**	**0.186**	**0.879**	1.615	1.711	2.021
	±0.011	±0.002	±0.026	±0.024	±0.023	±0.004	±0.018	±0.039	±0.018	±0.036
$\hat{Z}_{\text{MC},10}$	0.963	1.019	1.086	1.348	1.423	1.015	1.078	**1.416**	**1.502**	**1.596**
	±0.014	±0.012	±0.010	±0.027	±0.029	±0.012	±0.010	±0.033	±0.078	±0.038
$\hat{Z}_{\text{MC},10^2}$	2.014	2.131	2.131	2.137	2.164	2.130	2.131	2.136	2.171	2.190
	±0.001	±0.002	±0.001	±0.002	±0.001	±0.002	±0.002	±0.002	±0.001	±0.006

rotations between -10 to 10 degrees, scaling factors between 0.9 to 1.1, and cropping between 0 to 4 pixels horizontally and vertically. Because the applied spatial perturbations are non-differentiable, in place of Hamiltonian Monte Carlo we use Gaussian random walk Metropolis Hastings to sample from the unnormalized loss distribution. Specifically, we use the following proposal distributions: 1) perform a horizontal flip with probability 0.5; 2) scale the image by resizing by a factor of $r \sim \mathcal{N}(0, 0.5)$; 3) rotate d degrees with $d \sim \mathcal{N}(0, 5)$; 4) crop x or y in the horizontal or vertical direction, each drawn i.i.d. $\sim \mathcal{N}(0, 2)$ and then rounded to the nearest integer value. Additionally, at each value of the annealed t, we perform 20 burn-in steps and only keep the last value for estimation purposes.

Unless otherwise specified, the training setup is the same as described in Section 3.1.6 for CIFAR-10. We train a standard model on CIFAR-10 for 50 epochs and train according to $\hat{Z}_{\text{MC},q}$ (for $q \in \{1, 10, 100\}$) for 200 epochs. We use a learning rate schedule that linearly increases from 0 to the maximum value of 0.1 for the first two fifths of training epochs, and then linearly decreases to 0. When training using $\hat{Z}_{\text{MC},q}$, we do not perform random flip/crop data augmentation. For evaluation, estimates $\hat{Z}_{\text{MC},q}$ and $\hat{Z}_{\text{PS},q}$ are computed with $m = 500$ samples. For training, For training, we use $m = 10$ samples to compute capture the entire spectrum of robustness. Due to the computational complexity, we only evaluate on the first 1000 CIFAR-10 test examples. The adversarial loss is approximated by averaging the maximum loss value encountered for each example during the Metropolis Hastings sampling process.

For each model, we compare the estimates of the functional q-norm of the loss over this perturbation distribution generated by the Monte Carlo estimator to those generated by the path sampling estimator using Gaussian random-walk Metropolis sampling. The results in Table 3.4 show that with larger q, the path sampling estimator produces better (higher) estimates than random sampling. For $q = 10^3$, the path sampling estimation approaches the adversarial loss over the transformation space, suggesting that this estimator, even without the advantage of Hamiltonian Monte Carlo, can effectively capture the entire spectrum of robustness. The convergence plots for estimating the objective for $q = 1$ and $q = 10$ are shown in Figure 3.2.

As with the case of the ℓ_∞-norm ball perturbation distribution on CIFAR-10, a larger number of samples are needed to get good estimates of the desired integral, making training using these estimates challenging, as shown by the lack of improvement in robust performance for larger values of q for models trained according to $\hat{Z}_{\text{MC},q}$.



(a) $q = 1$ (b) $q = 10$

Figure 3.2: Convergence of path sampling and Monte Carlo estimators on CIFAR-10 (spatial transformations).

3.1.7 Discussion

In this section, we proposed a definition of intermediate-q robustness that smooths the gap between robustness to random perturbations and adversarial robustness by generalizing these notions of robustness as functional ℓ_q norms of the loss function over the perturbation distribution. In order to evaluate intermediate-q robustness in practice, we introduced an approach for approximating the high dimensional integral over the perturbation distribution that uses path sampling, an effective estimator based on MCMC sampling. We showed that across different datasets, models trained on different training objectives, and different perturbation distributions (both differentiable and non-differentiable cases), our path sampling approach produces much better estimates of the integral than simple Monte Carlo sampling. Additionally, we illustrated the benefit of using the gradient-based Hamiltonian Monte Carlo method as the MCMC sampler when the loss is differentiable with respect to the perturbation distribution. We further illustrated that the intermediate-q objective is differentiable, and can be used for training. In the next section, we'll discuss how we can further improve training for intermediate robustness, and we consider alternatives to training using the intermediate-q robustness objective.

While we showed the utility of our proposed intermediate robustness metric for evaluating models on a range of robustness values, we note that it is not entirely clear in practice how to choose a value of q to evaluate on. Future work could study how to make this a more interpretable parameter, and additionally how to better evaluate these robustness levels in terms of accuracy.

3.2 Evaluating the robustness of CLIP

In this section, we consider evaluating the robustness of foundation models [9] using the intermediate-q robustness metric. Increasingly, these large models trained on vast amounts of diverse data are made publicly available, such as OpenAI's CLIP (Contrastive Language–Image Pre-training)

model [61]. CLIP models are trained on 400 million image/text pairs, with the task of predicting which text is paired with a given image. The zero-shot performance of these models on a variety of tasks is often on par with state-of-the-art models trained using standard supervised learning. CLIP can also be easily fine-tuned for a given task, which is appealing due to the reduction in training time as compared to training a model from scratch, and the diverse pre-training combined with fine-tuning can often improve the final performance of the model on a given dataset as compared to standard supervised learning. Zero-shot CLIP models have been deemed "robust" in the sense that these models have been shown to generalize well across tasks and across distribution shifts. However, while zero-shot CLIP has been shown to robust in this sense of being able to generalize to different tasks and distributions, CLIP is non-robust in the sense that it is still susceptible to adversarial examples, resulting in 0% adversarial accuracy [26].

We evaluate CLIP in the context of intermediate robustness, between the worst and average-case. The adversarial robustness objective is not entirely useful for comparing non-adversarially trained models, given the fact that even models pre-trained on vast amounts of data are still susceptible to small additive adversarial perturbations. However, comparing models using the intermediate robustness objectives we have discussed in this chapter could give us a better idea of how pre-training on vast, diverse sets of data (*without* any specific form of robust training) affects robustness beyond the average-case. While it has been shown that adversarial training requires substantially more samples than standard training to achieve adversarial robustness [66], the effects of sample complexity and dataset diversity on a (non-adversarially) pre-trained model's test-time robustness remain an open question.

We further study how the robustness of CLIP is affected by fine-tuning a linear classifier on top of the model for a particular task. It has been shown that fine-tuning a model on a downstream task can degrade the robustness of the model to distribution shifts [42]. To understand how this relates to robustness to corruptions, we study the effect of fine-tuning on robustness when considering input perturbations rather than distribution shifts. While several works have tried incorporating adversarial training methods in the pre-training phase, and studied how adversarial robustness transfers during fine-tuning [15, 24], in this work, we study how the intermediate robustness of the model is affected by the large-scale pre-training process and the fine-tuning process, without any adversarial training in either step.

3.2.1 Intermediate-q robustness of CLIP

We evaluate zero-shot and fine-tuned CLIP models, along with a model trained from scratch with no extra data, on the CIFAR-10 dataset. For simplicity, and ease of training a model from scratch, we evaluate the CLIP model with the ResNet50 architecture. We evaluate the robustness of each of these models towards ℓ_∞-norm ball perturbations and Gaussian perturbations, using the intermediate-q robustness objective as well as the quantile accuracy metric from Robey et al. [65]. We find that while in terms of quantile accuracy, the zero-shot CLIP model performs worse than the fine-tuned and standard (trained from scratch) models, our intermediate-q robustness objective shows a different result. Ultimately, we find that as we increase q, effectively increasing the evaluation robustness level, at a certain point the zero-shot CLIP model overtakes the fine-tuned and standard models, having *better* intermediate robustness. This shows that first, our intermediate-q robustness objective shows properties of models that other objectives do not, and

Table 3.5: Evaluations of intermediate-q robustness on CIFAR-10 towards perturbations uniformly distributed within the ℓ_∞-norm ball with $\epsilon = 8/255$.

Train method	Standard	$\hat{Z}_{\text{MC},1}$	$\hat{Z}_{\text{PS},10}$	$\hat{Z}_{\text{PS},10^2}$	$\hat{Z}_{\text{PS},10^3}$	PGD-50
Standard	**0.175**	0.304	0.854	4.999	14.686	50.791
CLIP zero-shot	1.030	1.144	1.384	**2.602**	**4.995**	**14.130**
CLIP fine-tuned	0.273	**0.289**	**0.775**	4.275	11.135	45.752

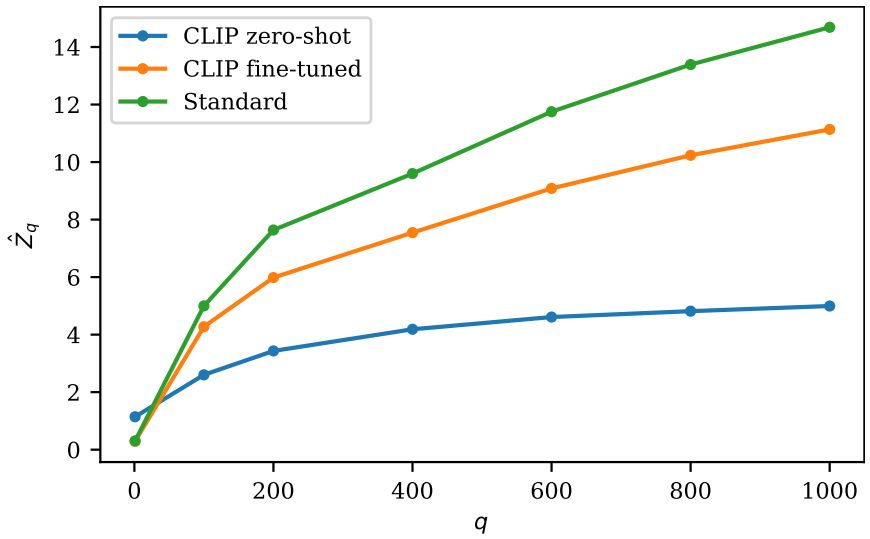


Figure 3.3: Estimated intermediate-q robust loss ($\hat{Z}_q$) over perturbations uniformly distributed within the ℓ_∞-norm ball with $\epsilon = 8.255$, evaluated at different robustness levels (q).

second, that zero-shot CLIP does exhibit some more than average robustness to noise perturbations.

