# Improved Methodology for Evaluating Adversarial Robustness in Deep Neural Networks

by

Kyungmi Lee

B.S., Seoul National University (2018)

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

Master of Science in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2020

Author . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
Department of Electrical Engineering and Computer Science
May 15, 2020

Certified by. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
Anantha P. Chandrakasan
Vannevar Bush Professor of Electrical Engineering and Computer Science
Thesis Supervisor

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

# Improved Methodology for Evaluating Adversarial Robustness in Deep Neural Networks

by

Kyungmi Lee

Submitted to the Department of Electrical Engineering and Computer Science
on May 15, 2020, in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering and Computer Science

## Abstract

Deep neural networks are known to be vulnerable to adversarial perturbations, which are often imperceptible to humans but can alter predictions of machine learning systems. Since the exact value of adversarial robustness is difficult to obtain for complex deep neural networks, accuracy of the models against perturbed examples generated by attack methods is empirically used as a proxy to adversarial robustness. However, failure of attack methods to find adversarial perturbations cannot be equated with being robust. In this work, we identify three common cases that lead to overestimation of accuracy against perturbed examples generated by bounded first-order attack methods: 1) the value of cross-entropy loss numerically becoming zero when using standard floating point representation, resulting in non-useful gradients; 2) innately non-differentiable functions in deep neural networks, such as Rectified Linear Unit (ReLU) activation and MaxPool operation, incurring "gradient masking" [2]; and 3) certain regularization methods used during training inducing the model to be less amenable to first-order approximation. We show that these phenomena exist in a wide range of deep neural networks, and that these phenomena are not limited to specific defense methods they have been previously investigated for. For each case, we propose compensation methods that either address sources of inaccurate gradient computation, such as numerical saturation for near zero values and non-differentiability, or reduce the total number of back-propagations for iterative attacks by approximating second-order information. These compensation methods can be combined with existing attack methods for a more precise empirical evaluation metric. We illustrate the impact of these three phenomena with examples of practical interest, such as benchmarking model capacity and regularization techniques for robustness. Furthermore, we show that the gap between adversarial accuracy and the guaranteed lower bound of robustness can be partially explained by these phenomena. Overall, our work shows that overestimated adversarial accuracy that is not indicative of robustness is prevalent even for conventionally trained deep neural networks, and highlights cautions of using empirical evaluation without guaranteed bounds.

Thesis Supervisor: Anantha P. Chandrakasan
Title: Vannevar Bush Professor of Electrical Engineering and Computer Science

## Acknowledgments

First of all, I would like to thank Prof. Anantha P. Chandrakasan for advising this thesis and giving me the opportunity to join his research group two years ago. Anantha introduced me to the concept of adversarial robustness, and his advice provided important directions to this thesis. I deeply appreciate Anantha's support and encouragements while working on this thesis.

Next, I would like to thank current and previous members of AnanthaGroup for enjoyable and welcoming atmosphere in the lab. I thank Vipasha for chats and discussions we had on courseworks and adapting to MIT in general. Also, thank you Taehoon, Wanyeong, and Jongchan for coffee runs and chats. Thanks Utsav and Miaorong for being both labmates and TAs, and bearing with my frequent questions. And to Preet, who organized online coffee hours and birthday parties that brought the group members together during this highly unusual time.

I am grateful to MIT Jacobs Presidential Fellowship, Korea Foundation for Advanced Studies, and Siebel Scholars Foundation for the fellowships that supported my graduate studies. Also, I would like to thank NXP and DARPA for sponsoring this research.

Finally, I would like to thank my parents for their love and support. Their encouragements to pursue my dreams and unconditional love made me the person I am today. Thank you for everything.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

This thesis develops an empirical metric for evaluating “adversarial” robustness of deep neural networks, based on the analyses of existing metrics and their limitations. In this chapter, we introduce the concept of adversarial robustness and explain important preliminaries and previous works, which this thesis builds upon.

## 1.1 Motivation

Deep neural networks achieved the state-of-the-art performances for various computer vision benchmarks, such as image classification [19] and object instance segmentation [20], that are relevant to real-world applications. However, recent studies showed that deep neural networks are not robust to input perturbations that deviate from typical “clean” input data distributions. Szegedy et al. found that small perturbations that are often imperceptible to humans can fool the state-of-the-art image classifiers [47], and described such perturbations “adversarial”, in that they are critical worst-case input perturbations that are hard to be found randomly. Figure 1-1 illustrates how adding an adversarial perturbation can change the prediction of a deep neural network, although perceptionally the resulting adversarial example is highly similar to the original example. Papernot et al. further showed that these adversarial perturbations are transferable between different machine learning models, not limited to deep neural networks.

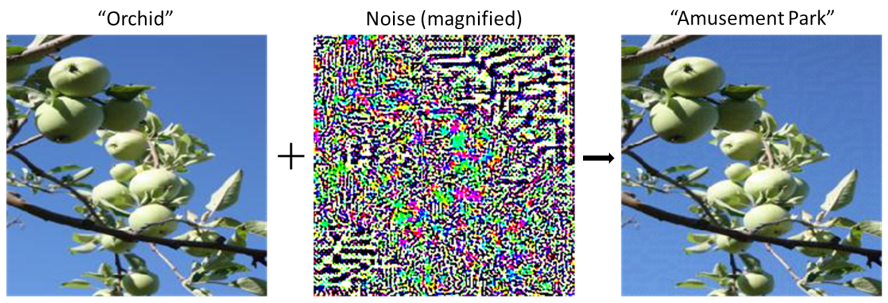


Figure 1-1: An illustration of adversarial perturbations in deep neural networks. The image is sampled from Places365 dataset [55], and we generate the adversarial perturbation (noise in the above figure) using Fast Gradient Sign Method (FGSM) [15]. When an image is encoded in 8-bit (0-255), the noise has the magnitude of 2. Thus, when an image is scaled to be in the range [0, 1], the original example in the left and the adversarial example in the right differs only by $\frac{2}{255} \approx 0.0078$ per pixel for each channel in RGB. To visualize this small noise, we magnify it with the factor $\frac{255}{4} = 63.75$. We use a pretrained ResNet 18 model provided by Zhou et al., that has 53.17% of top-1 accuracy on this dataset. The original example is correctly classified as "orchid" by this model, but the adversarial example is wrongly classified as "amusement park".

The vulnerability of deep neural networks to adversarial perturbations is concerning when deep neural networks are to be deployed in critical real-world applications, since raw inputs to those applications can be exposed to malicious attacks using adversarial perturbations. For example, Athalye et al. realized physical 3-dimensional adversarial examples that tricked image classifiers under different camera viewpoints and rotations [1]; Sharif et al. constructed an eyeglass-shaped adversarial perturbation that could be printed and wore to trick a face recognition system [43]; Eykholt et al. showed that simple black-and-white tapes can be added to road signs to mislead a road sign classifier used for autonomous driving [12].

Therefore, robustness of deep neural networks against adversarial perturbations (often referred as 'adversarial robustness' in short) is becoming an important metric for designing intelligent systems for security-critical applications. An essential but challenging methodology for understanding adversarial robustness is precise measurement of robustness for complex deep neural networks and high-dimensional input samples. This thesis investigates the methodologies for evaluating adversarial robustness, especially focusing on when those methodologies *fail* to indicate true robustness of deep neural networks. In the subsequent sections, we explain preliminaries and previous works, and concludes this chapter with the overview of this thesis.

## 1.2 Adversarial examples in machine learning

Adversarial examples that are crafted with the intention to trick machine learning based decision making systems were introduced and studied since early 2000s [4]. For instance, a linear classifier or a supporting vector machine used to filter spam mails was shown to be vulnerable to intentional modifications that did not hinder readability of spam contents [5].

More recently, adversarial examples have been discovered for complex deep neural networks, initially from the perspective local generalization [47]. Szegedy et al. questioned whether then state-of-the-art image classifiers can generalize well within the local neighborhood of training examples, and used a non-linear optimization method to solve it. In particular, let $f(\cdot)$ be a neural network model, and $x$ be a sample that this model classifies correctly. Let $z = f(x)$ denote output logits of this model on an input sample $x$. Then, the predicted label is $y = \arg\max_i z_i$ where $z_i$ is the value of logit for $i$th class. We can define a loss function (e.g., cross-entropy loss) when the ground truth class is $t$, and denote it $l(z, t)$. Szegedy et al. solved following optimization problem, and found that a small perturbation $r$ that alters the prediction of the model on $x + r$ to be the target class $t^\star \neq t$ exists:

$$\text{minimize } c \cdot \|r\|_2 + l(f(x + r), t^\star) \tag{1.1}$$

$$\text{such that } x + r \in \text{valid image domain} \tag{1.2}$$

where $c$ is a hyperparameter to scale the size of $r$ in the optimization objective. Szegedy et al. used L-BFGS to solve the above optimization problem, and showed that a small $r$ exists for deep neural networks trained on ImageNet dataset [40].

Goodfellow et al. showed that adversarial examples can be more simply found using first-order approximation for non-linear neural networks [15]. Main intuition behind their approximation was that adversarial perturbations can exist even for linear models when the dimension of input samples is sufficiently large. To find a perturbation $r$ with maximum per-pixel distortion $\epsilon$, in other words $\|r\|_\infty \leq \epsilon$, they proposed Fast Gradient Sign Method (FGSM) that chooses $r$ to be:

$$r = \epsilon \cdot \texttt{sign}(\frac{\partial l(f(x), t)}{\partial x}) \tag{1.3}$$

For a linear model, the above perturbation $r$ maximizes the inner-product with the weight vector of the model since its sign aligns with that of the weight vector, thus maximizing the influence of the perturbation to the output. FGSM can be similarly used to generate $r$ in different norms; for $L_2$ norm, we can replace `sign` of Eq (1.3) with division by $\|g\|_2$ where $g = \frac{\partial l(f(x),t)}{\partial x}$ for normalization.

These early works motivated subsequent research on developing *attack* methods that find adversarial examples in computationally efficient manners. Notably, Basic Iterative Method (BIM) [26] and Projected Gradient Descent (PGD) [29] iteratively used first-order approximation. Specifically, PGD initializes a perturbation $r$ using a vector $u$ that is randomly drawn from a distribution (e.g., $\mathcal{N}(0, I)$, Unif$[-1, 1]$), and updates $r^{(i)}$ for $i$th iteration for $\|r\|_\infty \leq \epsilon$ as follows:

$$r^{(0)} = \epsilon \cdot \texttt{sign}(u) \tag{1.4}$$

$$r^{(i)} = r^{(i-1)} + \alpha \cdot \texttt{sign}(\frac{\partial l(f(x + r^{(i-1)}), t)}{\partial r^{(i-1)}}) \tag{1.5}$$

$$r^{(i)} = \texttt{clip}(x + r^{(i)}) - x \tag{1.6}$$

where $\alpha$ is per-step update size in Eq (1.5) and `clip` is an operation that clips $x + r^{(i)}$ to be in the valid image domain in Eq (1.6). BIM omits the random initialization (Eq (1.4)). Computationally, these first-order attack methods require additional back-propagations, and computation increases along with the number of iterations used by the attack methods.

Carlini and Wagner reformulated the optimization objective for finding adversarial examples (C&W attack), such that the objective is minimized only when a prediction on a perturbed input is wrong [6]. While the high-level formulation is similar to that of Szegedy et al., Carlini and Wagner replaced the loss term in Eq (1.1) with $g_{obj}(x, t^\star)$, where $t^\star$ is the desired target label that is not the ground truth $t$, such that $g_{obj}(x, t^\star) < 0$ only when the predicted label on $x$ is $t^\star$. The choice of $g_{obj}$ affects the effectiveness of this attack method, and the authors found $g_{obj}(x, t^\star) = \max\{\max\{z_i; i \neq t^\star\} - z_{t^\star}, -\kappa\}$ works well when $z = f(x)$ and $\kappa$ is a constant for controlling the 'confidence' of the prediction. They used gradient descent along with binary search for the hyperparameter $c$ in the objective function to find adversarial examples. Note that this method produces *unbounded* perturbations; that is, this method aims to find the smallest possible $r$ for a given input sample $x$, but does not essentially bound the size of $r$ as in FGSM or PGD.

Although the above-mentioned attacks were originally developed for attacking image classifiers, other studies investigated adversarial examples for different tasks, such as object detection [51], automatic speech recognition [7], and reinforcement learning [22]. Furthermore, Papernot et al. showed that adversarial examples can be crafted for a variety of machine learning models, not limited to linear classifiers or deep neural networks, such as nearest-neighbor methods and decision trees [34].

The widespread existence of adversarial examples in machine learning motivated research on improving adversarial robustness of models. In other words, obtaining higher accuracy on adversarial examples ('adversarial accuracy' in short), not only on clean input samples, has become an important factor in designing machine learning system. Among many notable works, Goodfellow et al. and Madry et al. proposed to train deep neural networks on examples generated by attack methods ('adversarial training') as a defense against adversarial examples [15, 29]. While adversarial training can be thought as an implicit regularization on models using data augmentation or as an optimization technique for solving min-max problem, other approaches for achieving robustness often add explicit regularization terms to a loss function. For example, Ross and Doshi-Velez and Jakubovitz and Giryes adds a penalty term for the size of gradients (i.e., gradients of a loss function with respect to inputs or Jacobian of output logits with respect to inputs) to cross-entropy loss to enhance robustness for classification tasks [39, 24].

These advances in attack methods provided practical methodologies for evaluating adversarial robustness of deep neural networks for many empirical investigations. For example, Su et al. benchmarked popular deep neural network architectures for ImageNet classification challenge [40] for their adversarial robustness, using the above-mentioned attack methods such as FGSM and C&W attack [45]. However, understanding adversarial robustness of complex deep neural networks can be challenging, as the next part explains.

## 1.3 Difficulties of evaluating adversarial robustness

Suppose we want to know whether an adversarial example $x' = x + r$ exists in the $\epsilon$-ball in $L_p$ norm around an input sample $x$ (i.e., $\|r\|_p \leq \epsilon$) for a deep neural network $f(\cdot)$. If the distribution for input samples is discrete and low-dimensional, we can consider a brute-force method for every possible perturbation $r$ with the size less than $\epsilon$ to check the existence of adversarial perturbations. However, such brute-force method cannot scale to continuous (or discrete but large number of possible values, such as images encoded in 8-bit) and high-dimensional distributions, which are typical for applications using deep neural networks.