3.2.2 Evaluations on CIFAR-10

Implementation details For exact comparison purposes, we train a ResNet50 from scratch on CIFAR-10, using the same ResNet50 implementation used for CLIP[3]. Note that this particular architecture takes inputs of size 224×224, and so we resize the 32×32 CIFAR-10 images to 224×224 during training. For the fine-tuned model, we append a linear head to the CLIP ResNet50 model and fine-tune (updating all model parameters) on CIFAR-10 for 100 epochs using a starting learning rate of 0.005. For the model trained from scratch, we train for 200 epochs using a starting learning rate of 0.1. For both models, we train using SGD, with mixed precision, weight decay $5 \cdot 10^{-4}$, momentum 0.9, and use a cosine decay learning rate schedule. As in the previous section, we use $m = 100$ perturbation samples to evaluate each metric, and $L = 10$ Leapfrog steps for the path sampling-based objective. We apply the perturbations at the 32×32 resolution and then resize the images to 224×224.

[3] `https://github.com/openai/CLIP`

Table 3.6: Accuracy (%) on ℓ_∞ perturbations with $\epsilon = 8/255$ on CIFAR-10.

			ProbAcc(ρ)			
Method	Standard	Rand.	0.1	0.05	0.01	PGD-50
Standard	**95.24**	91.96	87.72	86.00	81.58	0.00
CLIP zero-shot	67.09	61.10	42.70	37.35	27.08	0.00
CLIP fine-tuned	95.22	**93.36**	**88.44**	**86.92**	**82.24**	0.00

Table 3.7: Evaluations of intermediate-q robustness on CIFAR-10 towards Gaussian perturbations with $\sigma = 0.1$

Train method	Standard	$\hat{Z}_{\text{MC},1}$	$\hat{Z}_{\text{PS},10}$	$\hat{Z}_{\text{PS},10^2}$	$\hat{Z}_{\text{PS},10^3}$	PGD-50
Standard	**0.175**	3.816	6.159	16.723	39.354	39.838
CLIP zero-shot	1.030	**2.277**	**2.337**	**3.658**	**6.170**	**13.660**
CLIP fine-tuned	0.273	4.581	6.242	15.304	32.444	34.393

Robustness to ℓ_∞-norm ball perturbations We first consider robustness towards perturbations uniformly distributed within the ℓ_∞-norm ball with $\epsilon = 128/255$. We compare intermediate-q robustness in Table 3.5 for different values of q. While for $q = 1$ and $q = 10$, the fine-tuned CLIP model has the lowest intermediate robust loss, at higher robustness levels beyond $q = 10$, we find the zero-shot CLIP model has significantly better intermediate robustness. We see that the fine-tuned model has better intermediate robustness than the standard model (trained from scratch) for large values of q, suggesting that the model still retains some of its robust properties from the pre-training phase. These results can be visualized for additional values of q in Figure 3.3. We find that these robustness results additionally hold for the adversarial loss, computed using PGD, with the zero-shot model having the lowest loss value. However, we show that these findings are not reflected by the quantile accuracy metric in Table 3.6, where we observe the fine-tuned CLIP model has the highest accuracy on all values of ρ. Consistent with findings from the previous section, the different results suggest that the quantile accuracy metric considers a less strict notion of robustness, not taking low-probability, large loss-inducing samples as strongly into account.

Robustness to Gaussian perturbations We also compare the robustness of these models to isotropic Gaussian perturbations with $\sigma = 0.1$. We compare intermediate-q robustness in Table 3.7 for different values of q, and plot additional values of q in 3.4. For this more challenging perturbation distribution, we find that the zero-shot CLIP model has the best intermediate-q robustness for all evaluated values of q. Interestingly, this is not the case for accuracy on random Gaussian perturbations, shown in Table 3.8. Rather, for the quantile accuracy metric, the standard model (trained from scratch) has the highest accuracy for each value of ρ. We hypothesize that this could be due to the fact that the quantile accuracy does not take overconfident, incorrect predictions into account.

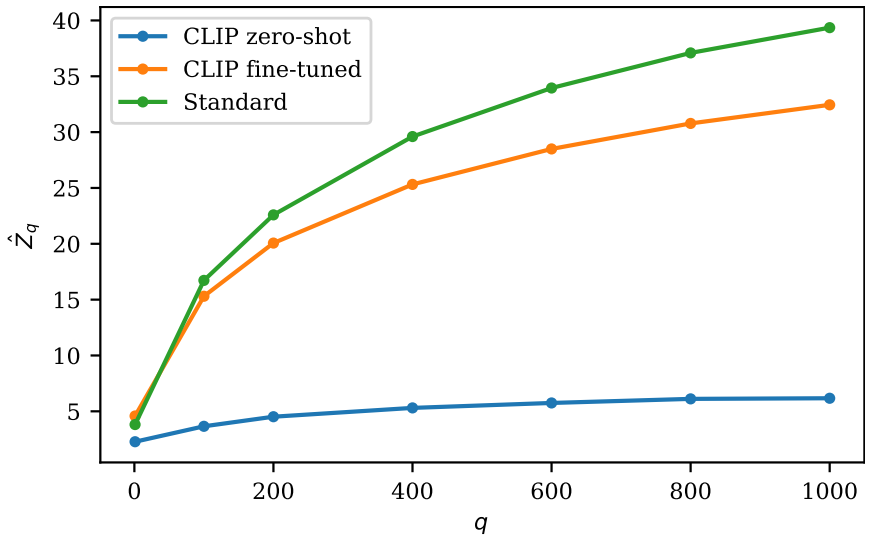


Figure 3.4: Estimated intermediate-q robust loss ($\hat{Z}_q$) over Gaussian perturbations with $\sigma = 0.1$, evaluated at different robustness levels (q).

Table 3.8: Accuracy (%) on Gaussian perturbations with $\sigma = 0.1$ on CIFAR-10. PGD here considers ℓ_2 perturbations.

			ProbAcc(ρ)			
Method	Standard	Rand.	0.1	0.05	0.01	PGD
Standard	**95.24**	**31.05**	**16.18**	**13.52**	**9.13**	0.08
CLIP zero-shot	67.09	18.57	5.40	3.32	0.78	0.00
CLIP fine-tuned	95.22	28.80	11.52	8.09	3.71	**0.20**

3.2.3 Discussion

In this section, we evaluated the intermediate robustness of the large, pre-trained CLIP model to better understand how pre-training on large diverse datasets affects robustness to input corruptions. We found that the intermediate-q objective shows a robustness property of zero-shot CLIP that other objectives do not. While these are rather preliminary findings on the robustness of large-scale pre-trained networks to input corruptions, they present interesting directions for future work. For example, one could use the intermediate-q objective to evaluate different fine-tuning methods, many of which have been proposed to improve out-of-distribution robustness preservation. Some examples of such fine-tuning methods include the following: Kumar et al. [42] suggest an alternative strategy of linear probing followed by fine-tuning for preserving the robustness to distribution shifts. Model soups, constructed by averaging the weights of multiple fine-tuned models, have also been shown to have improved robustness [80]. Wortsman et al. [81] interpolate the weights of fine-tuned and zero-shot models, and Goyal et al. [31] fine-tune using the contrastive loss used for CLIP pre-training with the end result of improving downstream robustness. Ultimately, evaluating resulting fine-tuned models according to intermediate-q robustness metrics could help better enlighten us on the advantages and disadvantages of different

fine-tuning methods.

3.3 Improved training for intermediate robustness

In this section, we discuss work towards further improving training for intermediate robustness. Because approximating the intermediate-q robustness objective requires multiple MCMC samples per training image, the computational complexity of training according to this objective is non-trivial. In order to reduce this computational burden, we first look at applying the training acceleration techniques discussed in Section 2.2.3, e.g. mixed precision arithmetic and cyclic learning rate schedules. The cyclic learning rate schedule allows us to reduce the number of epochs required for training from 200 to 50 on CIFAR-10, for example, and mixed precision arithmetic reduces the time to complete one epoch of training. By taking advantage of these computational speedups, we can use more samples m and Leapfrog steps L during training to improve the estimate of the intermediate-q robustness objective without significantly increasing the total training time.