One computationally feasible approach to solve this problem is to use attack methods, such as FGSM or PGD discussed in Section 1.2, to generate a perturbation $r$, and empirically test whether this perturbation can alter the prediction of $f(\cdot)$. The benefit of this empirical approach is that attack methods are computationally efficient (e.g., FGSM only requires one additional back-propagation) and scalable to complex deep neural networks as far as those models are differentiable. Additionally, these attack methods can be easily accelerated using modern Graphic Processor Units and popular numerical frameworks that support automatic differentiation, such as PyTorch [37]. However, it is important to note that even if $r$ generated by attack methods does not succeed in fooling the model, the real adversarial perturbation $r^\star$ can exist; in such case, attack methods simply fail to find $r^\star$. Therefore, when extending this approach to the entire dataset, we can obtain the *upper bound* of adversarial robustness of the model $f$ on that dataset. Section 1.3.1 further explains when the discrepancy between the upper bound obtained by this approach and the actual adversarial robustness goes large, making evaluation difficult.

Another approach is to directly formulate this problem as an optimization problem, and verify whether an adversarial perturbation exists using optimization solvers. While this approach can guarantee the existence of adversarial perturbation, thus has potential to prove exact adversarial robustness, an important challenge for this approach is the non-convexity of deep neural networks. Typically, such non-convexity is relaxed, and the resulting approach approximates the *lower bound* of adversarial robustness. Section 1.3.2 introduces notable works related to this approach.

### 1.3.1 Gradient masking affects attack methods

Since first-order attack methods rely on accurate computation of gradients to generate perturbations, sources of inaccurate gradient computation can inflate accuracy against examples generated by attack methods. Consequently, defense methods relying on those sources can appear to improve adversarial robustness. However, in-depth investigation into those methods reveals that they do not fundamentally enhance adversarial robustness.

Papernot et al. showed that Defensive Distillation [35], a defense method that retrains a model using knowledge distillation [21], is vulnerable to *transferred* perturbations that are generated from another model (also known as a 'black-box' attack), despite high accuracy against perturbations generated using attack methods on itself ('white-box' attack in short) [36]. Carlini and Wagner identified that Defensive Distillation induces output logits to have large 'margin', saturating the value of cross-entropy loss to zero numerically [6]. In such case, numerically computed gradients are not accurate, hurting the performance of attack methods. Papernot et al. called this phenomenon *gradient masking*, where empirical accuracy against attack methods is inflated due to inaccurate computation of gradients.

Furthermore, Athalye et al. found that many defense methods relied on gradient masking, and proposed new attack methods to circumvent such defense methods [2]. They identified that those defense methods rely on three mechanisms to obfuscate gradients: 1) stochastic gradients, where defense methods use random noises or inference-time drop-out [11] that cannot be deterministically predicted when computing gradients, 2) gradient shattering, where non-differentiable operations, often in the form of pre-processing functions [17], are in the computation graph for back-propagation, and 3) gradient vanishing and exploding, where defense methods result in extremely deep models when their computations are unrolled, as in the case of using generative models [41]. On the other hand, they showed that adversarial training does not rely on any of these three mechanisms, thus its benefit on adversarial robustness is not inflated.

However, even adversarial training can suffer from gradient masking in a more subtler manner. Tramèr et al. found that using FGSM, a single-step attack method, for adversarial training results in a degenerate solution that increases local curvature around input sample points [49]. Although computation of gradients itself is unaffected by this degeneracy, a loss function (e.g., cross-entropy loss for classification tasks) is not apt for local first-order approximation due to high curvature. Thus, while accuracy against examples generated by FGSM goes high, the model is still vulnerable to other attack methods and black-box attacks. To resolve a degeneracy, Tramèr et al. proposed to add a random perturbation before computing gradients when using FGSM for adversarial training, naming it Random-FGSM (R-FGSM):

$$u \sim \mathcal{N}(0, I) \tag{1.7}$$

$$r = \alpha \cdot \texttt{sign}(u) \tag{1.8}$$

$$r \leftarrow (\epsilon - \alpha) \cdot \texttt{sign}(\frac{\partial l(f(x+r), t)}{\partial r}) + r \tag{1.9}$$

where notations follow those of Eq (1.3), and $\alpha$ is a hyperparamter to set relative step size for the random initialization.

These examples of gradient masking illustrate difficulties of empirically evaluating adversarial robustness using attack methods. The upper bound of adversarial robustness is inflated by gradient masking, and taking the upper bound as a proxy for true adversarial robustness results in large discrepancy in these examples. This thesis builds on these exemplary works on understanding attack methods and implications of using empirical accuracy as adversarial robustness. As we show in subsequent chapters, we extend gradient masking beyond defense methods that make specific modifications to training and inference, and show that this phenomenon exists even for conventionally trained deep neural networks.

### 1.3.2 Verification

Finding an adversarial perturbation can be formally described as an optimization problem, and many works used optimization techniques to evaluate adversarial robustness [50, 38, 44, 48]. Since non-linear activation functions popularly used in deep neural networks violate convexity, this optimization-based approach often relaxes non-convex elements to be convex. For example, Wong and Kolter relaxed Rectified Linear Unit (ReLU) activation function as a bounded convex set that defines the outer bound of ReLU, and applied Linear Programming (LP) to solve the resulting convex optimization problem [50]. Often, using relaxation results in proving the lower bound of adversarial robustness, because relaxed sets serve as the outer bound of the actual space that adversarial perturbations can exist, as Wong and Kolter did.

Instead of approximating the lower bound, Tjeng et al. chose to use Mixed-Integer Linear Programming (MILP) to verify the existence of an adversarial perturbation for deep neural networks using ReLU as an activation function [48]. They utilized the fact that ReLU is piecewise-linear, thus exact adversarial robustness can be obtained (assuming computational complexity and numerical stability are not concerns) using MILP.

A benefit of these verification methods is that the provable lower bound (or often the exact value) of adversarial robustness is obtainable, thus resolving the problem of overestimated adversarial robustness when using the empirical methods (i.e., using first-order attack methods to generate perturbations). However, they are typically more computationally demanding than attack methods, especially as the lower bound becomes more tighter, and scaling them to large-scale datasets and models is challenging. Consequently, the empirical methods are popularly used to benchmark and numerically experiment adversarial robustness in practice; examples include comparing different deep neural network architectures [45, 10], and model compression techniques [28, 52], which are discussed in relation to hardware energy-efficiency in Section 1.4.

## 1.4 Relationship to energy-efficient deep neural networks

Deep neural networks are computationally demanding compared to other machine learning algorithms: deep neural networks typically have large number of parameters, and the number of multiply-accumulate operations for a single inference can be in the order of $10^9 \sim 10^{10}$ [46]. Thus, reducing the computational footprint of deep neural networks is essential for resource-constrained real-world applications, especially when they run on embedded hardware platforms. One popular approach to achieve this is to compress deep neural networks by pruning parameters and quantize the number of bits used for computations, such that both memory and computational footprint can be reduced [18].

While those compression methods are successful in reducing computations without sacrificing clean accuracy, their influence on adversarial robustness is not well understood. First, several studies postulate that deep neural networks with larger capacity, as in more number of neurons per layer, have better adversarial robustness [29, 10]. Furthermore, recent studies claim that model compression techniques might degrade adversarial robustness, in the context of activation quantization [28] and weight pruning [42]. These observations suggest that achieving energy-efficient deep neural networks might be at odds with adversarial robustness, motivating further investigation into what drives empirically measured adversarial robustness low for compressed deep neural networks. Chapter 3 explains these prior works further, and shows how overestimation of adversarial robustness affects these analyses when using the empirical methods.

## 1.5 Thesis overview and contributions

This thesis investigates cases when bounded first-order attack methods, such as FGSM and PGD, fail to find adversarial perturbations. Major contributions are

- Identify three cases when accuracy against examples generated by attack methods is overestimated due to superficial reasons, not indicating true robustness.
- Propose compensation methods that can be easily combined with existing attack methods for each of those cases, providing more precise metric for empirical evaluation of adversarial robustness.
- Show how overestimation of adversarial robustness affects conventionally trained deep neural networks, not limited to defense methods, and demonstrate the impact on practically important studies, including the relationship between the model capacity and adversarial robustness.
- Show how the three cases we identify can explain the gap between the lower bound obtained with verification approaches and the upper bound obtained by the attack methods.

This thesis is organized as follows. Chapter 2 analyzes when bounded first-order attack methods fail to find adversarial perturbations, and provides compensation methods when they fail for superficial reasons. Chapter 3 provides case studies to illustrate how overestimated adversarial robustness affects wide range of deep neural networks. Chapter 4 delves into the gap between the upper bound empirically obtained with attack methods and the lower bound obtained by verification methods. Chapter 5 summarizes our contribution, and concludes this thesis with a short remark on future work.

# Chapter 2

# Analysis on Failure Cases of Attack Methods

This chapter analyzes three cases when untargeted, bounded first-order attack methods fail to find adversarial perturbations: 1) cross-entropy loss becoming zero due to numerical saturation of the loss value, 2) innate non-differentiability of deep neural networks, induced by certain activation functions, and 3) certain training conditions increasing the number of iterations used by iterative attack methods (i.e., PGD) required to find adversarial perturbations. For each case, we provide compensation methods that can be easily integrated with existing attack methods.

We describe these phenomena and analyze their causes using experimental examples. For examples used in this chapter, we consider following deep neural network architectures: a *Simple* model with 4 convolutional layers and 2 fully-connected layers with ReLU activations, a *Simple-BN* model that has a batch normalization [23] after each convolutional layer in a Simple model, and a *WideResNet (WRN)* model with 28 layers [54]. Deep neural networks are trained on CIFAR-10 dataset [25] for the image classification task, otherwise stated. Details of these architectures and training hyperparameters are presented in Appendix A.

## 2.1 Zero loss

Cross-entropy loss gets closer to zero when the logit corresponding to the correct label has larger gap with other logits. Since logits take real values in the range $(-\infty, \infty)$, cross-entropy loss is mathematically larger than zero unless other logits except for the largest one are all $-\infty$. However, standard computing has limited precision to represent numbers, and logarithmic and exponential functions involved in cross-entropy loss can numerically saturate small values close to zero. Numerically, when the value of loss becomes zero, gradients computed on that loss is not meaningful. This becomes a problem for first-order attack methods that rely on gradients to generate perturbations. This section analyzes how this phenomenon exists in conventionally trained deep neural networks, and proposes compensation methods to resolve this issue.

### 2.1.1 Analysis

**Saturation of logarithmic and exponential functions in popular libraries**
Cross-entropy loss combines softmax function that involves exponential with negative log-likelihood. While standard 32-bit floating point format can represent wide range of real values, cascading exponential and logarithmic functions can saturate the resulting value in popular numerical computing libraries in Python, such as NumPy [33] and PyTorch [37]. For example, consider the following expression that is similar to cross-entropy loss:

$$c = -\log(\frac{\exp a}{\exp a + \exp b}) \tag{2.1}$$

The value of $c$ is larger than 0 unless $b = -\infty$, thus $\exp b = 0$. However, in NumPy and PyTorch, $c$ becomes *numerically* zero when $a - b \gtrapprox 18$.

In case of actual cross-entropy loss computation for multi-class classification, the divisor becomes the summation of exponential of logits, and the dividend becomes the exponential of the ground truth label's logit. Therefore, when the gap between logits corresponding to the ground truth label and other labels increase, numerical value of cross-entropy loss can become zero. This numerical saturation problem can more frequently occur when one uses low precision for faster training or inference, such as half precision (16-bit) floating point format.

**Example**

We illustrate how the numerical saturation of cross-entropy loss occurs in a conventionally trained deep neural network. Consider a deep neural network with a Simple architecture described in the beginning of this chapter. We train this model without any explicit regularization technique.

First, we measure accuracy of this model on perturbed examples generated by FGSM ($\epsilon = \frac{8}{255}$ in $L_\infty$ norm), which is 14.04%. We characterize the value of cross-entropy loss, the size of gradients (in $L_2$ norm), and the 'margin' of logits (the gap between logits corresponding to the most likely and the second most likely label) for this model, and separately analyze for input samples on which the attack succeeds and fails (Table 2.1). Observe that the average value of cross-entropy loss on input samples on which the attack fail is smaller (0.0011), and the average logit margin on those samples is 18.73. Note that this large value of logit margin can result in the numerical saturation of cross-entropy loss, as discussed in the above part. However, the average logit margin on input samples on which the attack succeeds is 7.20, and the size of gradients is large for these samples. Therefore, we can suspect that zero loss occurs for the samples on which the attack fails in this example.

Table 2.1: Characterizing input samples on which FGSM ($\epsilon = \frac{8}{255}$ in $L_\infty$ norm) succeeds and fails for the value of loss, size of gradients, and logit statistics including margin and variance, for the example deep neural network described in Section 2.1.1.

| | Attack succeed | Attack fail |
|---|---|---|
| Loss | 0.0643 | 0.0011 |
| Gradient | 2.0686 | 0.0339 |
| Logit margin | 7.20 | 18.73 |
| Logit variance | 73.44 | 142.19 |

Second, we measure accuracy of this model on a *black-box* attack using the model with the same architecture, but independently trained using weight decay regularization with strength $5 \times 10^{-4}$, as a substitute model on which FGSM (same $\epsilon$ as above) generates perturbations. The accuracy in this case is 13.22%, lower than 14.04% of above. A black-box attack resulting in lower accuracy than a white-box attack signals gradient masking; it represents that using the exact gradient of the model is somehow less effective than directions not related to the gradient (e.g., perturbations transferred from a substitute model) [2].

Then, for more in-depth inspection, we visualize how the value of cross-entropy loss changes as perturbations are added to the original input sample (Fig 2-1). The input sample is randomly chosen among samples on which the white-box FGSM fails to find adversarial perturbations. The sample used in Fig 2-1 has numerically zero cross-entropy loss. Observe that increasing the size of perturbation along the direction of this model's own gradient does not increase loss, but orthogonal directions (e.g., gradient obtained from a substitute model) easily increase loss. This example shows how gradients computed on this model do not result in a meaningful perturbation direction when the value of loss is zero (or close to zero).