An additional challenge of training according to the intermediate-q robustness objective when using the path sampling estimator with Hamiltonian Monte Carlo is tuning the HMC parameters, which are described in more detail in Section 3.1.4. The step size α and number of Leapfrog steps L determine the trajectory length αL and must be carefully tuned for HMC to work well. Given a fixed number of Leapfrog steps, too large of a step size and the algorithm will reject too many proposals, whereas too small of a step size will result in wasted computation. Similarly, if L is too large, then the trajectory can "loop back" to the initial state, and if L is too small, then the resulting samples will be highly correlated. Either case is suboptimal for the convergence of the estimator, and will likely result in an inaccurate approximation of the objective given a fixed number of samples. While several works have proposed methods for automatically tuning the HMC parameters [39, 40] to meet an optimal acceptance rate of 65% [57], we choose to use a simple incremental update approach that we find works well in practice for our purposes, without significantly affecting the total training time. We find that automatically tuning the HMC parameters, along with using more samples and Leapfrog iterations, improves upon preliminary results presented in Section 3.1.6. Specifically, we find that models trained using the path sampling-based objective greatly outperform the models trained using the Monte Carlo-based objective for the same value of q.

Finally, we explore alternatives to training using the intermediate-q robustness metric, considering more efficient training methods that could be used to train models that to intermediate robustness levels. While we find that the models trained according to the path-sampling based objectives perform quite well in terms of intermediate robustness when compared to those trained according to the Monte Carlo-based objectives, and we use training acceleration techniques to make training more efficient, we ultimately admit that it is still a rather expensive training procedure. For this reason, we consider other training methods that balance standard and robust performance in some manner, and evaluate the intermediate-q robustness of the resulting models, to determine the best alternative. Specifically, we train models using TRADES varying the trade-off between standard and adversarial performance, probabilistic robustness (PRL) varying the robustness tolerance, and PGD varying the perturbation size. We find that with the optimal

hyperparameter for a given robustness evaluation level, these alternative training methods can result in similar robustness to the model trained using the path sampling-based objective at that value of q. In some cases, the best setting of the alternative training methods we consider, can outperform the path sampling model trained. Ultimately, this study sheds more light on how to best train for robustness between the worst and average case.

3.3.1 Related work

Tuning MCMC parameters In Markov chain Monte Carlo (MCMC) sampling, often a burn-in period is used until the Markov chain converges to the target distribution, such that these initial burn-in samples are discarded. One common practice is to tune any MCMC parameters during the burn-in phase, and then freeze the parameters during sampling thereafter [29]. Hamiltonian Monte Carlo, discussed in Section 3.1.4, is one of the more efficient MCMC sampling methods. However this efficiency depends largely on two parameters: the step size α and number of Leapfrog steps L. There exists a large body of work on methods for automatically tuning the HMC parameters [39, 40]. However, we simply use the basic approach of using a fixed number of Leapfrog steps L and adjusting the step size until the sampling acceptance rate is close to optimal. The optimal acceptance rate for HMC can be analytically derived, and is 65% [57].

Alternative methods for balancing robustness TRADES, discussed in Section 2.1, is an adversarial training method that balances robust and standard accuracy by minimizing a surrogate loss with the first term encouraging low standard error, and the second term, weighted by β, encouraging low adversarial error. Typically TRADES is used with the setting of $\beta = 6$, which has been chosen for the best adversarial performance. In this section, we explore how different values of β affect the resulting intermediate-q robustness of the final model.

Following our introduction of the intermediate-q robustness objective, Robey et al. [65] proposed the notion of a probabilistically robust learning (PRL) based on the loss,

$$\mathbb{I}\{\mathbb{P}_{\delta\sim\mu}(h_\theta(x+\delta) \neq y) > \rho\},$$

where $0 \leq \rho < 1$. Probabilistic robustness encourages robustness to most perturbations, where ρ determines the level of robustness, with smaller values of ρ corresponding to stricter levels of robustness. This notion of probabilistic robustness can be generalized to any loss function, where

$$\begin{aligned} u^*(\rho) = \min_{u\in\mathbb{R}} & \; u \\ \text{s.t. } & \mathbb{P}_{\delta\sim\mu}[\ell(h_\theta(x+\delta), y) \leq u] > 1-\rho. \end{aligned}$$

More plainly, perturbations that incur high loss but are within a subset of the perturbation set that has probability less than ρ, can be ignored. In order to train using this objective, Robey et al. [65] uses the conditional value-at-risk (CVaR) convex upper bound, which can be computed using SGD by sampling perturbations from the perturbation distribution. In this section, to explore alternative training methods as well as compare alternative notions of intermediate robustness, we compare how varying ρ affects intermediate-q robustness for different values of q. Robey

et al. [65] also introduced a quantile accuracy metric based on the zero-one probabilistically robust loss, such that

$$\text{ProbAcc}(\rho) = \mathbb{I}[\mathbb{P}_{\delta \sim \mu}[h_\theta(x+\delta) = y] \geq 1 - \rho.$$

This metric essentially measures the percentage of examples that are probabilistically robust, given tolerance level ρ. In this section, in addition to evaluating the trained models using our intermediate-q objective, we also evaluate models according to this quantile accuracy metric.

3.3.2 More accurately approximating the objective during training

In order to improve the approximation of the intermediate-q objective during training, we increase both the number of samples m and the number of Leapfrog steps L, as compared to preliminary training results presented in Section 3.1.6, and incorporate automatic step size tuning for HMC to maintain an optimal average acceptance rate throughout training.

Automatically tuning the HMC step size For simplicity, we choose to fix the number of Leapfrog steps and use a simple burn-in phase to determine the step size. Recall that our usage of HMC for path sampling involves sampling from multiple different (annealed) distributions rather than a single distribution. Given a mini-batch of data and an initial guess for the step size, we use path sampling to approximate the intermediate-q robustness objective on this mini-batch, and record the acceptance rate of HMC when sampling from the m annealed distributions $p(\delta|t^{(i)})$ (as described in Algorithm 5). We compute the average acceptance rate over the m sampling distributions, and over the mini-batch. To achieve an average acceptance close to the optimal 65% [57], we then adjust the step size to increase or decrease the acceptance rate accordingly. Specifically in our implementation, if the acceptance rate is larger than 70%, we increase the step size by 20%, or if the acceptance rate is smaller than 60%, we decrease the step size by 20%, repeatedly evaluating over the given mini-batch with the updated step size (and a random initial state) until the acceptance rate is between 60% and 70%.

During training, we observe that a single burn-in period at the beginning of training is insufficient. We find that the step size needs to be re-adjusted with the updated model parameters as training progresses in order to maintain the optimal acceptance rate of around 65%. To handle this, we include a burn-in phase at the beginning of each training epoch using a single mini-batch of training data to determine any step size adjustments. In practice, we limit the number of burn-in iterations to avoid excessively long burn-in periods, however with a reasonable initial guess for the step size, we observe that the acceptance rate criterion is usually met before the limit is hit. We also observe that this burn-in phase typically requires more iterations during the first few epochs of training, but then the acceptance rate begins to stabilize for later epochs.

We ultimately improve intermediate-q robustness training via better approximations of the objective during training through this combination of automatically adjusting the HMC step size at the beginning of each epoch, using more samples, and taking more Leapfrog steps. The use of mixed precision arithmetic and cyclic learning rate schedules, along with using a small network architecture (PreActResNet18), makes these improvements computationally feasible. We also note that the choice of m is upper bounded by what GPU memory will allow, given a fixed batch size.

3.3.3 Alternatives to intermediate-q robust training

While the cyclic learning rate schedule and the use of mixed precision arithmetic make training using intermediate-q robustness objectives more computationally reasonable, approximating the objective using path sampling with HMC is still quite expensive. While we may allow a computationally expensive evaluation procedure, in practice, for training purposes, we might seek less costly alternatives. For this reason, in addition to training according to intermediate-q objectives, we also explore alternative training methods that encourage robustness between the worst and average case. For example, we train models using TRADES varying the parameter β that trades off standard and adversarial performance. We also train models according to the probabilistically robust learning (PRL) objective proposed by Robey et al. [65] and vary the robustness tolerance parameter ρ. Lastly, we consider training using the worst-case objective varying the parameters of the perturbation distribution to change the size of the threat model. For example, we consider training with PGD on smaller perturbation sets than considered during evaluation. We evaluate the intermediate-q robustness of each of these models across robustness levels (different values of q) to better determine how these alternative training methods engender intermediate robustness. We additionally evaluate the quantile accuracy [65] of each model with respect to the evaluation perturbation distribution.

3.3.4 Experiments

We consider robustness on CIFAR-10 with respect to two different additive noise perturbations, including perturbations uniformly distributed within ℓ_∞-norm ball with radius $\epsilon = 8/255$ and isotropic Gaussian perturbations with $\sigma = 0.1$.

Evaluation implementation details When computing $\hat{Z}_{\{\text{MC, PS}\},q}$ for evaluation purposes, we use $m = 100$ samples. When computing $\hat{Z}_{\text{PS},q}$ with HMC, we use $L = 10$ Leapfrog steps. We use a burn-in period on one mini-batch of test data to determine an appropriate HMC step size and then fix this step size and compute the intermediate-q robustness objective on the full test set. We evaluate intermediate-q robustness for $q \in [1, 1000]$. We limit the number of burn-in steps during evaluation to 25, scaling the step size α by 20% in the proper direction at each burn-in step accordingly. Following Robey et al. [65], we use $m = 100$ random samples from the perturbation distribution to compute ProbAcc(ρ), and evaluate for $\rho \in \{0.01, 0.05, 0.1\}$.