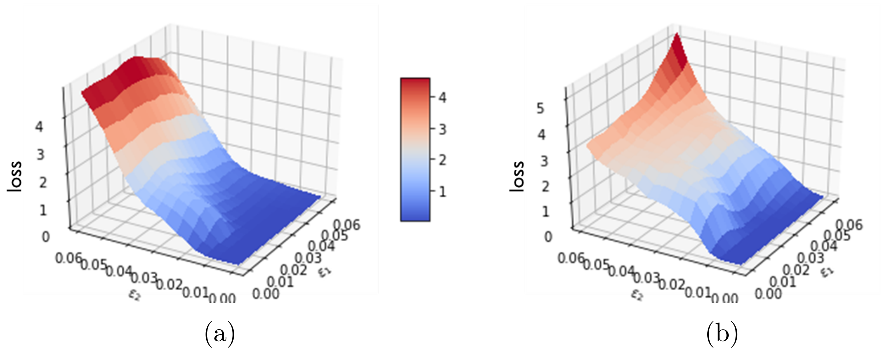


Figure 2-1: Visualization of the value of cross-entropy loss (z-axis) evaluated for points $x^* = x + \epsilon_1 \cdot \texttt{sign}(g) + \epsilon_2 \cdot \texttt{sign}(g')$, where $g$ is gradients of loss with respect to the input sample computed on this model, and $g'$ is (a) gradients computed from the surrogate model used for the black-box attack, and (b) gradients computed using the second most likely class as the target label when computing cross-entropy loss instead of the ground truth label.

### 2.1.2 Compensation methods: increasing loss

Since the cause for inaccurate gradient computation is small value of loss that numerically saturates to zero, ensuring the value of cross-entropy loss to be sufficiently large can circumvent this problem.

**Rescaling logits**

First, we consider rescaling logits $z$ by a constant $T$. Then, cross-entropy loss is computed as $l(\frac{f(x)}{T}, t)$: when $T > 1$, absolute values of logits decrease, leading to the larger value of cross-entropy loss. The attack methods, such as FGSM or PGD, can be applied as usual on this loss.

**Targeted objective**

Second, suppose we compute cross-entropy loss using *another* label $t'$ as a target instead of the ground truth label $t$. Naturally, this turns the attack to be *targeted* towards the label $t'$. However, note that although we change $t$ to $t'$, our objective is still finding an *untargeted* perturbation. Since the value of loss with respect to the ground truth label $t$ is small, changing the target label when computing loss will increase the value of loss.

### 2.1.3 Impact on measurement of robustness

We test the effectiveness of the proposed compensation methods on wide range of deep neural networks and datasets. First, the results on the model used in the above example is shown in Table 2.2. We find that changing the target label $t'$ to be the second most likely label gives the best compensation (e.g., brings down accuracy against perturbed examples the most).

Then, we compare different choices of architectures for how they are affected by the zero-loss phenomenon (Table 2.3). Observe that the model trained with weight decay regularization is less affected by this phenomenon, as manifested by the smaller gap in accuracy against the baseline and the compensated attacks using the targeted objective with $t'$ as the second most likely classes. This observation can be explained by weight decay's effect on weight matrices of the model; weight decay penalizes the Frobenius norm of weight matrices, inducing weights to have small magnitude. Naturally, the magnitude of output logits decrease when weights that are used to

Table 2.2: Accuracy against perturbed examples generated by different attack types that are compensated for the zero loss phenomenon (in %). We sweep a constant $T$ for rescaling logits, and target labels $t'$ for targeted loss computation; 'random' sets $t'$ to be randomly sampled among all possible classes, 'least' and 'second' indicate the least-likely and the second most-likely classes, respectively. We also state the gap between accuracies against the baseline and the compensated attacks.

| Evaluation type | | Baseline | Rescaling | | | Targeted | | | Gap |
|---|---|---|---|---|---|---|---|---|---|
| | | | $T = 10$ | $T = 50$ | $T = 100$ | Random | Least | Second | |
| Clean | | 84.75 | - | - | - | - | - | - | - |
| $L_\infty, \epsilon = \frac{4}{255}$ | FGSM | 19.50 | 13.77 | 15.24 | 15.38 | 14.59 | 17.62 | **10.79** | 8.71 |
| | R-FGSM | 29.81 | 30.35 | 30.44 | 30.46 | 30.02 | 30.73 | **29.11** | 0.70 |
| | PGD | 2.55 | 2.06 | 2.26 | 2.26 | 2.03 | 2.67 | **1.68** | 0.87 |
| $L_\infty, \epsilon = \frac{8}{255}$ | FGSM | 14.04 | 3.51 | 4.40 | 4.51 | 4.20 | 6.52 | **3.08** | 10.96 |
| | R-FGSM | 15.03 | 9.58 | 11.29 | 11.50 | 11.12 | 13.78 | **7.23** | 7.80 |
| | PGD | 0.24 | 0 | 0.01 | 0.01 | 0.01 | 0.08 | **0** | 0.24 |
| $L_2, \epsilon = 0.5$ | FGM | 23.18 | 20.45 | 21.46 | 21.54 | 20.77 | 22.67 | **18.10** | 5.08 |
| | R-FGM | 39.64 | 40.09 | 40.23 | 40.26 | 39.81 | 40.33 | **39.08** | 0.56 |
| | PGD | 6.67 | 5.06 | 5.68 | 5.75 | 5.25 | 6.85 | **3.83** | 2.84 |
| $L_2, \epsilon = 1.0$ | FGM | 17.75 | 7.75 | 9.37 | 9.56 | 9.03 | 12.32 | **6.14** | 11.61 |
| | R-FGM | 19.84 | 17.88 | 18.96 | 19.04 | 18.36 | 20.32 | **15.24** | 4.60 |
| | PGD | 3.82 | 0.11 | 0.28 | 0.28 | 0.25 | 1.39 | **0.03** | 3.79 |

produce those logits have small magnitude, reducing the margin in logits. Other remark is that the model with larger 'width', as in the number of neurons per layer, or with batch normalization suffers more from the zero-loss phenomenon.

Finally, we benchmark how this zero-loss phenomenon affects wide range of deep neural network architectures and datasets (Table 2.4, Column **Zero loss**). We compare accuracy against perturbed examples generated by the baseline and the compensated attacks, with targeting to the second most likely labels as the compensation method for the zero-loss phenomenon. We find that this phenomenon exists among diverse conventionally trained deep neural networks.

### 2.1.4 Connection to other attack methods

Gradient masking induced by zero-loss was previously observed by Carlini and Wagner. Carlini and Wagner found that Defensive Distillation [35] increases the margin in logits, as Defensive Distillation uses temperature softmax (essentially same as rescaling logits) during retraining [6]. Carlini and Wagner showed that the value of cross-entropy loss saturates to zero for the models using Defensive Distillation, but their

Table 2.3: The gap in accuracy against perturbed examples generated by attacks (in %) when compensating for the zero loss phenomenon by changing target labels $t'$ to be the second most likely classes. 'No Reg' represents a deep neural network with Simple architecture trained without explicit regularization (as in the example). 'Weight Decay' represents the same model as 'No Reg', but trained with weight decay of $5 \times 10^{-4}$. 'Width x4' represents a model with 4 times more neurons per layer compared to 'No Reg'. 'Batch Norm' indicates a Simple-BN model, which is same as 'No Reg' except for batch normalization following each convolutional layer.

| Evaluation type | | No Reg | Weight Decay | Width x4 | Batch Norm |
|---|---|---|---|---|---|
| $L_\infty, \epsilon = \frac{4}{255}$ | FGSM | 8.71 | 4.48 | 19.48 | 18.73 |
| | R-FGSM | 0.70 | 0.59 | 5.78 | 9.36 |
| | PGD | 0.87 | 0.19 | 4.51 | 6.16 |
| $L_\infty, \epsilon = \frac{8}{255}$ | FGSM | 10.96 | 8.12 | 18.43 | 19.99 |
| | R-FGSM | 7.80 | 3.89 | 17.75 | 15.68 |
| | PGD | 0.24 | 0.15 | 1.20 | 2.05 |
| $L_2, \epsilon = 0.5$ | FGM | 5.08 | 2.41 | 16.54 | 16.78 |
| | R-FGM | 0.56 | 0.63 | 2.72 | 5.64 |
| | PGD | 2.84 | 0.57 | 15.29 | 12.68 |
| $L_2, \epsilon = 1.0$ | FGM | 11.61 | 7.91 | 27.34 | 21.02 |
| | R-FGM | 4.60 | 1.73 | 16.24 | 15.51 |
| | PGD | 3.79 | 2.14 | 19.62 | 9.45 |

attack method is capable of finding adversarial perturbations even for those models. Our work contributes further by finding that the zero-loss phenomenon also affects conventionally trained models, not confined to specific defenses that intentionally caused this phenomenon.

We also analyze how the zero-loss phenomenon affects C&W attack depends on the choice of $g_{obj}$ that serves as an alternative loss function. We find that choosing $g_{obj}$ that directly uses logits, instead of softmaxed logits or the value of cross-entropy loss, makes C&W attack more resilient to the zero-loss phenomenon. For example, the default choice of $g_{obj}$,

$$g_{obj}(x, t) = \max\{-\max\{z_i; i \neq t\} + z_t, -\kappa\} \tag{2.2}$$

achieves 100% success rate of finding adversarial perturbations for the example deep neural network using Simple architecture described above. Note that we flipped the signs of $\max\{z_i; i \neq t\}$ and $z_t$ from those in the objective function introduced in Chapter 1, since we are considering an *untargeted* attack where $t$ is the ground truth

Table 2.4: Accuracy of neural networks against perturbed examples generated by the first-order attack methods in the order of FGSM/R-FGSM/PGD for $\epsilon = \frac{4}{255}$ in $L_\infty$ norm. We apply compensation methods for zero loss and innate non-differentiability discussed in Sections 2.1 and 2.2. As compensation methods are applied in cascading manner (i.e., samples on which baseline attacks fail are subjected to compensation methods), we set the number of random starts for baseline R-FGSM and PGD to be same as the total number of evaluations compensation methods use for fair comparison. All models are trained without explicit regularization on the specified dataset.

| Dataset | Architecture | Accuracy (%) | | | | |
|---|---|---|---|---|---|---|
| | | Clean | Attack Baseline | Zero loss | Non-differentiability | Both |
| CIFAR-10 | Simple | 84.75 | 19.50 / 29.81 / 2.55 | 10.79 / 29.11 / 1.68 | 18.41 / 29.17 / 2.54 | 8.74 / 28.31 / 1.67 |
| | Simple-BN | 87.09 | 28.66 / 28.39 / 6.26 | 9.93 / 19.03 / 0.10 | 26.92 / 28.05 / 6.11 | 6.89 / 17.93 / 0.07 |
| | WRN 28 | 91.65 | 21.90 / 20.79 / 0.02 | 11.34 / 13.46 / 0 | 15.87 / 19.76 / 0.02 | 5.94 / 11.25 / 0 |
| SVHN | Simple-BN | 94.29 | 28.60 / 43.36 / 2.80 | 23.81 / 42.21 / 2.59 | 24.35 / 42.60 / 2.60 | 19.10 / 41.31 / 2.38 |
| | WRN 28 | 95.42 | 49.74 / 58.22 / 4.06 | 45.46 / 57.04 / 3.73 | 39.69 / 56.70 / 3.79 | 34.30 / 55.05 / 3.69 |
| TinyImageNet | VGG 11 | 50.32 | 11.32 / 20.16 / 7.94 | 7.44 / 16.56 / 4.10 | 11.30 / 20.62 / 7.98 | 7.44 / 16.86 / 4.22 |
| | VGG-BN 11 | 50.72 | 4.16 / 11.68 / 0.64 | 3.22 / 10.70 / 0.48 | 3.82 / 11.80 / 0.68 | 3.06 / 11.00 / 0.48 |
| | WRN 50 | 57.24 | 19.78 / 23.40 / 2.02 | 3.36 / 9.86 / 0.40 | 18.24 / 23.48 / 1.58 | 2.88 / 10.02 / 0.36 |

label. Also, we set $\kappa = 0$ for this experiment. However, when we choose $g_{obj}$ to be

$$g_{obj}(x, t) = 1 - l(f(x), t) \tag{2.3}$$

the attack succeeds only for 88.62% of input samples (in other words, the model shows 11.38% of accuracy against perturbed examples generated by this attack configuration). We observe that input samples on which the later $g_{obj}$, which directly operates on cross-entropy loss $l(z = f(x), t)$, fails to find adversarial perturbations have small value of cross-entropy loss, leading to the zero-loss phenomenon. Although we do not investigate unbounded attacks, such as C&W attack, in depth, this analysis reveals that the choice of the optimization objective function affects the vulnerability of attack methods against the zero-loss phenomenon, and that the objective function that directly operates on logits is more beneficial to prevent the zero-loss phenomenon.

## 2.2 Innate non-differentiability

During back-propagation, gradients cannot be computed for non-differentiable functions if they are in the computation graph. When non-differentiable functions are used for defense methods, such as in the form of pre-processing operation [17], they make the computation of gradients inaccurate and result in a form of gradient masking known as *gradient shattering* [2]. While gradient shattering has been previously identified, we show that innate non-differentiability in popular deep neural network architectures, such as ReLU activation and MaxPool, can result in inaccurate computation of gradients as well. In this section, we analyze how ReLU and MaxPool can cause gradient shattering, and propose to extend Backward Pass Differentiable Attack (BPDA) [2] for approximating gradients through ReLU and MaxPool.

### 2.2.1 Analysis

#### Switching of ReLU and MaxPool

ReLU is a piecewise linear function:

$$\text{ReLU}(x) = \max\{x, 0\} \tag{2.4}$$

and is only non-differentiable at $x = 0$. During usual training where back-propagation is used, this non-differentiability rarely becomes a problem since neurons do not become exactly zero. However, this non-differentiability can be concerning when computing gradients for attacks.

During back-propagation, gradients are not passed through negative-valued neurons, as they are set to zero by ReLU. Therefore, the computed gradients are actually the gradients of the sub-model comprising positive-valued neurons. For training, this property results in sparse gradients, but non-differentiability at zero does not become a problem itself since the values of neurons are not exactly zero. However, for attack methods, we add perturbations using these gradients and forward-propagate them to evaluate the success of attack methods. These added perturbations can “switch" some of previously negative-valued neurons to take positive values, and vice versa. In such case, the sub-model that contributes to the final prediction changes, as effective neurons that are non-zero after ReLU are different due to added perturbations, and the computed gradients are no longer accurate.