Training implementation details For all models, besides those trained using the PRL objective of Robey et al. [65], we train for 50 epochs using a cyclic learning rate schedule. We use the default settings of the cyclic learning rate scheduler implemented in PyTorch[4]. We use the SGD optimizer with a maximum learning rate of 0.2, batch size of 128, momentum of 0.9, and weight decay of $5 \cdot 10^{-4}$. For training using $\hat{Z}_{\{\text{MC, PS}\},q}$, we use $m = 10$ samples. For training with $\hat{Z}_{\text{PS},q}$, with HMC, we use $L = 10$ Leapfrog steps. We use a burn-in phase on a single mini-batch of training data at the beginning of each epoch and limit the number of burn-in steps

[4] `torch.optim.lr_scheduler.OneCycleLR`

Table 3.9: Evaluations of intermediate-q robustness on CIFAR-10 towards perturbations uniformly distributed within the ℓ_∞-norm ball with radius $\epsilon = 0.03$.

Train method	Variation	Standard	$\hat{Z}_{\text{MC},1}$	$\hat{Z}_{\text{PS},10}$	$\hat{Z}_{\text{PS},10^2}$	$\hat{Z}_{\text{PS},10^3}$	PGD-50
Standard		0.188	0.297	0.747	3.732	8.948	25.835
$\hat{Z}_{\text{MC}}$	$q = 1$	0.206	0.203	0.364	1.825	5.562	23.944
	$q = 10$	0.198	0.198	0.277	1.057	3.601	19.638
	$q = 10^2$	0.199	0.198	0.264	0.951	3.353	19.447
	$q = 10^3$	0.190	0.190	0.258	0.956	3.379	19.432
$\hat{Z}_{\text{PS}}$	$q = 1$	0.200	0.199	0.327	1.533	4.985	23.231
	$q = 10$	0.186	**0.190**	0.233	0.700	2.168	17.184
	$q = 10^2$	0.218	0.220	**0.232**	0.368	0.924	8.622
	$q = 10^3$	0.227	0.228	0.238	**0.342**	0.845	7.115
TRADES	$\beta = 0.01$	0.189	0.205	0.359	1.732	5.026	22.933
	$\beta = 0.05$	0.199	0.202	0.251	0.745	1.985	13.770
	$\beta = 0.25$	0.263	0.264	0.286	0.486	1.203	6.978
	$\beta = 1$	0.299	0.301	0.312	0.418	0.848	3.928
	$\beta = 6$	0.738	0.742	0.743	0.749	0.803	**1.318**
PRL	$\rho = 0.5$	0.890	0.890	0.896	0.987	1.455	5.916
	$\rho = 1$	0.867	0.868	0.875	0.981	1.401	5.056
	$\rho = 2$	0.882	0.882	0.888	0.986	1.360	4.094
	$\rho = 4$	0.902	0.904	0.910	0.999	1.352	3.509
PGD-10	$\epsilon = 0.001$	**0.180**	0.191	0.266	0.999	3.096	18.71
	$\epsilon = 0.0075$	0.225	0.228	0.241	0.370	0.663	6.48
	$\epsilon = 0.015$	0.285	0.288	0.293	0.348	**0.569**	2.884
	$\epsilon = 0.03$	0.520	0.523	0.525	0.542	0.689	1.377

to 10, scaling the step size α by 20% until an optimal acceptance rate is recorded. For training using the PRL objective, we modify our training setup to match that described in Robey et al. [65], training for 115 epochs using an initial learning rate of 0.01, decaying the learning rate by a factor of 10 at epochs 55, 75, and 90, and using a weight decay of $3.5 \cdot 10^{-3}$. We train with $\rho \in \{0.5, 1.0, 2.0, 4.0\}$, also finding that larger values of ρ result in higher levels of robustness, as observed by Robey et al. [65]. For TRADES, we use 10 steps and train with $\beta \in \{0.01, 0.05, 0.25, 0.5, 6.0\}$. We implement mixed precision training using the automatic mixed precision package included with PyTorch[5], and use mixed precision for all training runs.

Robustness to ℓ_∞-norm ball perturbations

We first evaluate the robustness of models trained on CIFAR-10 towards perturbations uniformly distributed within the ℓ_∞-norm ball with radius 0.03. The intermediate-q robust losses for each

[5] `torch.amp`

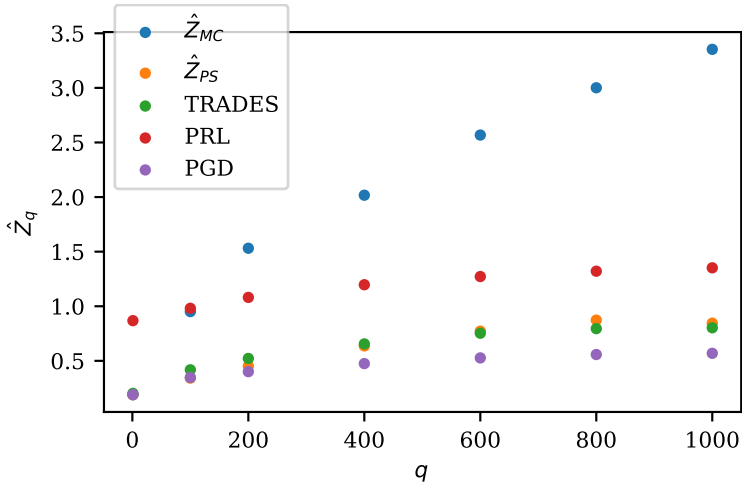
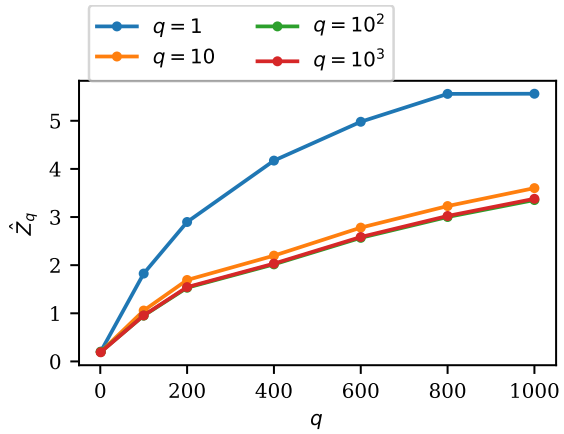


Figure 3.5: Estimated intermediate-q robust loss ($\hat{Z}_q$) over perturbations uniformly distributed within the ℓ_∞-norm ball with $\epsilon = 8/255$, evaluated at different robustness levels (q). Each point represents the best (lowest) loss the training method achieves for the given q-evaluation across different hyperparameter sweeps tested.

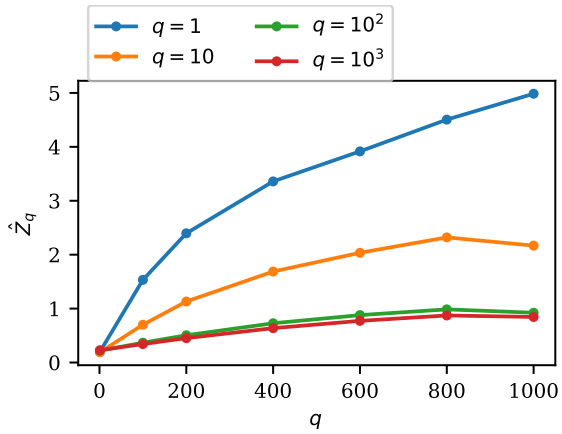
model and hyperparameter are shown in Table 3.9 for $q \in \{1, 10, 10^2, 10^3\}$, along with standard and PGD losses. Intermediate robust losses at more fine-grained intervals between 1 and 10^3 are plotted for each method in Figure 3.6. This figure further helps to visualize the robustness trade-offs across hyperparameter sweeps for each training method. Finally, we also plot the best robustness achieved across all hyperparameter sweeps for each method at each value of q in Figure 3.5.

We find that models trained using $\hat{Z}_{\text{PS},q}$ for $q \in \{10, 10^2, 10^3\}$ have the best intermediate-q robustness (a lower value of the objective) for evaluated robustness levels up to and including $q = 10^2$. At robustness level $q = 10^3$, the models trained using PGD or TRADES (with $\beta = 6$) exhibit more robustness. Because accurately approximating the intermediate-q objective at larger values of q typically requires more samples, likely the number of samples used during training under-approximates the objective for $q = 10^3$. This can also be observed in Figure 3.6b, which shows that the curve for the model trained with $q = 10^3$, plotting intermediate-q objective values for increasing q, does not differ much from that of the model trained with $q = 10^2$. Regardless, we find that these updates to the path sampling-based training procedure, even with the still relatively small number of samples $m = 10$, much more effectively approximates the desired objective as compared to using Monte Carlo-based training. As shown in Figure 3.6a, Monte Carlo-based training with values of q greater than 10 does not significantly change the intermediate robustness of the resulting model. The superiority of the path-sampling based objective for training compared to the Monte-Carlo based objective can further be visualized in Figure 3.5, where after $q = 10^2$, the best Monte-Carlo trained models have a much higher robust loss than the best path sampling trained models.