MaxPool chooses the maximum-valued neuron among all neurons in a given window. For example, if MaxPool uses a window size of 2-by-2,

$$\text{MaxPool}(x_{i,j}, x_{i+1,j}, x_{i,j+1}, x_{i+1,j+1}) = \max\{x_{i,j}, x_{i+1,j}, x_{i,j+1}, x_{i+1,j+1}\} \tag{2.5}$$

and gradients only flow through the maximum-valued neuron in this window. Thus, MaxPool has similar “switching" problem as ReLU, when added perturbations changes the maximum-valued neuron in a given window to another neuron that was previously ignored when computing gradients.

**Example**

We measure how frequently switching occurs when perturbations generated by attack methods are added to input samples. We use the same deep neural network used for the example in Sec 2.1 for the analysis here. When perturbations generated by FGSM ($\epsilon = \frac{8}{255}$ in $L_\infty$ norm) are added to input samples, we observe that 7.73% of neurons using ReLU as an activation function switch, and that 20.60% of neurons resulting from MaxPool operation switch.

### 2.2.2 Compensation methods: approximating gradients

Athalye et al. proposed Backward Pass Differentiable Attack (BPDA) to approximate gradients through non-differentiable functions when defense methods use such functions [2]. When $h(x)$ is a non-differentiable function, and $f_1(x)$ and $f_2(x)$ are differentiable functions that are cascaded with $h(x)$ as $f_1(h(f_2(x))$ in the computation graph, BPDA uses a differentiable function $g(x) \approx h(x)$ and uses gradients through $g(x)$ during back-propagation, although forward-propagation uses $h(x)$ [2]. Thus, when computing gradients through $f_1$, it uses the output of $h(x)$ computed during forward-propagation, while gradients through $h(x)$ is replaced by gradients through $g(x)$.

While Athalye et al. used BPDA to break defense methods relied on explicitly non-differentiable operations, we adopt BPDA to approximate gradients through ReLU and MaxPool. In the following part, we discuss differentiable approximation functions for ReLU and MaxPool.

**ReLU**

Softplus function smoothly approximates ReLU as

$$\text{softplus}(x) = \frac{1}{\beta} \cdot \log(1 + \exp(\beta \cdot x)) \tag{2.6}$$

where $\beta$ scales how similar softplus function gets to ReLU. For example, as $\beta \to \infty$, softplus will be more similar to ReLU, with sharper transition around zero.

Exponential Linear Unit (ELU) [9] models the leakage when an input is negative, instead of setting the output as zero. Specifically,

$$\mathrm{ELU}(x) = \begin{cases} x & x > 0 \\ \exp x - 1 & x \leq 0 \end{cases} \tag{2.7}$$

and we can easily see that ELU is left-differentiable and right-differentiable at $x = 0$, and they are the same:

$$\lim_{x \to 0^-} \frac{(\exp x - 1) - (\exp 0 - 1)}{x} = \lim_{x \to 0^+} \frac{x - 0}{x} = 1 \tag{2.8}$$

Thus, the first-order derivative for ELU exists at $x = 0$.

Finally, we consider Continuously-differentiable ELU (CELU) [3] that modified ELU as

$$\mathrm{CELU}(x) = \begin{cases} x & x > 0 \\ \beta \cdot (\exp \frac{x}{\beta} - 1) & x \leq 0 \end{cases} \tag{2.9}$$

where $\beta$ controls the slope of exponential decay when $x < 0$. As $\beta \to 0$, CELU gets closer to ReLU. Note that when $\beta = 1$, CELU is essentially same as ELU.

**MaxPool**

MaxPool can be thought as $L_\infty$ norm pooling, and natural approximation for MaxPool would be $L_p$ norm pooling with sufficiently large $p$. That is, for pooling operation in a 2-by-2 window as in Eq (2.5),

$$L_p\mathrm{Pool}(x_{i,j}, x_{i+1,j}, x_{i,j+1}, x_{i+1,j+1}) = \sqrt[p]{x_{i,j}^p + x_{i+1,j}^p + x_{i,j+1}^p + x_{i+1,j+1}^p} \tag{2.10}$$

is used to approximate gradients for MaxPool.

Table 2.5: Accuracy against perturbed examples generated by attacks (in %) of the example deep neural network used for Sec 2.2.1, compensated for innate non-differentiability using BPDA. We investigate three differentiable functions for substituting ReLU: Softplus ($\alpha = 2$, thresholded at 2), CELU ($\alpha = 2$), and ELU while fixing $L_p$-norm pool's $p$ to be 5. Then, we sweep for $p$ by fixing ReLU substitute function to be the one achieved the best performance.

| Evaluation Type | | Baseline | ReLU Substitute | | | $L_p$-norm Pool | | | Gap |
|---|---|---|---|---|---|---|---|---|---|
| | | | Softplus | CELU | ELU | $p = 2$ | $p = 5$ | $p = 10$ | |
| Clean | | 84.75 | - | - | - | - | - | - | - |
| $L_\infty, \epsilon = \frac{4}{255}$ | FGSM | 19.50 | **18.41** | 18.93 | 18.68 | 18.58 | **18.41** | 18.44 | 1.09 |
| | R-FGSM | 29.81 | **29.17** | 30.51 | 29.92 | 30.26 | 29.17 | **29.15** | 0.66 |
| | PGD | 2.55 | 2.54 | 2.54 | **2.50** | 2.57 | 2.50 | **2.47** | 0.08 |
| $L_\infty, \epsilon = \frac{8}{255}$ | FGSM | 14.04 | 12.33 | **12.06** | 12.15 | **11.85** | 12.06 | 12.11 | 2.19 |
| | R-FGSM | 15.03 | 14.88 | 14.95 | **14.72** | 15.03 | **14.72** | 14.78 | 0.31 |
| | PGD | 0.24 | 0.21 | **0.15** | 0.16 | 0.22 | 0.15 | **0.13** | 0.11 |
| $L_2, \epsilon = 0.5$ | FGM | 23.18 | **21.07** | 22.43 | 22.22 | 21.66 | 21.07 | **20.59** | 2.59 |
| | R-FGM | 39.64 | **38.85** | 40.05 | 39.72 | 39.77 | 38.85 | **35.67** | 3.97 |
| | PGD | 6.67 | 6.68 | 6.57 | **6.55** | 6.56 | **6.55** | 6.64 | 0.12 |
| $L_2, \epsilon = 1.0$ | FGM | 17.75 | 16.07 | **16.04** | 16.25 | 15.87 | 16.04 | **15.78** | 1.97 |
| | R-FGM | 19.84 | **18.69** | 19.66 | 19.39 | 19.06 | 18.69 | **17.98** | 1.86 |
| | PGD | 3.82 | 3.90 | 3.64 | **3.63** | **3.58** | 3.63 | 3.67 | 0.24 |

### 2.2.3 Impact on measurement of robustness

First, we test the effectiveness of using BPDA to generate more accurate perturbations for adversarial attacks. Table 2.5 shows the example model's accuracy against perturbed examples generated by different attack methods and BPDA approximation functions for ReLU and MaxPool. We find that using BPDA can decrease accuracy against the perturbed examples subtly, usually by 1-3% for this example model. Generally, we observe that single-step attacks, such as FGSM and R-FGSM, are more affected by using BPDA compared to PGD, which is an iterative method. This can be understood from the fact that PGD has smaller effective step size $\alpha$ compared to $\epsilon$ of single-step attacks. If the step size is small, then fewer number of neurons with ReLU or MaxPool will switch.

Furthermore, we benchmark how innate non-differentiability affects accuracy against perturbed examples generated by attack methods for various architectures and datasets, for the same setting as in Table 2.4 (Column **'Non-differentiability'**). For this analysis, we fix the approximation functions for BPDA as Softplus for ReLU and $p = 5$ for MaxPool. Similar as the example model discussed above, compensating for the non-differentiability phenomenon with BPDA gives subtle differences (1-3%) in the accuracy for various architectures and datasets. However, there exist certain cases where the difference is more significant, such as a WRN model trained on SVHN for which using BPDA gives 10% difference in accuracy against FGSM. Also, note that combining the compensation methods for both the zero loss and the non-differentiability phenomena gives better attack success rate (Table 2.4, Column **'Both'**).

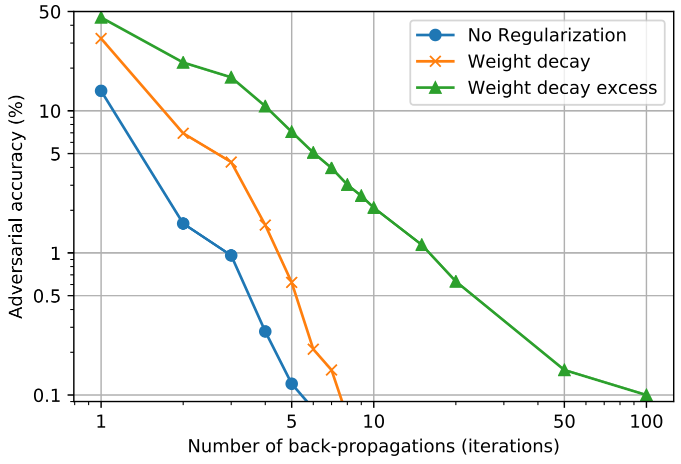
## 2.3 Requiring more iterations

We observe an interesting phenomenon that training a deep neural network with certain regularization techniques, such as weight decay, increases the number of iterations used by PGD to find adversarial perturbations. That is, the apparent accuracy of such model against perturbed examples generated by PGD with *small, limited* iterations appears to be higher than that of another model not using those regularization techniques. In this section, we provide examples of this phenomenon, and suggest using the second-order information to reduce the number of iterations required.

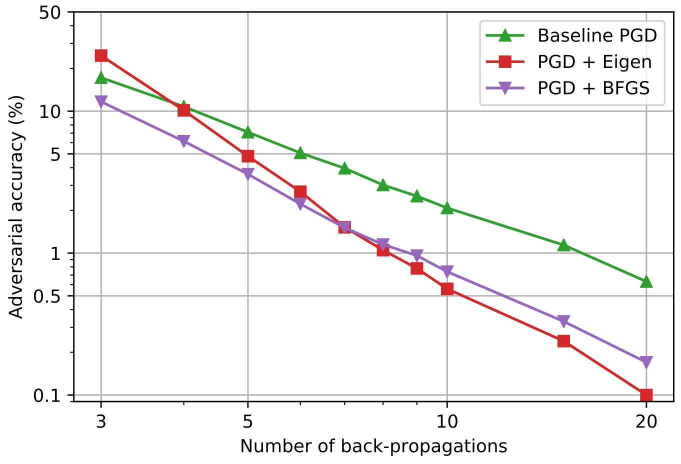
### 2.3.1 Analysis

We train three deep neural networks with WRN architecture under different regularization scheme: 1) no explicit regularization, 2) fixed weight decay regularization with strength $5 \times 10^{-4}$ throughout all 100 epochs, and 3) excessive weight decay regularization with increasing strength, starting with $1 \times 10^{-4}$, and multiply this value with factor of 10 every 40 epochs. In the last scenario, weight decay regularization gets stronger towards the later stage of training, ending up with final strength of $1 \times 10^{-2}$. Other than regularization scheme, all training conditions are the same for these three models. Then, we evaluate these three models for their accuracy against perturbed examples generated by PGD ($\epsilon = 0.5$ in $L_2$ norm) while sweeping the number of iterations of PGD (Figure 2-2 (a)).

Observe that the model with excessive weight decay regularization shows higher



(a)



(b)

Figure 2-2: (a) Adversarial accuracy against perturbed examples generated by a PGD attack ($\epsilon = 0.5$, $L_2$, compensated for zero loss and non-differentiability) of WRN models trained on CIFAR-10 with different regularization conditions, as a function of the number of iterations the attack uses. Shown in log-log scale. (b) Comparison of different compensation methods (Eigen and BFGS) discussed in Section 2.3 on a WRN model trained with excessive weight decay (same as in (a)), under the same number of back-propagations required to find adversarial perturbations.

accuracy when PGD uses few iterations ($< 10$), while the other models show less than 0.1% of accuracy. However, as PGD uses more iterations, accuracy of the model with excessive weight decay regularization degrades and ultimately reaches less than 0.1% of accuracy, similar as the other models, when PGD uses 100 iterations. This observation can be concerning when one uses PGD with small number of iterations, which is common when evaluating complex models where back-propagation is computationally consuming, to compare these regularization techniques for adversarial robustness. In such scenario, one can be misled by higher accuracy of the model with excessive weight decay regularization to conclude that this regularization technique provides robustness.

For the model with excessive weight decay regularization, we characterize cases when PGD with small number of iterations fails. We observe two behaviors: 1) random initialization (Eq (1.4)) affects the success of PGD, such that re-starting PGD from different initialization often succeeds in finding adversarial perturbations despite earlier attempt failed, and 2) successful initialization, from which PGD iterations (Eq (1.5)) succeed to find adversarial perturbations, yields higher increase in loss and *the size of gradients* per iteration. The first observation is a well-known reason for using multiple re-starts when evaluating adversarial robustness using attacks with stochasticity, such as random initialization. While these observations themselves can be trivial, they hint that initialization matters when using PGD, and that initializing to the points where the size of gradients increases rapidly, thus with high curvature, might help subsequent PGD iterations.

### 2.3.2 Compensation methods: approximating second-order information

Although using sufficiently large number of iterations (e.g., 100) for PGD is an uncomplicated method to avoid being misled by this phenomenon, we consider two methods to reduce the number of iterations needed by PGD to find adversarial perturbations to better understand why this phenomenon occurs. We hypothesized above that initializing to the points with high curvature might benefit PGD to easily find adversarial perturbations. To experimentally test this hypothesis, we consider two methods that consider the curvature of the objective function, which is loss on a given input sample, to initialize PGD. Using either of the following methods to initialize, we run PGD iterations as usual to obtain perturbations.