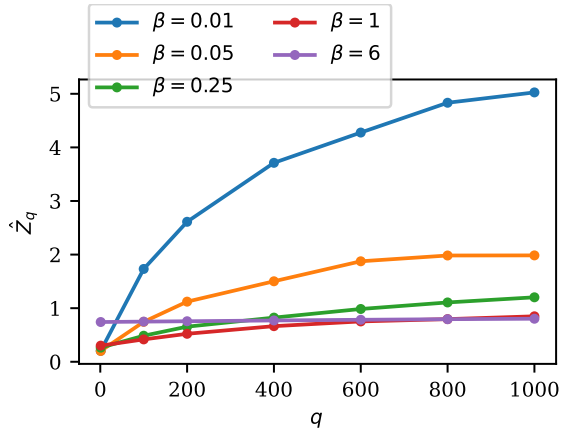
We find that reducing the value of β when training for TRADES, which reduces the weight of the adversarial loss term in the TRADES objective, does result in a trade-off in different levels of robustness, which is more easily visualized in Figure 3.6c. The best TRADES trained



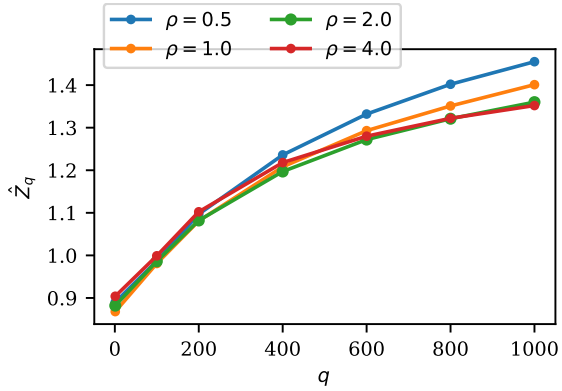
(a) $\hat{Z}_{\text{MC}}$ varying q.



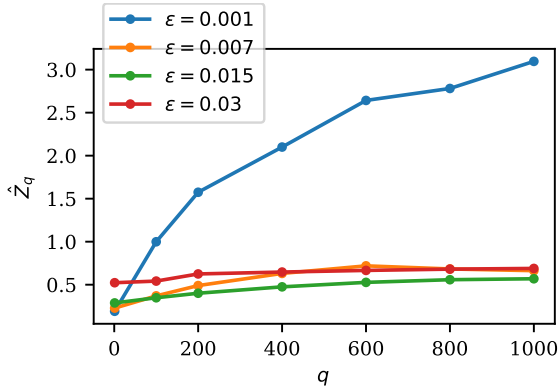
(b) $\hat{Z}_{\text{PS}}$ varying q.



(c) TRADES varying β.



(d) PRL varying ρ.



(e) PGD-10 varying ϵ.

Figure 3.6: Estimated intermediate-q robust loss ($\hat{Z}_q$) over perturbations uniformly distributed within the ℓ_∞-norm ball with $\epsilon = 8/255$, evaluated at different robustness levels (q). Each figure evaluates models that were trained according to the specified objective.

Table 3.10: Accuracy (%) on perturbations uniformly distributed within the ℓ_∞-norm ball with $\epsilon = 8/255$ on CIFAR-10.

				ProbAcc(ρ)			
Train method	Variation	Standard	Rand.	0.1	0.05	0.01	PGD-50
Standard		94.37	91.36	86.21	84.39	79.22	0.00
$\hat{Z}_{\text{MC}}$	$q = 1$	94.03	94.10	91.82	90.84	88.43	0.00
	$q = 10$	93.99	94.06	92.19	91.45	90.02	0.05
	$q = 10^2$	93.73	93.84	92.23	91.61	90.25	0.05
	$q = 10^3$	93.86	93.87	92.12	91.55	90.39	0.03
$\hat{Z}_{\text{PS}}$	$q = 1$	93.97	93.87	91.39	90.57	88.78	0.00
	$q = 10$	94.32	**94.29**	**92.96**	**92.56**	**91.31**	0.17
	$q = 10^2$	92.90	92.82	91.87	91.65	91.05	8.26
	$q = 10^3$	92.33	92.21	91.38	91.09	90.50	10.64
TRADES	$\beta = 0.05$	93.64	93.61	92.25	91.74	90.62	0.88
	$\beta = 0.25$	92.24	92.21	91.20	91.00	90.42	20.78
	$\beta = 1$	90.87	90.85	90.00	89.60	89.06	37.40
	$\beta = 6$	81.69	81.59	80.85	80.62	80.13	**50.71**
PRL	$\rho = 0.5$	89.85	89.48	85.86	84.47	81.38	1.13
	$\rho = 1$	90.36	89.70	86.50	85.21	82.44	1.02
	$\rho = 2$	91.68	91.24	88.29	87.21	84.93	5.00
	$\rho = 4$	92.37	92.22	89.58	88.70	86.31	11.06
	ρ^*[6]	93.82	93.77	91.45	90.63	88.55	0.71
PGD-10	$\epsilon = 0.001$	**94.47**	94.13	92.65	91.95	90.34	0.05
	$\epsilon = 0.0075$	92.72	92.59	91.74	91.53	90.85	18.22
	$\epsilon = 0.015$	90.43	90.32	89.56	89.38	88.72	35.85
	$\epsilon = 0.03$	83.88	83.72	82.94	82.73	82.10	49.90

models start to outperform models trained using $\hat{Z}_{\text{PS}}$ on intermediate-q robustness for values of $q \in [600, 1000]$. We find that none of the models trained with the PRL objective perform as well as those trained with the intermediate-q, TRADES, or PGD objectives on any of the evaluation metrics. We note that while we did base our PRL implementation on the official code repository[7] and used hyperparameters from Robey et al. [65], there could be some implementation differences. As shown in Table 3.9 and Figure 3.6d, training with different values of ρ does not change the test-time intermediate-q robustness very significantly, with most of the model differences becoming more apparent at stricter evaluation robustness levels. We see that training with PGD with a smaller radius results in better intermediate robustness when evaluated at $\epsilon = 0.03$ than training using PGD with the same $\epsilon = 0.03$, as shown in Figure 3.6e and actually performs the best out of all the considered training methods for values of $q \geq 200$. However, decreasing the

[7] `https://github.com/arobey1/advbench`

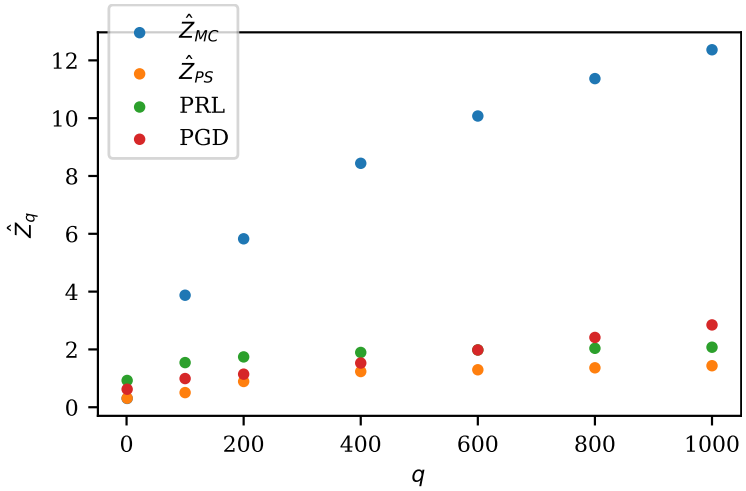


Figure 3.7: Estimated intermediate-q robust loss ($\hat{Z}_q$) over Gaussian perturbations with $\sigma = 0.1$, evaluated at different robustness levels (q). Each point represents the best (lowest) loss the training method achieves for the given q-evaluation across different hyperparameter sweeps tested.

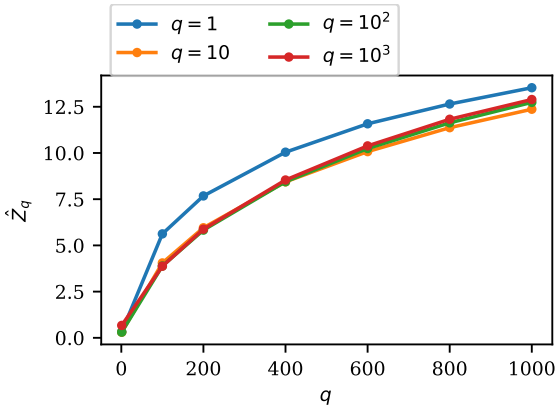
radius too small, e.g. $\epsilon = 0.001$ in this case, can unsurprisingly result in degraded intermediate robustness.

Finally, we also compare each of these models in terms of standard accuracy, adversarial (PGD) accuracy, average accuracy on random samples (denoted by Rand.) and the quantile accuracy metric (ProbAcc(ρ)) proposed by Robey et al. [65], shown in Table 3.10. We find that the model trained using $\hat{Z}_{\text{PS},10}$ performs best on random perturbations along with all of the quantile accuracy metrics. This suggests that the quantile accuracy metric is less strict than the intermediate-q robustness metrics we consider. However, this is not surprising, as this metric does not consider loss values. Additionally, the samples the models are evaluated on are randomly drawn from the perturbation distribution, and so low probability "hard" samples are less likely to be drawn.

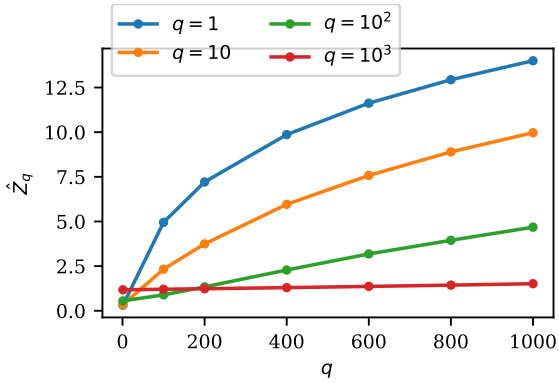
From Table 3.10, we can also examine the trade-off between standard and robust accuracy for different hyperparameters. For example, one can observe the trade-off between standard and robust accuracy that occurs as we increase q when training with $\hat{Z}_{\{\text{PS, MC}\},q}$, and as we increase β when training with TRADES. Similarly, as we decrease ϵ on PGD, we see an increase in all accuracy metrics, other than adversarial accuracy. However, for PRL we observe that the standard accuracy increases with the robust accuracy as we increase ρ, supporting the observation by Robey et al. [65] that there appears to be difficulty in learning with lower values of ρ.