**Eigen: Approximating the Principal Eigenvector**

The principal eigenvector of the Hessian matrix, $H = \nabla\nabla_x l(f(x), t)$, is the eigenvector associated with the maximum eigenvalue of $H$. The principal eigenvector indicates the direction associated with the highest curvature. Initializing along the direction of this principal eigenvector can set subsequent first-order iterations to have rapid increase in the value of loss. However, obtaining $H$ requires second-order derivatives that are numerically challenging to obtain for complex deep neural networks. Therefore, we consider approximating the principal eigenvector of $H$, using the method proposed by Miyato et al. [31]. First, for a matrix $A$, power iteration method can be used to approximate the principal eigenvector $u$:

$$u \leftarrow \frac{A \cdot u}{\|A \cdot u\|_2} \tag{2.11}$$

where the initial $u$ is randomly sampled from some distribution (e.g., $\mathcal{N}(0, I)$) assuming that it has non-zero projection to the actual $u$. For our case, $A = H$, and we have to approximate $H \cdot u$ as well, since the exact $H$ is unattainable. Miyato et al. proposed to use finite difference method to approximate $H \cdot u$:

$$H \cdot u = \frac{\left.\frac{\partial l(f(x'),t)}{\partial x'}\right|_{x'=x+\delta u} - \left.\frac{\partial l(f(x'),t)}{\partial x'}\right|_{x'=x}}{\delta} \tag{2.12}$$

We adopt these two approximation methods to obtain $u$, in particular with a single-step power iteration (Eq (2.11)) to keep the computational overhead of using this method low. Back-propagations are incurred when computing $H \cdot u$, where finite difference method (Eq (2.12)) uses two evaluation of gradients to approximate this value. Therefore, a single-step power iteration incurs two additional back-propagations.

**BFGS: Pseudo-Newton Direction**

Broyden–Fletcher–Goldfarb–Shanno algorithm (BFGS) [13] is a second-order optimization method that approximates the Newton direction $H^{-1} \cdot g$, where $H$ is the Hessian matrix and $g$ is the first-order gradients. BFGS iteratively approximates the inverse of Hessian, $H^{-1}$, and updates the parameters subjected to the optimization using the Quasi-Newton direction $H^{-1} \cdot g$ using the line search to obtain the step size. For our purpose, we are interested in finding a Quasi-Newton direction that incorporates the second-order curvature information only for initialization, instead of full second-order optimization. Thus, we omit the line search, and instead use the Quasi-Newton direction $H^{-1} \cdot g$ for initializing with the fixed vector size $\epsilon$. Furthermore, similar as in the case for Eigen, we only use a single-step approximation for $H^{-1}$ to limit the computational overhead. As such, the resulting computation is:

$$\Delta_g = \frac{\partial l(f(x'), t)}{\partial x'}\Big|_{x'=x+d} - \frac{\partial l(f(x'), t)}{\partial x'}\Big|_{x'=x} \tag{2.13}$$

$$H^{-1} = (I - \frac{d\Delta_g^T}{\Delta_g^T d})(I - \frac{\Delta_g d^T}{\Delta_g^T d}) + \frac{dd^T}{\Delta_g^T d} \tag{2.14}$$

where $d \sim \mathcal{N}(0, I)$ and scaled to have a size $\delta$ (i.e., $\|d\|_2 = \delta$). Note that Eq (2.13) requires two back-propagations, similar to Eq (2.12). However, unlike approximating the principal eigenvector, where intermediate values are vectors ($u$ or $g$) or inner-products of matrices and vectors ($H \cdot u$), this method directly computes the inverse of the Hessian matrix, $H^{-1}$, incurring additional memory consumption. For example, when a flattened (e.g., convert a 3-dimensional input sample to a 1-dimensional vector) input sample has the size of $m$, this method needs additional $m^2$ memory space to store $H^{-1}$. Therefore, using BFGS might limit the size of batch during evaluation.

### 2.3.3 Impact on measurement of robustness

We test the effectiveness of these two initialization methods under the same number of total back-propagations. That is, for $n$ ($n > 2$) total back-propagations, PGD with random initialization uses $n$ first-order iterations (Eq (1.5)), and PGD using either of these initialization methods uses 2 back-propagations for initialization and $n-2$ first-order iterations. Figure 2-2 (b) shows accuracy against perturbed examples generated by PGD with different initialization methods, as a function of the number of total back-propagations. Observe that the proposed initialization methods, Eigen and BFGS, provide stronger attack success rate. For example, when using only 5 back-propagations, PGD with Eigen and PGD with BFGS result in the accuracy of 4.82% and 3.60%, respectively, while the original PGD results in the acuracy of 7.11%. Therefore, exploiting the second-order information for initialization can reduce the number of total back-propagations needed by PGD to find adversarial perturbations when the model trained with specific regularization techniques boosts the apparent accuracy.

## 2.4 Summary

In this chapter, we analyzed three phenomena that inflates adversarial accuracy of deep neural networks. These phenomena affect many conventionally trained deep neural networks, not limited to those using specific defense methods. To summarize, we identify: 1) the **zero loss** phenomenon caused by numerical saturation of loss; 2) innate **non-differentiability** of popularly used ReLU activation and MaxPool operation; 3) certain training conditions **increasing the number of iterations** used by PGD to successfully find adversarial perturbations. For each case, we provide the compensation methods that assist attack methods to compute gradients more accurately and efficiently.

# Chapter 3

# Case Studies

This chapter demonstrates how the three phenomena leading to overestimation of adversarial robustness, discussed in Chapter 2, affect broad range of deep neural networks and obscure analyses on practically important cases, such as comparing adversarial robustness of models with different capacity and regularization techniques. To illustrate the impact of these three phenomena, we compare the accuracy against perturbed examples generated by the conventional attack methods and the compensated attack methods from Chapter 2. For the comparison, both the conventional and the compensated attack methods are set to have the same number of back-propagations and the number of effective re-starts if stochasticity is involved. Appendix B details the settings for how the compensation methods are applied in cascading manner.

## 3.1 Model capacity and adversarial robustness

In this section, we explore the relationship between model capacity and adversarial robustness. As briefly discussed in Section 1.4, recent works suggest that larger model capacity can benefit adversarial robustness. We investigate this relationship for two settings of scaling model capacity: 1) training deep neural networks with the same architecture, but with different number of neurons per layer (in short, we will refer it as 'width') from the scratch, and 2) pruning weights with small magnitude [18] from a pre-trained model with large capacity. We begin this section by summarizing previous works that studied on this relationship. Then, we analyze the two settings and reveal how overestimated adversarial robustness can confuse the understanding of this relationship.

### 3.1.1 Overview of related works

Madry et al. observed that a model with more neurons per layer shows higher accuracy against perturbed examples generated by FGSM and PGD [29]. They found that this observation holds for both the conventionally trained models and the adversarially trained models. For example, they showed that a ResNet model with 10 times larger number of filters for convolutional layers has $3-5\%$ better adversarial accuracy against FGSM and PGD compared to the baseline ResNet model [29]. In such case, the number of total parameters used in convolutional layers will increase by $10^2$, since the number of filters in the layer $i$ becomes the number of input channels in the layer $i+1$. Similar results have been reported by Deng et al., who investigated the trade-off between clean accuracy and adversarial robustness. These works trained models under the same condition except for their widths, and empirically measured adversarial robustness with accuracy against perturbed examples generated by attack methods. We adopt this experimental framework for Section 3.1.2.

The impact of pruning on adversarial robustness has been studied for the purpose of reducing the computational overhead when using adversarial training. That is, as Madry et al. identified, adversarial training often requires larger model capacity to converge and achieve higher accuracy, and recent works focused on reducing the capacity of adversarially trained models via pruning to mitigate the computational burden. Ye et al. and Gui et al. proposed methods to impose pruning constraints (as in the level of sparsity or the structure of sparsity) in adversarial training [52, 16]. On the other hand, Sehwag et al. more recently argued that adversarial training or provably robust training might not be compatible with pruning, and proposed a method to preserve adversarial robustness while reducing the model capacity [42]. In our work, we focus on benchmarking how pruning affects adversarial robustness of undefended, conventionally trained models. This allows us to investigate the relationship between

Table 3.1: The number of parameters and the total number of multiplications required for a single forward pass during inference. We show these metrics for different deep neural network architectures used for each dataset discussed in Section 3.1.2. Also, we compare these metrics for the models with the same architecture, but with different widths (showing the smallest and the largest width considered in Section 3.1.2 for each architecture). Note that parameters and multiplications for batch normalization are not considered, as batch normalization can be folded with preceding layer during the inference when implementing the model.

| | CIFAR10, SVHN | | | | TinyImageNet | | | |
|---|---|---|---|---|---|---|---|---|
| | Simple (BN) | | WRN 28 | | VGG 11 (BN) | | WRN 50 | |
| | $w = 1$ | $w = 16$ | $w = 2$ | $w = 10$ | $w = 1$ | $w = 4$ | $w = 1$ | $w = 5$ |
| Parameters (M) | 0.13688 | 34.6149 | 1.46761 | 36.4792 | 12.5788 | 157.140 | 23.9178 | 333.030 |
| Multiply (G) | 0.00183 | 0.41460 | 0.21435 | 5.24333 | 0.61441 | 9.70163 | 1.30608 | 18.1001 |

the model capacity and inherent adversarial robustness of the model without deploying defense methods. We use iterative weight pruning that removes weights with smallest magnitude [18] in Section 3.1.3.

### 3.1.2 Training models with different capacity from scratch

We benchmark accuracy of models with different widths against perturbed examples generated by attack methods, for diverse architectures and datasets. 'Width' determines the number of output neurons in each layer; for example, the width of convolutional layer is determined by the number of filters. Note that if the width of a model is multiplied by $N$, the number of parameters in that model will be roughly multiplied by $N^2$. This is because the number of output filters in the $i$th layer becomes the number of input channels in the $i + 1$th layer. Exceptions are the first and the last layer, where the input to the first layer is fixed as the input sample's dimension and the output of the last layer is fixed as the total number of classes.

Generally, we observe that the three phenomena can inflate the accuracy of models with larger capacity more frequently, and that using the compensation methods can reduce the gap in the accuracy between small and large models. However, this observation does not homogeneously hold for all datasets and architectures, and we explain the relationship between the model capacity and adversarial robustness for

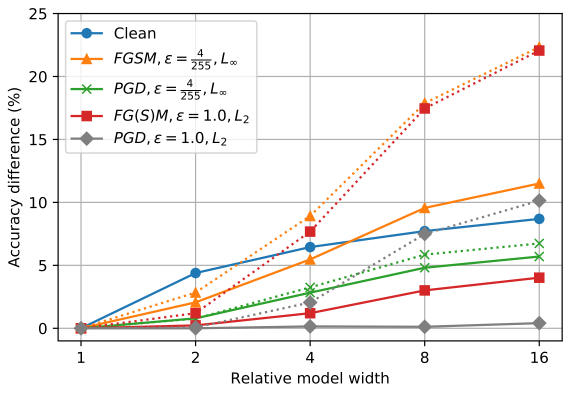
CIFAR-10 [25], SVHN [32], and TinyImageNet [40] datasets. In order to provide a sense for how small and large these models are, we present the number of parameters and the total number of multiplications required for a single forward pass during the inference, for each model architecture we consider in this section (Table 3.1).

### CIFAR-10

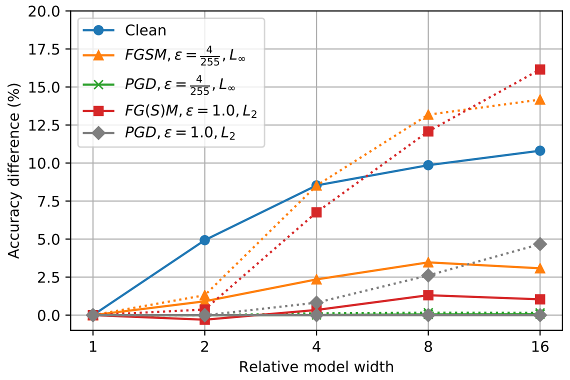
Figure 3-1 shows the relationship between the model capacity and adversarial robustness for Simple (a), Simple-BN (b), and WRN (c) architectures. The figure describes the difference in clean and adversarial accuracy of the models with different relative widths, with respect to the accuracy of the model with the smallest relative width. We find that larger models have better adversarial accuracy, coherent with previous observations. However, the gap in adversarial accuracy between models with different capacity is actually smaller after compensating for the zero-loss and the non-differentiability phenomena. For example, PGD with $\epsilon = 1.0$ in $L_2$ norm gives 10.14% difference in adversarial accuracy between Simple models with width 1 and 16, but compensated PGD gives only 0.41% difference under the same conditions. Similar observations hold for Simple-BN and WRN models. For models trained on CIFAR-10, we observe that the zero-loss phenomenon mostly accounts for overestimated adversarial robustness of models with larger capacities. These observations imply that the benefit of using larger models for higher adversarial accuracy could have been overestimated due to these two phenomena.

### SVHN

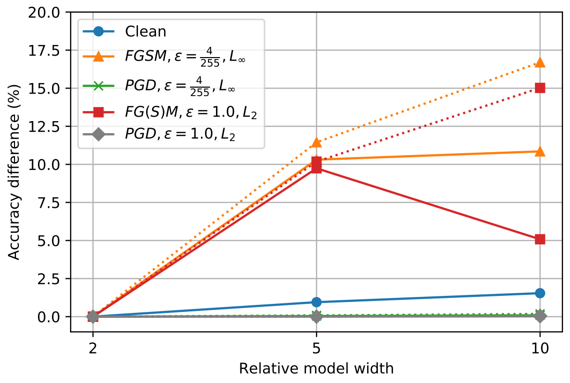
Figure 3-2 shows the same relationship for models with Simple-BN (a) and WRN (b) architectures trained on SVHN dataset. Unlike the models trained on CIFAR-10, we observe that the compensated attack methods do not decrease the gap in the accuracy between large and small models. Rather, we find that the gap between models with different widths is small from the first place, and that compensating often increases the gap. In other words, adversarial accuracy of the smaller models is more overestimated compared to that of the larger ones for SVHN.