Robustness to Gaussian perturbations

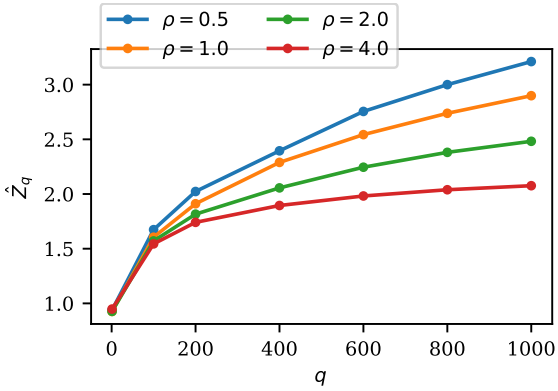
Next, we present evaluations of models trained for robustness towards isotropic Gaussian perturbations with $\sigma = 0.1$. We compare intermediate-q robust loss values in Table 3.11 and Figure 3.8. We note PGD here refers to ℓ_2-norm ball with $\epsilon = 128/255$, and should not be interpreted as the "worst-case" for this perturbation distribution.



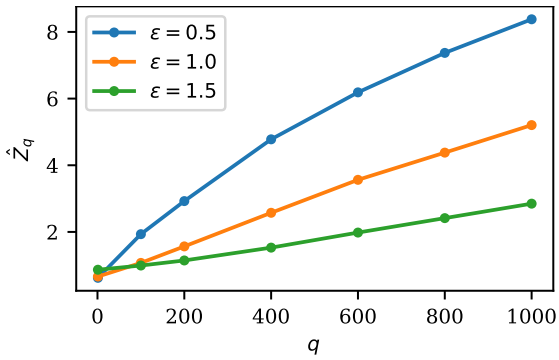
(a) $\hat{Z}_{\text{MC}}$ varying q.



(b) $\hat{Z}_{\text{PS}}$ varying q.



(c) PRL varying ρ.



(d) PGD (ℓ_2) varying ϵ.

Figure 3.8: Estimated intermediate-q robust loss ($\hat{Z}_q$) over Gaussian perturbations with $\sigma = 0.1$, evaluated at different robustness levels (q). Each figure evaluates models that were trained according to the specified objective.

Table 3.11: Evaluations of intermediate-q robustness on CIFAR-10 towards Gaussian perturbations with $\sigma = 0.1$.

Train method	Variation	Standard	$\hat{Z}_{\text{MC},1}$	$\hat{Z}_{\text{PS},10}$	$\hat{Z}_{\text{PS},10^2}$	$\hat{Z}_{\text{PS},10^3}$	PGD-50
Standard		**0.188**	5.007	6.685	12.124	18.172	19.433
$\hat{Z}_{\text{MC}}$	$q = 1$	0.371	0.323	1.117	5.624	13.530	3.624
	$q = 10$	0.326	**0.305**	0.705	4.051	12.367	2.123
	$q = 10^2$	0.327	0.309	0.669	3.887	12.738	1.956
	$q = 10^3$	0.333	0.313	0.671	3.875	12.892	1.946
$\hat{Z}_{\text{PS}}$	$q = 1$	0.363	0.313	0.914	4.953	14.00	2.90
	$q = 10$	0.352	0.346	**0.506**	2.328	9.969	1.254
	$q = 10^2$	0.548	0.555	0.582	**0.891**	4.681	0.889
	$q = 10^3$	1.166	1.176	1.179	1.205	**1.515**	1.296
PRL	$\rho = 0.5$	0.959	0.938	1.027	1.674	3.211	1.745
	$\rho = 1$	0.947	0.926	1.018	1.605	2.899	1.714
	$\rho = 2$	0.948	0.927	1.017	1.571	2.482	1.666
	$\rho = 4$	0.967	0.948	1.037	1.544	2.076	1.586
PGD-10	$\epsilon = 0.5$	0.345	0.623	0.729	1.933	8.379	0.883
	$\epsilon = 1$	0.533	0.622	0.694	1.071	5.205	**0.845**
	$\epsilon = 1.5$	0.780	0.866	0.876	0.992	2.848	0.978

As shown in Table 3.11, we find that the models trained using $\hat{Z}_{\text{PS},q}$ perform best on the intermediate-q evaluation that corresponds to the value of q they were trained with for $q > 1$. Consistent with the ℓ_∞-norm ball setting, we find that for all values of $q > 1$, the model trained using $\hat{Z}_{\text{PS},q}$ outperforms the corresponding model trained using $\hat{Z}_{\text{MC},q}$, suggesting that even with a limited number of samples during training, the path sampling estimator more accurately approximates the objective for this perturbation distribution as well. Figure 3.8a especially shows the inability of the Monte Carlo estimator to accurately approximate the objective over this Gaussian perturbation distribution for larger values of q, as all of the models trained with larger levels of q result in similar intermediate robust performance. Additionally, given this more challenging perturbation distribution, we find the trade-off between (large q) robustness and standard performance is more extreme. This is especially notable for the model trained with $\hat{Z}_{\text{PS},10^3}$, which has the lowest intermediate robust loss for evaluations at $q = 10^3$, but has much higher intermediate robust losses at lower values of q (which approach average robustness).

We also observe that, similarly to the ℓ_∞-norm ball setting, varying ρ when training with the PRL objective does not significantly change the resulting intermediate robustness of the models as compared to other training methods, as visualized in Figure 3.8c. We see that training with increasing robustness levels on Gaussian perturbations here also improves adversarial robustness to ℓ_2-norm ball perturbations. Conversely, training using ℓ_2 PGD adversarial training with different values of ϵ has an effect on the intermediate-q robustness of the resulting models, as shown in Table 3.11. However, as compared to the ℓ_∞ setting, the optimal choice of ϵ is less obvious,

Table 3.12: Accuracy (%) on Gaussian perturbations with $\sigma = 0.1$ on CIFAR-10. The PGD-50 evaluation considers ℓ_2-norm ball perturbations with $\epsilon = 0.5$.

				ProbAcc(ρ)			
Train method	Variation	Standard	Rand.	0.1	0.05	0.01	PGD-50
Standard		**94.37**	23.87	12.24	10.02	5.79	0.02
$\hat{Z}_{\text{MC}}$	$q = 1$	89.60	**90.45**	82.96	79.70	71.52	38.93
	$q = 10$	89.74	90.22	**84.22**	81.78	75.88	51.93
	$q = 10^2$	89.13	89.82	84.08	82.04	76.21	52.89
	$q = 10^3$	89.30	89.77	84.11	81.82	76.09	53.05
$\hat{Z}_{\text{PS}}$	$q = 1$	88.64	89.72	82.20	78.95	70.92	40.18
	$q = 10$	88.31	88.56	84.01	**82.29**	**78.34**	63.79
	$q = 10^2$	81.90	81.79	77.99	77.08	74.73	67.62
	$q = 10^3$	67.55	67.19	64.58	63.80	62.10	59.42
PRL	$\rho = 0.5$	82.39	84.48	73.54	69.05	58.94	56.41
	$\rho = 1$	83.60	84.95	75.14	71.20	62.22	53.38
	$\rho = 2$	85.39	86.44	77.53	74.02	64.98	54.85
	$\rho = 4$	87.49	88.16	80.01	76.37	67.78	58.13
PGD-10	$\epsilon = 0.5$	88.33	78.01	73.02	71.29	67.68	68.38
	$\epsilon = 1$	82.87	76.54	72.64	71.36	68.96	**68.89**
	$\epsilon = 1.5$	75.00	70.89	67.85	66.96	65.04	65.39

and the best performance of ℓ_2 PGD never surpasses that of $\hat{Z}_{\text{PS},q}$, as shown in Figure 3.7.

Lastly, we also compare model accuracy on Gaussian perturbations in Table 3.12. We again see that the models trained with $q = 10$ perform best on the quantile accuracy metrics, with the path sampling-based training performing best for the more stringent robustness levels of $\rho \in \{0.05, 0.01\}$. The model trained with path sampling at $q = 10^3$ has especially degraded accuracy across these metrics, due to the stringency of the robust training method.

3.3.5 Discussion

In this section, we improved training for intermediate-q robustness, and explored alternative methods for training for fine-grained levels of robustness. Training using the intermediate-q objective approximated using path sampling with HMC resulted in the best intermediate-q robustness for smaller values of q for the ℓ_∞ perturbation setting, until more samples are required for a more accurate estimation of the objective. For the Gaussian setting, training using the intermediate-q objective approximated using path sampling with HMC resulted in the best intermediate-q robustness for all values of q. We showed that training using $\hat{Z}_{\text{PS},q}$ also outperformed alternatives in terms of quantile accuracy, a probabilistic robustness metric, for both perturbation distributions. However, we recognize that training using $\hat{Z}_{\text{PS},q}$ is still significantly slower than alternatives, even after incorporating general acceleration techniques, as shown in

Table 3.13: Average training times (in hours) for different robust training methods on CIFAR-10. Each experiment was run on a single GeForce RTX 2080 Ti using the PreActResNet18 architecture. Models trained using PRL were trained for 115 epochs, and the remainder were trained for 50 epochs.