(a)

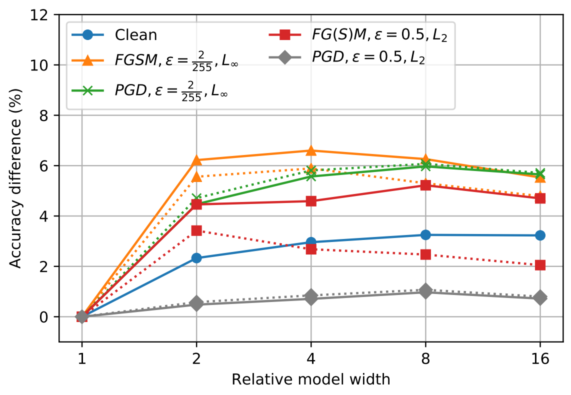


(b)

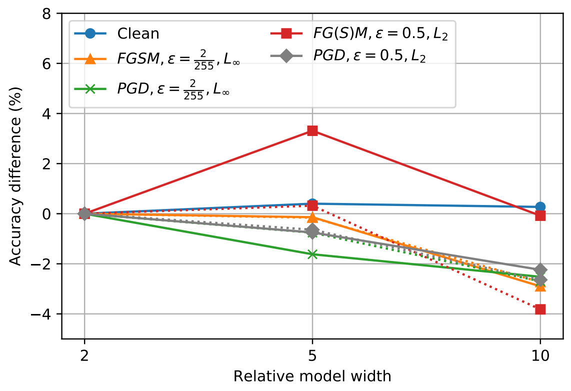


(c)

Figure 3-1: Comparison of clean accuracy and accuracy against perturbed examples generated by attack methods for (a) Simple, (b) Simple-BN, and (c) WRN models with different relative widths , all trained on CIFAR-10. We present difference of accuracy of these models with respect to accuracy of the model with width 1 for Simple and Simple-BN and width 2 for WRN. Dashed and solid line represent accuracy against baseline and compensated attacks, respectively.



(a)

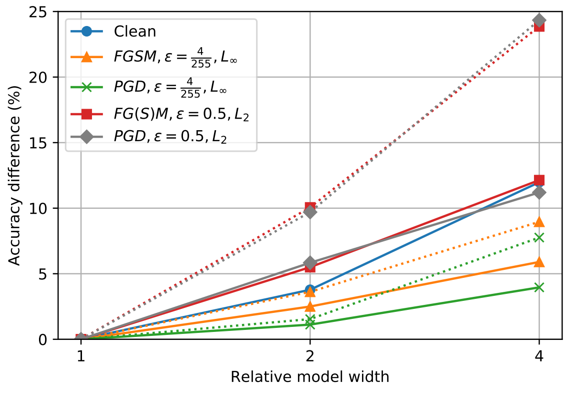


(b)

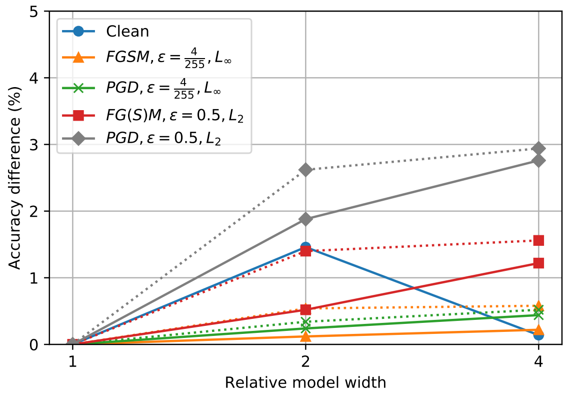
Figure 3-2: Accuracy difference (with respect to the accuracy of the model with width=1 for Simple-BN and width=2 for WRN 28) of (a) Simple-BN models and (b) WRN 28 models trained on SVHN with different relative widths. Dashed and solid lines represent accuracy against baseline and compensated attacks, respectively.

### TinyImageNet

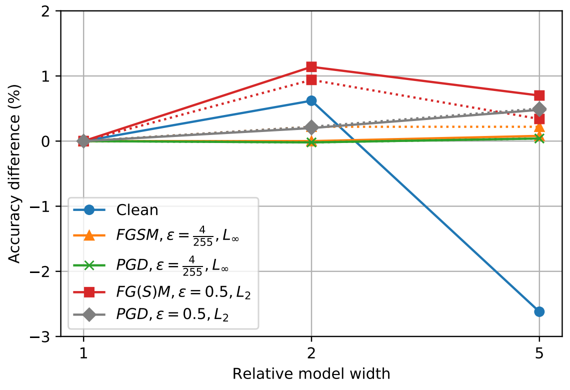
For TinyImageNet dataset, which is a downscaled dataset from ILSVRC classification dataset [40], we experiment on three different architectures, VGG with depth of 11, VGG-BN with depth of 11, and WRN with depth of 50. Figure 3-3 shows the result on these architectures. We find that models with VGG and VGG-BN architectures show similar behavior as models trained on CIFAR-10; that is, robustness of the larger models are overestimated and the compensation methods reduce the gap between small and large models. However, models with WRN architecture behave like models trained on SVHN, without significant difference in robustness between models with different



(a)



(b)



(c)

Figure 3-3: Accuracy difference (with respect to the model with width=1) of (a) VGG 11 models, (b) VGG-BN 11 models, and (c) WRN 50 models trained on TinyImageNet with different relative widths. Dashed and solid lines represent accuracy against baseline and compensated attacks, respectively.

widths. Overall, these experiments show that the relationship between model capacity and adversarial robustness might not be homogeneous across different architectures and datasets, and that the source of overestimation of adversarial robustness might differ as well.

### 3.1.3 Incrementally removing weights via pruning

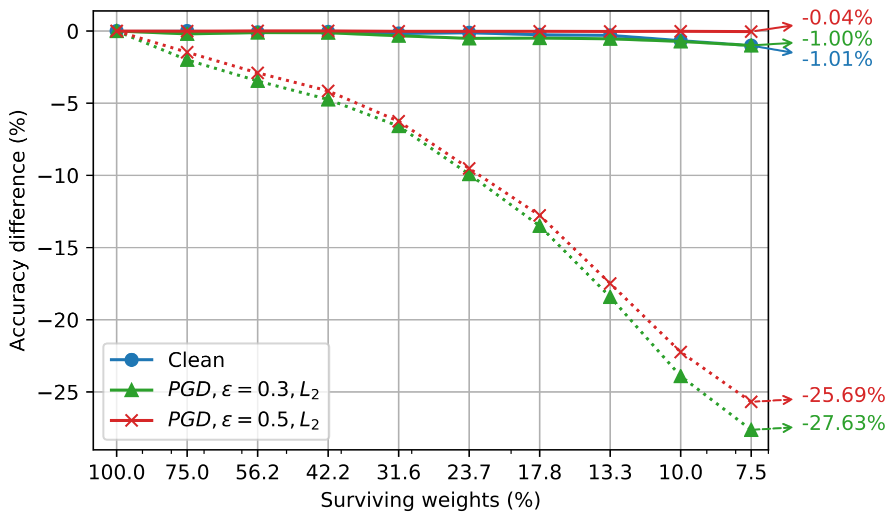
We consider weight pruninig, a popular technique to reduce the number of parameters in a model by removing weights deemed less important, and its relationship with adversarial robustness. For the experiment, we first train a deep neural network with WRN architecture and large width ($w = 10$) on CIFAR-10 dataset. Then, we prune this model by removing weights with small magnitudes and finetune the resulting sparse model for few epochs. We repeat this process until we remove 92.5% of weights from the initial model. We measure clean accuracy and accuracy against perturbed examples generated by PGD for the original dense model and the sparse models (Figure 3-4). We experiment for two different regularization schemes, one without any explicit regularization technique (a), and one with weight decay regularization during training and finetuning (b).

#### Without weight decay

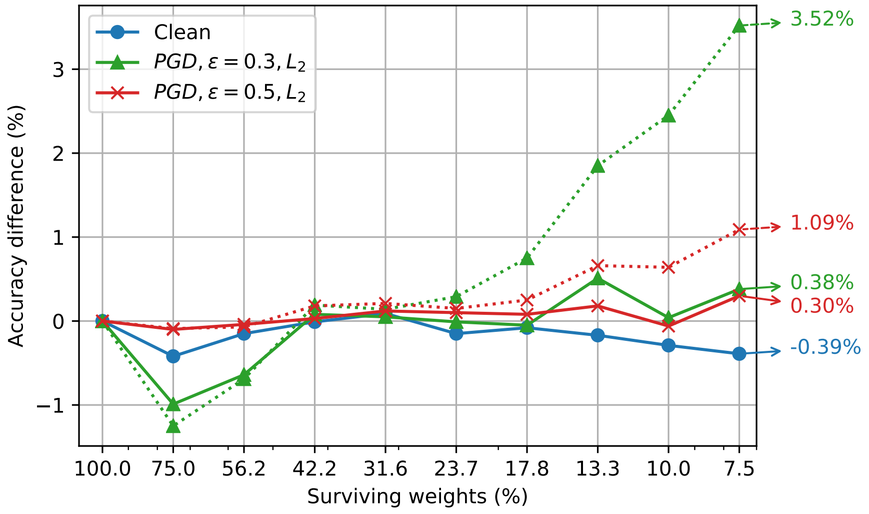
We observe that accuracy against perturbed examples generated by the conventional PGD significantly deteriorates ($> 25\%$) as the level of sparsity increases, although clean accuracy only drops about 1% (Figure 3-4 (a)). However, we find that the adversarial accuracy of the dense model is inflated mostly due to the zero-loss phenomenon, thus using the compensation methods reveals that the accuracy drops only by 1%. In this case, one might wrongly conclude that pruning harms adversarial robustness *without* proper compensation methods.

#### With weight decay

Unlike the case without weight decay, we observe that the accuracy against perturbed examples slightly increases along with the level of sparsity (Figure 3-4 (b)). For



(a)



(b)

Figure 3-4: Change of clean accuracy and accuracy against perturbed examples generated by attack methods through iterative pruning on an over-parameterized WRN 28 (trained on CIFAR-10), trained and finetuned without explicit regularization (a) and with weight decay (b). Dashed and solid lines represent accuracy against baseline attacks and compensated attacks, respectively. The total number of back-propagations is fixed to 9.

example, the accuracy evaluated using PGD with $\epsilon = 0.3$ in $L_2$ norm increases by 3.52% compared to the original dense model at the end of pruning. While this behavior seems contradictory to the case without weight decay discussed above, we find that compensating for the three phenomena, especially using the second-order initialization method for PGD, brings this increase in accuracy down to 0.40%. During iterative pruning and finetuning of the resulting model, finetuning can add up significant number of epochs (e.g., 10 epochs per finetune $\times$ 10 pruning iterations $\rightarrow$ 100 epochs). We think using weight decay for 100 additional epochs can act similar as excessive weight decay regularization discussed in Section 2.3, inflating the apparent accuracy when PGD uses fixed number of iterations as in this case. Therefore, although pruning does not significantly improve or harm adversarial robustness for this WRN model trained on CIFAR-10 dataset, overestimated empirical robustness can confuse the conclusion.

## 3.2 Regularization techniques for robustness

In this section, we investigate regularization techniques that have been proposed for better generalization or robustness, on whether the accuracy on perturbed examples is overestimated due to the three phenomena discussed in Chapter 2. This case study can guide which regularization techniques to use for adversarial robustness, and help understanding characteristics of those techniques. We first introduce regularization techniques we explore in this section (Section 3.2.1) and show experiments on those techniques (Section 3.2.2).

### 3.2.1 Overview of techniques

Regularization can exist in various forms, from explicit penalty term added to loss function to data augmentation and pre-processing. In this section, we focus on few among them that claims to improve generalization or robustness. Some of them explicitly add penalty term to loss function, while some others can be thought as data augmentation.

**Weight Decay**

Weight decay penalizes the Frobenius norm of weight matrices, reducing the magnitude of weights in effect. The resulting loss function $l'(f, x, t)$ during training where $f$ is the model with $L$ layers, $x$ is the input sample, and $t$ is the ground truth label is:

$$l'(f, x, t) = l(f(x), t) + \lambda \sum_{i=1}^{L} \|W_i\|_F^2 \tag{3.1}$$

$W_i$ represents the weight matrix of the $i$th layer of $f$. $\lambda$ controls the strength of the penalty term. Weight decay can be thought as $L_2$ penalty on weights, but note that it cannot be directly associated with margins of classification boundaries as in the case of linear models.

**Spectral Normalization**

Spectral normalization [30] is designed to set the Lipschitz constant of each layer to be 1, to stabilize the training of Generative Adversarial Networks [14] and to improve generalization of models. Miyato et al. proposed to update the approximation of the singular value $\sigma(W_i)$ of each layer $W_i$ $(i = 1, ..., L)$ throughout the training phase, and divide the weight matrix of each layer with the corresponding $\sigma(W_i)$. Also, similar form of regularization that penalizes large $\sigma(W_i)$ was used to improve generalization of deep neural networks and reduce sensitivity to small input perturbations [53]. In our work, we use spectral normalization following each convolutional or linear layer, except for those in residual connections of WRN and the last linear classification layer. Also, spectral normalization is used instead of batch normalization in case batch normalization is originally used for that layer.

**Orthonormal Regularization**

Orthonormal regularization [8] has similar motivation as Spectral normalization, such that it also aims to keep the Lipschitz constant to be 1 to prevent amplification of noise through layers. Orthonormal regularization induces weight matrices to be orthonormal, such that eigenvalues of $W_i^T W_i$ would all become 1. The resulting loss function is:

$$l'(f, x, t) = l(f(x), t) + \lambda \sum_{i=1}^{L} \|W_i^T W_i - I\|_F^2 \tag{3.2}$$

where $I$ is an identity matrix with the same size as $W_i^T W_i$. Note that this orthonormal regularization is part of Parseval training proposed by Cisse et al., but we do not consider other elements of Parseval training [8], such as sampling and other regularization methods for convexity.

**Jacobian & Input Gradient Regularization**

Jacobian regularization [24] or input gradient regularization [39] can be thought as a method to reduce the first-order term in the Taylor's expansion. For small $r$ (e.g., $\|r\| \ll 1$), the Taylor's expansion is

$$l(f(x + r), t) \simeq l(f(x), t) + r^T \cdot \nabla_x l(f(x), t) + O(\|r\|^2) \tag{3.3}$$

For Jacobian regularization, the first-order term is the Jacobian matrix, and we have to compute $\frac{\partial z_i}{\partial x} = [\frac{\partial z_i}{x_1}, ..., \frac{\partial z_i}{x_m}]^T$ for all $i$ where $z$ is the output logits and $m$ is the dimension of the flattened input sample $x$. The resulting loss function is

$$l'(f, x, t) = l(f(x), t) + \lambda \sqrt{\sum_{i=1}^{C} \sum_{j=1}^{m} (\frac{\partial z_i}{\partial x_j})^2} \tag{3.4}$$

where $C$ is the number of class, or the dimension of $z$. Because most libraries for automatic differentiation, such as PyTorch, support differentiation on the scalar value, using Jacobian regularization requires $C$ back-propagations to compute the penalty term. For datasets with large $C$, such as ImageNet with 1000 classes and TinyImageNet with 200 classes, this property can be computationally consuming.