Train method	Train time (hrs)
Standard	0.31
$\hat{Z}_{\text{MC}}$	1.28
$\hat{Z}_{\text{PS}}$	16.13
TRADES	2.83
PRL	6.85
PGD-10	1.44

Table 3.13. Furthermore, while we were able to train according to $\hat{Z}_{\text{PS},q}$ in a reasonable amount of time due to using a small network architecture, we note that when using a larger architecture, the training batch size will likely need to be reduced based on the number of samples m due to GPU memory considerations. However, we showed that it is possible to improve intermediate robustness by varying the parameters of alternative training methods, for example by reducing the radius of the ℓ_∞-norm ball and training using PGD. We showed that our intermediate-q robustness evaluation metric remains valuable for evaluating these alternative training methods at intermediate robustness levels.

Some interesting directions for future work are further accelerating the estimation of $\hat{Z}_{\text{PS},q}$ during training, and applying HMC parameter tuning techniques to the parameter L in addition to step size. Another direction that would be of benefit, is making the choice of q a more interpretable parameter. Lastly, another interesting direction could be to use model soups (linear combinations of model parameters), which have been used to trade-off robustness to different ℓ_p-norm ball perturbation sets [18], to trade-off average-case and worst-case robustness.

Chapter 4

Conclusion

In this thesis, we have presented methods for training and evaluating the robustness of machine learning models. In Chapter 2, we discussed the topic of adversarial robustness, which studies robustness of machine learning models to worst-case input perturbations. We presented several improvements upon adversarial training, an empirical approach for producing robust networks by training on worst-case perturbed inputs, or adversarial examples. We showed that the original adversarial training approach using the fast gradient sign method, which had been largely determined to be ineffective against the later introduced projected gradient descent attack, can surprisingly result in models robust to PGD attacks with certain training modifications. The efficiency of this single-step adversarial training method, combined with techniques for accelerating standard training of deep neural networks, allowed us to train robust networks much faster than before. We also uncovered a property of adversarial training that does not commonly occur in the standard training of deep neural networks, namely that overfitting during robust training significantly harms robust test performance. By early stopping training, we achieved significant gains in the robustness of the final trained models, across different forms of adversarial training, datasets and threat models. We showed that projected gradient descent remains competitive with subsequently proposed algorithmic improvements for training robust deep networks.

In Chapter 3, we transitioned from focusing on worst-case robustness, and discussed motivation for an alternative notion of robustness that lies between average-case and worst-case robustness. We showed that there exists a natural interpretation of robustness that encompasses both average-case and worst-case notions of robustness, while also capturing the spectrum of robustness between these two extremes. Our proposed objective ultimately allows for evaluating the robustness of models on a wide range of robustness levels. We further showed that we can train on this objective to achieve models that perform well at stricter robustness levels, while being less conservative than adversarial training. We then discussed the utility of our metric for evaluating the robustness of models trained without any specific robustness objective. Specifically, we applied our metric to measure how large-scale pre-training and task-specific fine-tuning affects robustness to input corruptions by evaluating OpenAI's CLIP model. We found that our proposed robustness metric showed the robustness of the zero-shot predictor, and the degradation in robustness that results from fine-tuning with respect to input corruptions where other metrics did not. Finally, we showed ways of improving training using the path sampling-based intermediate robustness objective, and compared to alternative methods for training for robustness

between the worst and average case.

Bibliography

[1] Jean-Baptiste Alayrac, Jonathan Uesato, Po-Sen Huang, Alhussein Fawzi, Robert Stanforth, and Pushmeet Kohli. Are labels required for improving adversarial robustness? *Advances in Neural Information Processing Systems*, 32, 2019. 2.3.1, 2.3.6

[2] Maksym Andriushchenko and Nicolas Flammarion. Understanding and improving fast adversarial training. *Advances in Neural Information Processing Systems*, 33:16048–16059, 2020. 2.2.4

[3] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *International conference on machine learning*, pages 274–283. PMLR, 2018. 2.1

[4] Anish Athalye, Logan Engstrom, Andrew Ilyas, and Kevin Kwok. Synthesizing robust adversarial examples. In *International conference on machine learning*, pages 284–293. PMLR, 2018. 2.1

[5] 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. 2.3, 2.3.1, 2.3.5

[6] Aharon Ben-Tal, Laurent El Ghaoui, and Arkadi Nemirovski. *Robust optimization*, volume 28. Princeton university press, 2009. 1

[7] Charles H Bennett. Efficient estimation of free energy differences from monte carlo data. *Journal of Computational Physics*, 22(2):245–268, 1976. 3.1.1

[8] Michael Betancourt. A conceptual introduction to hamiltonian monte carlo. *arXiv preprint arXiv:1701.02434*, 2017. 3.1.1

[9] Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021. 3.2

[10] Jacob Buckman, Aurko Roy, Colin Raffel, and Ian Goodfellow. Thermometer encoding: One hot way to resist adversarial examples. In *International conference on learning representations*, 2018. 2.1

[11] Nicholas Carlini. Is ami (attacks meet interpretability) robust to adversarial examples? *arXiv preprint arXiv:1902.02322*, 2019. 2.1

[12] Nicholas Carlini and David Wagner. Adversarial examples are not easily detected: By-

passing ten detection methods. In *Proceedings of the 10th ACM workshop on artificial intelligence and security*, pages 3–14, 2017. 2.1

[13] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*, pages 39–57. Ieee, 2017. 2.1

[14] Yair Carmon, Aditi Raghunathan, Ludwig Schmidt, John C Duchi, and Percy S Liang. Unlabeled data improves adversarial robustness. *Advances in Neural Information Processing Systems*, 32, 2019. 2.3.1, 2.3.6, 2.3.6

[15] Tianlong Chen, Sijia Liu, Shiyu Chang, Yu Cheng, Lisa Amini, and Zhangyang Wang. Adversarial robustness: From self-supervised pre-training to fine-tuning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 699–708, 2020. 3.2

[16] 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. 2.2, 2.2.1

[17] Francesco Croce and Matthias Hein. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In *International conference on machine learning*, pages 2206–2216. PMLR, 2020. 2.3

[18] Francesco Croce, Sylvestre-Alvise Rebuffi, Evan Shelhamer, and Sven Gowal. Seasoning model soups for robustness to adversarial and natural distribution shifts. *arXiv preprint arXiv:2302.10164*, 2023. 3.3.5

[19] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017. 2.3, 2.3.1, 2.3.6

[20] Yinpeng Dong, Fangzhou Liao, Tianyu Pang, Hang Su, Jun Zhu, Xiaolin Hu, and Jianguo Li. Boosting adversarial attacks with momentum. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9185–9193, 2018. 2.1

[21] Simon Duane, Anthony D Kennedy, Brian J Pendleton, and Duncan Roweth. Hybrid monte carlo. *Physics letters B*, 195(2):216–222, 1987. 3.1.1

[22] Logan Engstrom, Brandon Tran, Dimitris Tsipras, Ludwig Schmidt, and Aleksander Madry. A rotation and a translation suffice: Fooling cnns with simple transformations. 2017. 2.1

[23] Logan Engstrom, Andrew Ilyas, Shibani Santurkar, and Dimitris Tsipras. Robustness (python library), 2019. URL `https://github.com/MadryLab/robustness`. 2.3.2, 2.3.2

[24] Lijie Fan, Sijia Liu, Pin-Yu Chen, Gaoyuan Zhang, and Chuang Gan. When does contrastive learning preserve adversarial robustness from pretraining to finetuning? *Advances in Neural Information Processing Systems*, 34:21480–21492, 2021. 3.2

[25] Reuben Feinman, Ryan R Curtin, Saurabh Shintre, and Andrew B Gardner. Detecting adversarial samples from artifacts. *arXiv preprint arXiv:1703.00410*, 2017. 2.1

[26] Stanislav Fort. Adversarial examples for the openai clip in its zero-shot classification regime and their semantic generalization. `https://stanislavfort.github.io/`

blog/OpenAI_CLIP_adversarial_examples/, 2021. 3.2

[27] Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A Wichmann, and Wieland Brendel. Imagenet-trained cnns are biased towards texture; increasing shape bias improves accuracy and robustness. *arXiv preprint arXiv:1811.12231*, 2018. 3.1.1

[28] Andrew Gelman and Xiao-Li Meng. Simulating normalizing constants: From importance sampling to bridge sampling to path sampling. *Statistical science*, pages 163–185, 1998. 1.1.2, 3.1.1, 3.1.3

[29] Andrew Gelman, John B Carlin, Hal S Stern, and Donald B Rubin. *Bayesian data analysis*. Chapman and Hall/CRC, 1995. 3.3.1

[30] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014. 1.1.1, 2.1

[31] Sachin Goyal, Ananya Kumar, Sankalp Garg, Zico Kolter, and Aditi Raghunathan. Finetune like you pretrain: Improved finetuning of zero-shot vision models. *arXiv preprint arXiv:2212.00638*, 2022. 3.2.3

[32] Chuan Guo, Mayank Rana, Moustapha Cisse, and Laurens Van Der Maaten. Countering adversarial images using input transformations. *arXiv preprint arXiv:1711.00117*, 2017. 2.1

[33] 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, 2016. 2.2.5

[34] 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, 2016. 2.2.5, 2.3.6

[35] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. *arXiv preprint arXiv:1903.12261*, 2019. 3, 3.1.1

[36] Dan Hendrycks, Mantas Mazeika, Saurav Kadavath, and Dawn Song. Using self-supervised learning can improve model robustness and uncertainty. *Advances in neural information processing systems*, 32, 2019. 3.1.1

[37] Dan Hendrycks, Norman Mu, Ekin D Cubuk, Barret Zoph, Justin Gilmer, and Balaji Lakshminarayanan. Augmix: A simple data processing method to improve robustness and uncertainty. *arXiv preprint arXiv:1912.02781*, 2019. 3.1.1

[38] Dan Hendrycks, Steven Basart, Norman Mu, Saurav Kadavath, Frank Wang, Evan Dorundo, Rahul Desai, Tyler Zhu, Samyak Parajuli, Mike Guo, et al. The many faces of robustness: A critical analysis of out-of-distribution generalization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8340–8349, 2021. 3.1.1

[39] Matthew Hoffman, Alexey Radul, and Pavel Sountsov. An adaptive-mcmc scheme for setting trajectory lengths in hamiltonian monte carlo. In *International Conference on Artificial Intelligence and Statistics*, pages 3907–3915. PMLR, 2021. 3.3, 3.3.1

[40] Matthew D Hoffman, Andrew Gelman, et al. The no-u-turn sampler: adaptively setting path lengths in hamiltonian monte carlo. *J. Mach. Learn. Res.*, 15(1):1593–1623, 2014.