Input gradient regularization similarly penalizes the first-order term, but directly use the gradients of *loss* with respect to input samples. Thus, this method only requires one additional back-propagation to compute the penalty term. The resulting loss function is:

$$l'(f,x,t) = l(f(x),t) + \lambda \sum_{j=1}^{m} (\frac{\partial l(f(x),t)}{\partial x_j})^2 \tag{3.5}$$

We use input gradient regularization for our experiments.

**Adversarial Training**

As briefly introduced in Chapter 1, adversarial training uses perturbed examples as input samples during the training to improve adversarial robustness of the model. Goodfellow et al. proposed to use perturbed examples as a form of data augmentation, and trained models using both clean input samples and perturbed examples [15]. Madry et al. interpreted adversarial training in the perspective of robust optimization [29]. They showed that adversarial training solely using perturbed examples can be thought as solving the following min-max optimization problem where the model $f$ is parameterized with $\theta$:

$$\theta^* = \arg\min_{\theta}\ l(f_\theta(x_{adv}), t) = \arg\min_{\theta}\ \max_{x'}\ l(f_\theta(x'), t) \tag{3.6}$$

where $x'$ is constrained to be within the neighborhood (e.g. $\epsilon$ distance) of the clean sample $x$. In this formulation, attack methods are the approximation of the inner maximization problem.

### 3.2.2 Experiments

First, we train models with WRN architecture on CIFAR-10 dataset using different regularization techniques. Details of the training hyperparameters are presented in Appendix B. Then, we measure each model's accuracy on perturbed examples generated by attack methods, with and without using the compensation methods (Table 3.2). The table displays adversarial accuracy against attacks with different perturbation size $\epsilon$ and the norms ($L_2, L_\infty$). We observe that adversarial accuracy of the model trained with input gradient regularization or adversarial training is *least* affected by the compensation methods. That is, improved adversarial robustness

Table 3.2: Accuracy of WRN 28 models trained on CIFAR-10 using different regularization techniques against perturbed examples generated by first-order attack methods in the order of FGSM/R-FGSM/PGD for stated perturbation sizes $\epsilon$ in $L_\infty$ norm (above) and $L_2$ norm (below). We compare the accuracy against baseline and compensated attacks.

| Regularization | Clean | $\epsilon = \frac{2}{255}$ | | $\epsilon = \frac{4}{255}$ | | $\epsilon = \frac{8}{255}$ | |
|---|---|---|---|---|---|---|---|
| | | Baseline | Compensated | Baseline | Compensated | Baseline | Compensated |
| None | 91.65 | 33.20 / 46.17 / 1.59 | 16.48 / 36.06 / 1.23 | 21.90 / 23.78 / 0.02 | 5.94 / 11.25 / 0 | 17.57 / 7.51 / 0 | 3.36 / 1.30 / 0 |
| Weight decay | 93.64 | 40.27 / 49.01 / 2.24 | 35.50 / 47.26 / 2.00 | 31.47 / 29.69 / 0.02 | 20.31 / 27.32 / 0.02 | 24.78 / 13.62 / 0 | 6.62 / 9.24 / 0 |
| Weight decay excess | 91.52 | 46.45 / 53.60 / 8.68 | 46.45 / 53.60 / 5.12 | 47.83 / 45.48 / 1.85 | 36.46 / 41.58 / 0.63 | 42.91 / 33.05 / 0.32 | 16.90 / 24.24 / 0.01 |
| Spectral norm | 87.31 | 42.64 / 58.19 / 25.62 | 33.52 / 57.66 / 23.50 | 31.50 / 40.79 / 7.50 | 9.86 / 31.67 / 2.25 | 21.10 / 27.45 / 1.77 | 1.46 / 7.84 / 0.01 |
| Orthonormal | 93.47 | 36.54 / 48.34 / 2.48 | 25.91 / 44.66 / 2.40 | 27.19 / 27.29 / 0.03 | 13.25 / 19.76 / 0.01 | 21.78 / 10.97 / 0 | 8.42 / 5.18 / 0 |
| Input Gradient | 89.75 | 55.06 / 74.21 / 49.14 | 53.52 / 73.72 / 48.93 | 24.93 / 53.03 / 12.80 | 23.18 / 52.14 / 12.79 | 7.51 / 20.72 / 0.33 | 3.56 / 19.86 / 0.25 |
| Adv Training | 82.67 | 75.44 / 78.87 / 75.03 | 74.97 / 78.76 / 74.64 | 67.62 / 75.11 / 65.58 | 66.46 / 74.81 / 64.82 | 54.03 / 66.93 / 46.89 | 52.18 / 66.13 / 45.80 |

| Regularization | Clean | $\epsilon = 0.3$ | | $\epsilon = 0.5$ | | $\epsilon = 1.0$ | |
|---|---|---|---|---|---|---|---|
| | | Baseline | Compensated | Baseline | Compensated | Baseline | Compensated |
| None | 91.65 | 49.86 / 53.78 / 26.29 | 19.59 / 38.64 / 0.83 | 47.30 / 47.11 / 21.47 | 11.72 / 21.93 / 0.01 | 45.33 / 39.07 / 15.20 | 7.30 / 7.73 / 0 |
| Weight decay | 93.64 | 48.53 / 55.92 / 1.82 | 41.84 / 54.56 / 1.56 | 41.81 / 45.10 / 0.07 | 28.21 / 42.17 / 0.02 | 35.43 / 28.88 / 0 | 11.26 / 20.55 / 0 |
| Weight decay excess | 91.52 | 56.10 / 57.61 / 9.21 | 51.12 / 56.10 / 4.55 | 53.23 / 53.74 / 2.79 | 44.23 / 50.97 / 0.75 | 50.48 / 44.72 / 0.92 | 26.82 / 37.35 / 0.09 |
| Spectral norm | 87.31 | 43.34 / 58.44 / 28.15 | 33.51 / 57.80 / 22.25 | 38.30 / 45.49 / 21.52 | 16.43 / 39.40 / 4.45 | 34.96 / 36.50 / 18.72 | 4.48 / 15.39 / 0.03 |
| Orthonormal | 93.47 | 43.24 / 53.37 / 2.30 | 29.43 / 47.18 / 1.73 | 36.75 / 41.29 / 0.15 | 19.83 / 31.24 / 0.02 | 30.57 / 25.51 / 0.05 | 13.16 / 14.42 / 0 |
| Input Gradient | 89.75 | 54.07 / 74.04 / 48.41 | 52.46 / 73.48 / 48.10 | 32.67 / 60.09 / 19.55 | 30.69 / 59.20 / 19.51 | 10.32 / 30.43 / 1.90 | 6.38 / 29.31 / 1.16 |
| Adv Training | 83.07 | 73.87 / 78.35 / 73.36 | 73.40 / 78.21 / 73.00 | 67.27 / 75.27 / 65.56 | 66.58 / 75.08 / 64.98 | 52.33 / 67.14 / 47.23 | 50.61 / 66.52 / 46.31 |

when using these regularization techniques is *not* overestimated due to the three phenomena we discussed in Chapter 2. However, adversarial accuracy can be partially overestimated for other regularization techniques, such as weight decay and spectral normalization. For example, for PGD with $\epsilon = \frac{4}{255}$ in $L_\infty$ norm, the model with spectral normalization shows 5.25% drop in the accuracy against perturbed examples generated by this PGD when the compensation methods are applied. Nevertheless, the models with weight decay and spectral normalization show superior robustness compared to the model with no explicit regularization even after compensating for the three phenomena.

Another observation to note is that the drop in adversarial accuracy when using the compensation methods is often smaller for attacks with small $\epsilon$ (e.g., $\epsilon = \frac{2}{255}$ for $L_\infty$ norm or $\epsilon = 0.3$ for $L_2$ norm) compared to that of attacks with larger $\epsilon$. This is evident for the models whose adversarial accuracy against attacks with larger $\epsilon$ is comparable to that with smaller $\epsilon$. For example, the model trained with weight decay regularization shows 4.77% drop in adversarial accuracy against FGSM with $\epsilon = \frac{2}{255}$ in $L_\infty$ norm. However this model shows 11.15% drop when $\epsilon = \frac{4}{255}$, and 21.42% drop when $\epsilon = \frac{8}{255}$ for the same attack and compensation methods. Therefore, one has to be more cautious when evaluating adversarial robustness for larger $\epsilon$.

We repeat this experiment for Simple and Simple-BN architectures (Table 3.3). Observe that the Simple models trained with weight decay and spectral normalization show less drop in adversarial accuracy compared to the WRN models with the same regularization techniques. Furthermore, adversarial accuracy of Simple models is generally less affected by the compensation methods compared to the case of Simple-BN and WRN models. For example, Simple models' adversarial accuracy against PGD changes only about 1% with the compensation methods.

Furthermore, the regularization techniques show different behaviors for different datasets as well (Table 3.4, Table 3.5). For example, adversarial accuracy of the WRN models trained with input gradient regularization drops when the compensation methods are applied in the case of TinyImageNet. On the other hand, the models trained on SVHN do not show large drop in adversarial accuracy when the compensa-

Table 3.3: Accuracy of Simple and Simple-BN models trained on CIFAR-10 using different regularization techniques against perturbed examples generated by baseline and compensated attack methods, in the order of FGSM/R-FGSM/PGD with $\epsilon = \frac{4}{255}$ in $L_\infty$ norm. Note that spectral normalization for Simple-BN is *same* as that for Simple, as spectral normalization layer is used instead of batch normalization layer.

| Regularization | Simple | | |
|---|---|---|---|
| | Clean | Baseline | Compensated |
| None | 84.75 | 19.50 / 30.78 / 2.55 | 8.74 / 28.31 / 1.58 |
| Weight decay | 85.06 | 17.74 / 35.30 / 3.40 | 11.08 / 33.35 / 3.08 |
| Weight decay excess | 84.92 | 15.14 / 36.30 / 3.29 | 11.64 / 34.22 / 3.20 |
| Spectral norm | 81.47 | 22.33 / 47.20 / 13.32 | 20.29 / 45.96 / 13.20 |
| Orthonormal | 84.82 | 16.38 / 39.05 / 5.12 | 12.89 / 37.01 / 5.21 |
| Input gradient | 84.26 | 24.98 / 50.24 / 14.91 | 22.81 / 48.86 / 14.84 |
| Adv training | 67.04 | 54.14 / 60.61 / 53.48 | 52.95 / 60.05 / 52.19 |

| Regularization | Simple-BN | | |
|---|---|---|---|
| | Clean | Baseline | Compensated |
| None | 87.09 | 28.66 / 29.81 / 6.26 | 6.89 / 17.93 / 0.07 |
| Weight decay | 89.84 | 13.62 / 18.32 / 0.13 | 4.69 / 15.15 / 0.01 |
| Weight decay excess | 87.58 | 6.10 / 16.80 / 0.03 | 4.90 / 15.33 / 0.03 |
| Spectral norm | - | - | - |
| Orthonormal | 88.59 | 9.53 / 18.92 / 0.07 | 4.05 / 16.25 / 0.06 |
| Input gradient | 84.67 | 26.74 / 50.98 / 15.02 | 23.71 / 49.68 / 15.00 |
| Adv training | 70.91 | 57.40 / 64.07 / 56.48 | 56.36 / 63.54 / 55.43 |

Table 3.4: Accuracy of Simple-BN and WRN 28 models trained on SVHN using different regularization techniques against perturbed examples generated by baseline and compensated attack methods, in the order of FGSM/R-FGSM/PGD with $\epsilon = \frac{4}{255}$ in $L_\infty$ norm.

| Regularization | Simple-BN | | |
|---|---|---|---|
| | Clean | Baseline | Compensated |
| None | 94.29 | 28.60 / 45.39 / 2.80 | 19.10 / 41.31 / 2.38 |
| Weight decay | 95.37 | 28.06 / 51.61 / 4.57 | 23.90 / 49.64 / 4.64 |
| Weight decay excess | 92.85 | 26.76 / 51.31 / 6.51 | 22.56 / 49.19 / 6.33 |
| Spectral norm | 88.26 | 32.83 / 59.93 / 22.83 | 29.45 / 57.95 / 22.50 |
| Orthonormal | 94.89 | 24.37 / 46.15 / 1.72 | 15.24 / 40.82 / 1.66 |
| Input Gradient | 91.43 | 39.50 / 65.73 / 25.52 | 34.64 / 63.38 / 24.99 |
| Adv training | 83.75 | 66.63 / 74.87 / 63.41 | 64.07 / 74.34 / 61.95 |

| Regularization | WRN 28 | | |
|---|---|---|---|
| | Clean | Baseline | Compensated |
| None | 95.42 | 49.74 / 60.31 / 4.06 | 34.30 / 55.05 / 3.65 |
| Weight decay | 96.38 | 63.99 / 70.25 / 8.71 | 56.25 / 68.12 / 7.31 |
| Weight decay excess | 95.03 | 57.56 / 66.51 / 13.86 | 51.95 / 64.31 / 12.14 |
| Spectral norm | 94.63 | 40.91 / 63.94 / 22.05 | 37.03 / 61.84 / 21.97 |
| Orthonormal | 96.25 | 56.78 / 65.28 / 5.78 | 39.59 / 59.80 / 4.87 |
| Input gradient | 95.45 | 50.22 / 76.16 / 35.49 | 46.49 / 74.80 / 35.13 |
| Adv training | 91.91 | 78.27 / 85.15 / 75.58 | 76.16 / 84.76 / 74.47 |

Table 3.5: Accuracy of WRN 50 models trained on TinyImageNet using different regularization techniques against perturbed examples generated by baseline and compensated attack methods, in the order of FGSM/R-FGSM/PGD with $\epsilon = \frac{2}{255}$ in $L_\infty$ norm.

| Regularization | Clean | Baseline | Compensated |
|---|---|---|---|
| None | 57.24 | 24.38 / 27.90 / 8.50 | 10.40 / 23.76 / 5.30 |
| Weight decay | 55.22 | 5.80 / 14.22 / 0.68 | 4.68 / 13.30 / 0.64 |
| Weight decay excess | 57.54 | 9.56 / 19.88 / 2.06 | 7.04 / 17.64 / 1.84 |
| Input gradient ($\lambda = 0.01$) | 53.70 | 24.66 / 27.96 / 12.94 | 10.76 / 24.88 / 6.68 |
| Input gradient ($\lambda = 0.05$) | 55.58 | 24.56 / 28.62 / 14.52 | 12.50 / 26.94 / 8.66 |

tion methods are applied, even for the model trained with no explicit regularization that typically showed large drop in other datasets. Therefore, whether adversarial robustness of a regularization technique is overestimated depends on the architecture of the model and the dataset on which the model has been trained.