3.3, 3.3.1

[41] Anders Krogh and John Hertz. A simple weight decay can improve generalization. *Advances in neural information processing systems*, 4, 1991. 2.3.1

[42] Ananya Kumar, Aditi Raghunathan, Robbie Jones, Tengyu Ma, and Percy Liang. Fine-tuning can distort pretrained features and underperform out-of-distribution. *arXiv preprint arXiv:2202.10054*, 2022. 3.2, 3.2.3

[43] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*, 2016. 2.1

[44] Tian Li, Ahmad Beirami, Maziar Sanjabi, and Virginia Smith. Tilted empirical risk minimization. *arXiv preprint arXiv:2007.01162*, 2020. 3.1.1

[45] Tian Li, Ahmad Beirami, Maziar Sanjabi, and Virginia Smith. On tilted losses in machine learning: Theory and applications. *arXiv preprint arXiv:2109.06141*, 2021. 3.1.1

[46] 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. 2.1

[47] 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. (document), 1.1.1, 2.1, 2.2, 2, 2.1, 2.2.5, 2.4, 2.3, 2.7, 2.3.2, 13

[48] Pratyush Maini, Eric Wong, and Zico Kolter. Adversarial robustness against the union of multiple perturbation models. In *International Conference on Machine Learning*, pages 6640–6650. PMLR, 2020. 2.1

[49] Xiao-Li Meng and Wing Hung Wong. Simulating ratios of normalizing constants via a simple identity: a theoretical exploration. *Statistica Sinica*, pages 831–860, 1996. 3.1.1

[50] Jan Hendrik Metzen, Tim Genewein, Volker Fischer, and Bastian Bischoff. On detecting adversarial perturbations. *arXiv preprint arXiv:1702.04267*, 2017. 2.1

[51] Laurent Meunier, Meyer Scetbon, Rafael B Pinot, Jamal Atif, and Yann Chevaleyre. Mixed nash equilibria in the adversarial examples game. In *International Conference on Machine Learning*, pages 7677–7687. PMLR, 2021. 3.1.1

[52] 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. 2.2, 2.2.1, 2.2.3

[53] Hadi Mohasel Afshar and Justin Domke. Reflection, refraction, and hamiltonian monte carlo. *Advances in neural information processing systems*, 28, 2015. 3.1.1

[54] Nelson Morgan and Hervé Bourlard. Generalization and parameter estimation in feedforward nets: Some experiments. *Advances in neural information processing systems*, 2, 1989. 2.3.1

[55] 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. 2.1, 2.3

[56] Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. *Journal of Statistical Mechanics: Theory and Experiment*, 2021(12):124003, 2021. 2.3, 2.3.1, 2.3.5

[57] Radford M Neal et al. Mcmc using hamiltonian dynamics. *Handbook of markov chain monte carlo*, 2(11):2, 2011. 3.1.1, 3.3, 3.3.1, 3.3.2

[58] Behnam Neyshabur, Srinadh Bhojanapalli, David McAllester, and Nati Srebro. Exploring generalization in deep learning. *Advances in neural information processing systems*, 30, 2017. 2.3

[59] Yosihiko Ogata. A monte carlo method for high dimensional integration. *Numerische Mathematik*, 55:137–157, 1989. 3.1.1

[60] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *2016 IEEE symposium on security and privacy (SP)*, pages 582–597. IEEE, 2016. 2.1

[61] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763. PMLR, 2021. 1.1.2, 3.2

[62] Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do cifar-10 classifiers generalize to cifar-10? *arXiv preprint arXiv:1806.00451*, 2018. 2.3.4

[63] Leslie Rice, Eric Wong, and Zico Kolter. Overfitting in adversarially robust deep learning. In *International Conference on Machine Learning*, pages 8093–8104. PMLR, 2020. 1.1.1

[64] Leslie Rice, Anna Bair, Huan Zhang, and J Zico Kolter. Robustness between the worst and average case. *Advances in Neural Information Processing Systems*, 34:27840–27851, 2021. 1.1.2

[65] Alexander Robey, Luiz FO Chamon, George J Pappas, and Hamed Hassani. Probabilistically robust learning: Balancing average-and worst-case performance. *arXiv preprint arXiv:2202.01136*, 2022. 3.2.1, 3.3.1, 3.3.3, 3.3.4, 3.3.4, 3.3.4

[66] Ludwig Schmidt, Shibani Santurkar, Dimitris Tsipras, Kunal Talwar, and Aleksander Madry. Adversarially robust generalization requires more data. *Advances in neural information processing systems*, 31, 2018. 2.3.1, 2.3.6, 3.2

[67] Ali Shafahi, Mahyar Najibi, Mohammad Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S Davis, Gavin Taylor, and Tom Goldstein. Adversarial training for free! *Advances in Neural Information Processing Systems*, 32, 2019. 2.1, 2.2, 2.2.1, 1, 2.1, 2.2.5, 2.4, 2.2.5

[68] 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. 2.2.3, 4

[69] Leslie N Smith and Nicholay Topin. Super-convergence: Very fast training of neural networks using large learning rates. In *Artificial intelligence and machine learning for multi-*

domain operations applications, volume 11006, pages 369–386. SPIE, 2019. 2.2, 2.2.1, 2.2.3, 2.2.5

[70] 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. 2.1

[71] 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. 2.3.1

[72] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013. 1, 2.1

[73] Guanhong Tao, Shiqing Ma, Yingqi Liu, and Xiangyu Zhang. Attacks meet interpretability: Attribute-steered detection of adversarial samples. *Advances in Neural Information Processing Systems*, 31, 2018. 2.1

[74] Vincent Tjeng, Kai Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. *arXiv preprint arXiv:1711.07356*, 2017. 2.2.5

[75] Florian Tramer and Dan Boneh. Adversarial training and robustness for multiple perturbations. *Advances in neural information processing systems*, 32, 2019. 2.1

[76] 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), 2.1, 2.2.1, 2.2.2, 2.2.4, 2.3, 2.2.5, 2.2.5

[77] Jianyu Wang and Haichao Zhang. Bilateral adversarial training: Towards fast training of more robust models against adversarial attacks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6629–6638, 2019. 2.2.1

[78] Eric Wong, Frank Schmidt, and Zico Kolter. Wasserstein adversarial examples via projected sinkhorn iterations. In *International Conference on Machine Learning*, pages 6808–6817. PMLR, 2019. 2.1

[79] Eric Wong, Leslie Rice, and J Zico Kolter. Fast is better than free: Revisiting adversarial training. *arXiv preprint arXiv:2001.03994*, 2020. 1.1.1

[80] Mitchell Wortsman, Gabriel Ilharco, Samir Ya Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, et al. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *International Conference on Machine Learning*, pages 23965–23998. PMLR, 2022. 3.2.3

[81] Mitchell Wortsman, Gabriel Ilharco, Jong Wook Kim, Mike Li, Simon Kornblith, Rebecca Roelofs, Raphael Gontijo Lopes, Hannaneh Hajishirzi, Ali Farhadi, Hongseok Namkoong, et al. Robust fine-tuning of zero-shot models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7959–7971, 2022. 3.2.3

[82] 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

[83] 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/CVF conference on computer vision and pattern recognition*, pages 501–509, 2019. 2.1, 2.3

[84] Yuzhe Yang, Guo Zhang, Dina Katabi, and Zhi Xu. Me-net: Towards effective adversarial robustness with matrix estimation. *arXiv preprint arXiv:1905.11971*, 2019. 2.1, 2.3

[85] Dong Yin, Raphael Gontijo Lopes, Jon Shlens, Ekin Dogus Cubuk, and Justin Gilmer. A fourier perspective on model robustness in computer vision. *Advances in Neural Information Processing Systems*, 32, 2019. 3.1.1

[86] Dong Yin, Ramchandran Kannan, and Peter Bartlett. Rademacher complexity for adversarially robust generalization. In *International conference on machine learning*, pages 7085–7094. PMLR, 2019. 2.3.1

[87] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016. 2.3.5

[88] 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. 2.3.1, 2.3.6

[89] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, 64(3):107–115, 2021. 2.3, 2.3.1

[90] Dinghuai Zhang, Tianyuan Zhang, Yiping Lu, Zhanxing Zhu, and Bin Dong. You only propagate once: Accelerating adversarial training via maximal principle. *Advances in Neural Information Processing Systems*, 32, 2019. 2.1, 2.2, 2.2.1

[91] Hongyang Zhang, Yaodong Yu, Jiantao Jiao, Eric Xing, Laurent El Ghaoui, and Michael Jordan. Theoretically principled trade-off between robustness and accuracy. In *International conference on machine learning*, pages 7472–7482. PMLR, 2019. 2.1, 2.3, 2.3.2, 2.3.2, 12, 2.3.4

[92] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017. 2.3, 2.3.1, 2.3.6