## 3.3 Impact on evaluation against black-box attacks

In this section, we analyze whether the three phenomena that inflates empirical adversarial robustness affect the black-box attack scenarios, where one generates perturbations using attack methods on the substitute model without direct access to the target model. From the following experiments, we show that the transferred examples can be less effective when the substitute model suffers from those three phenomena, and that using the proposed compensation methods can be beneficial for the black-box scenarios as well.

We experiment for the following black-box scenarios: 1) the substitute model has the same architecture as the target model, but has different width and is independently trained (Table 3.6), 2) the substitute model has the same architecture and width, but is trained independently using different regularization technique (Table 3.7), and 3) the substitute model has different architecture from that of the target model (Table 3.8). For all scenarios, we train models on CIFAR-10. We observe that black-box adversarial accuracy is overestimated when the substitute model suffers from the three phenomena, such as the model without explicit regularization in Table 3.7.

Table 3.6: Adversarial robustness of models with Simple architecture and different relative widths (with fixed weight decay of $5 \times 10^{-4}$) under the *black-box* setting, where we craft perturbed examples using the source model. We use PGD with $\epsilon = \frac{4}{255}$ in $L_\infty$ norm, and state accuracy (%) against perturbed examples generated by baseline (before arrow) and compensated (after arrow) attacks. Labels of rows and columns indicate relative width.

| Target Source | 1 | 2 | 4 | 8 | 16 |
|---|---|---|---|---|---|
| 1 | - | $51.80 \rightarrow 46.05$ | $61.55 \rightarrow 56.30$ | $64.33 \rightarrow 59.29$ | $62.82 \rightarrow 57.11$ |
| 2 | $40.12 \rightarrow 34.67$ | - | $48.55 \rightarrow 42.28$ | $50.23 \rightarrow 44.17$ | $50.49 \rightarrow 44.26$ |
| 4 | $39.83 \rightarrow 35.55$ | $37.23 \rightarrow 32.51$ | - | $38.59 \rightarrow 33.48$ | $39.29 \rightarrow 34.42$ |
| 8 | $38.15 \rightarrow 33.40$ | $33.24 \rightarrow 29.04$ | $32.62 \rightarrow 28.33$ | - | $28.70 \rightarrow 25.26$ |
| 16 | $32.33 \rightarrow 27.36$ | $30.89 \rightarrow 26.55$ | $30.08 \rightarrow 26.26$ | $25.60 \rightarrow 22.53$ | - |

Therefore, this experiment emphasizes that overestimated adversarial robustness is also problematic for black-box attack scenarios, although we analyze and develop these phenomena and the compensation methods under the white-box scenario.

Table 3.7: Adversarial robustness of WRN 28 models with different regularization techniques under the *black-box* setting, where adversarial examples are crafted on the source model. These models are independently trained under their own training conditions, but are identically initialized. We use the same attack method as in Table 3.6

| Source \ Target | None | Weight decay | Weight decay excess | Spectral | Orthonormal | Input gradient | Adv train |
|---|---|---|---|---|---|---|---|
| None | ■ | $22.57 \rightarrow 9.54$ | $19.28 \rightarrow 7.92$ | $77.51 \rightarrow 74.78$ | $28.04 \rightarrow 13.99$ | $80.28 \rightarrow 75.61$ | $81.34 \rightarrow 81.12$ |
| Weight decay | $25.37 \rightarrow 19.57$ | ■ | $23.72 \rightarrow 18.69$ | $79.94 \rightarrow 76.40$ | $25.61 \rightarrow 19.58$ | $82.73 \rightarrow 78.36$ | $81.49 \rightarrow 80.87$ |
| Weight decay excess | $36.27 \rightarrow 29.17$ | $39.17 \rightarrow 30.34$ | ■ | $81.56 \rightarrow 79.29$ | $45.89 \rightarrow 35.75$ | $84.66 \rightarrow 81.55$ | $81.68 \rightarrow 81.08$ |
| Spectral | $59.91 \rightarrow 56.33$ | $68.59 \rightarrow 66.03$ | $65.34 \rightarrow 62.75$ | ■ | $70.20 \rightarrow 68.18$ | $72.95 \rightarrow 70.76$ | $80.40 \rightarrow 79.89$ |
| Orthonormal | $26.85 \rightarrow 12.26$ | $21.04 \rightarrow 7.72$ | $26.64 \rightarrow 11.86$ | $79.97 \rightarrow 77.96$ | ■ | $82.43 \rightarrow 79.11$ | $81.53 \rightarrow 81.16$ |
| Input gradient | $26.21 \rightarrow 24.03$ | $33.59 \rightarrow 31.31$ | $32.73 \rightarrow 30.16$ | $52.61 \rightarrow 48.71$ | $36.90 \rightarrow 34.40$ | ■ | $79.41 \rightarrow 78.47$ |
| Adv train | $82.54 \rightarrow 81.34$ | $86.25 \rightarrow 85.55$ | $83.46 \rightarrow 82.40$ | $76.84 \rightarrow 75.04$ | $86.07 \rightarrow 85.24$ | $80.34 \rightarrow 79.53$ | ■ |

Table 3.8: Adversarial robustness of models with different architectures (with fixed weight decay of $5 \times 10^{-4}$) under the *black-box setting*. The Simple and Simple-BN models have width 4, and the WRN 28 model has width 2. Details of evaluation is same as in Table 3.6

| Source \ Target | Simple | Simple-BN | WRN 28 |
|---|---|---|---|
| Simple | ■ | $51.61 \rightarrow 47.81$ | $65.82 \rightarrow 62.48$ |
| Simple-BN | $69.34 \rightarrow 66.19$ | ■ | $37.96 \rightarrow 31.28$ |
| WRN 28 | $78.54 \rightarrow 74.23$ | $62.15 \rightarrow 45.60$ | ■ |

# Chapter 4

# Comparison with verified lower bounds of robustness

This chapter investigates input samples on which verification methods discussed in Section 1.3.2 find adversarial perturbations within the $\epsilon$-ball, while first-order attack methods fail to do so. We find that the three phenomena discussed in Chapter 2 can partially explain such difference, and demonstrate how the compensation methods can reduce the gap between the verified lower bound and the empirically measured upper bound of adversarial robustness. Finally, we show how such gap changes as we sweep for different configurations of PGD, such as PGD's step size $\alpha$ and the number of iterations it uses.

## 4.1 Gap between verified lower bounds and empirically measured upper bounds of robustness

Verification methods provide the provable lower bound of robustness, but usually there exists a gap between the lower bound and the upper bound obtained from the accuracy against perturbed examples generated by attack methods. This gap is generally expected for verification methods that adopt approximations, such as relaxations for non-convex elements in deep neural elements. However, there still

Table 4.1: Comparison of the lower bounds of robustness obtained with MILP [48] and accuracy (%) against perturbed examples generated by the baseline and the compensated PGD attacks (5 random starts for both; the total number of back-propagations is 50 for MNIST and 10 for CIFAR-10). Models are trained to be provably robust [50] in stated $\epsilon$-ball for $L_\infty$ norm. For each model, attacks use the same $\epsilon$ the model has been trained for as the maximum perturbation size.

| Model | Lower bound | Adversarial accuracy | |
|---|---|---|---|
| | | Baseline | Compensated |
| MNIST-A, $\epsilon = 0.4$ | 52.40[1] | 54.96 | 54.09 |
| MNIST-B, $\epsilon = 0.3$ | 75.81[2] | 78.96 | 77.35 |
| CIFAR-A, $\epsilon = \frac{2}{255}$ | 49.80[2] | 50.95 | 50.28 |
| CIFAR-B, $\epsilon = \frac{8}{255}$ | 22.40[2] | 23.13 | 22.46 |

[1] Exact robustness obtained with MILP [48]
[2] Values directly taken from Tjeng et al. (2019)

exists a discrepancy even when the verification method is capable of computing the exact robustness, as in the case of MILP [48] that handle ReLU with Mixed-Integer Programming. Therefore, this gap indicates that attack methods miss some input samples although they are not robust within the $\epsilon$-ball.

We experiment on whether the compensation methods proposed in Chapter 2 can reduce this gap by tightening the upper bound. In doing so, we can reveal whether the gap can be explained by the three phenomena that inflates empirical adversarial robustness. We evaluate the models that are trained to be provably robust using LP with relaxations [50], and obtain their lower bounds with MILP [48]. For the upper bounds, we compare accuracy against perturbed examples generated by the baseline PGD and the compensated PGD. Details of these models are explained in Appendix C.

Table 4.1 shows the result of comparing the lower bounds and the upper bounds. First, observe that there is 0.73%-3.15% gap between the lower bounds (an exact bound for MNIST-A) and the accuracy against perturbed examples generated by the baseline PGD. Second, using the compensation methods can reduce this gap to 0.06%-1.69%. The major source of difference is the non-differentiability phenomenon for the models MNIST-A and MNIST-B, and the zero-loss phenomeon for the models

Table 4.2: Accuracy (%) against perturbed examples generated by the baseline PGD and the compensated PGD for MNIST-B for different pairs of PGD configurations (step size $\alpha$ and the number of iterations used by PGD).

| PGD Configurations (Step Size, Number of Iter) | Baseline | Compensated |
|---|---|---|
| (0.01, 30) | 80.29 | 79.82 |
| (0.05, 30) | 78.78 | 76.64 |
| (0.10, 30) | 79.42 | 76.67 |
| (0.01, 50) | 78.96 | 77.35 |
| (0.05, 50) | 78.76 | 76.67 |
| (0.10, 50) | 79.16 | 76.64 |
| (0.01, 100) | 78.35 | 76.66 |
| (0.05, 100) | 78.77 | 76.64 |
| (0.10, 100) | 79.35 | 76.67 |

CIFAR-A and CIFAR-B. Therefore, identifying the sources of overestimation when using attack methods and compensating for the causes can tighten the approximation for the upper bounds of adversarial robustness.

## 4.2 Sensitivity to attack configurations

In this section, we analyze sensitivity of the accuracy against perturbed examples for different configurations of PGD. Since PGD's step size $\alpha$ that controls the per-iteration update in Eq (1.5) and the number of iterations are pre-defined, these configurations can affect the performance of PGD. In Table 4.2, we report the accuracy for 9 different configurations ($\alpha = \{0.01, 0.05, 0.10\}, \text{iter} = \{30, 50, 100\}$). We can see that the compensated PGD consistently provides tighter upper bounds compared to the baseline PGD. The baseline PGD achieves the best performance of 78.35% when it uses $\alpha = 0.01$ and 100 iterations, and the compensated PGD tightens the upper bound by 1.69% for this configuration.

# Chapter 5

# Conclusion

## 5.1 Contributions

Overestimated adversarial robustness has been mainly investigated in the context of defense methods [36, 2, 49]. However, we demonstrate that the sources of overestimated adversarial robustness exist in more broader range of deep neural networks, across different architectures and datasets. Also, those sources of overestimation are not results of intentional design to obfuscate gradients or induce attacks to be less efficient, unlike defense methods that relied on similar mechanisms to increase their apparent adversarial robustness. Thus, our work highlights cautions of using the empirical approach to evaluate adversarial robustness for more general applications.

To summarize, we identify the three phenomena that inflate the model's accuracy against perturbed examples generated by bounded first-order attack methods:

1. **Zero loss** when the value of cross-entropy loss numerically saturates to zero (Section 2.1)

2. **Innate non-differentiability** of ReLU and MaxPool causing gradient shattering (Section 2.2)

3. **Requiring more iterations** to find adversarial perturbations when using PGD on the model trained with certain regularization techniques, such as weight decay regularization (Section 2.3)

For each phenomenon, we provide easy-to-use compensation methods that can be combined with existing first-order attack methods, such as FGSM, R-FGSM, and PGD, to sharpen the empirical evaluation metric.

We illustrate consequences of these three phenomena using the case studies with practical interest (Chapter 3). In particular, we show how the relationship between the model capacity and adversarial robustness can be misled by these three phenomena (Section 3.1), and how evaluation of regularization techniques for their benefit on adversarial robustness can be affected (Section 3.2) as well. Furthermore, we extend our analyses to the black-box attack scenarios (Section 3.3). Among the observations we made in Chapter 3, we emphasize that using the compensation methods can change the analyses on the relationship between the model capacity and adversarial robustness. Therefore, our proposed compensation methods can be beneficial when benchmarking energy-efficiency of deep neural networks in relation to their adversarial robustness.

Finally, we explain the gap between the verified lower bounds and the upper bounds that are empirically measured using the accuracy against perturbed examples with these three phenomena. We demonstrate using the compensation methods for these three phenomena can tighten the upper bounds for the models that are trained to be provably robust (Chapter 4).

## 5.2 Future work

First, the three phenomena we describe in our work might not be an exhaustive list responsible for overestimated adversarial robustness. For example, there is still a gap between the *exact* value of robustness obtained by MILP and the accuracy against perturbed examples, even after the compensation in the case of MNIST-A in Table 4.1. There might exist another phenomenon that can explain this gap although we have not captured them in this work.

Second, there can be better compensation methods for the phenomena we describe here. Note that our compensation methods involve many heuristics, such as how to change the target label when compensating for the zero loss phenomenon and choosing the approximation function to be used for BPDA. Therefore, it is possible that our compensation methods are not optimal.

Third, our analyses are based on empirical observations and experiments, but lack theoretical groundings. Although the zero loss phenomenon is not a theoretically expected behavior since it is caused by numerical precision, other two phenomena might have theoretically grounded explanations.

Lastly, applying our proposed empirical metric using the compensation methods to practically important domains, such as understanding the impact of other model compression techniques on adversarial robustness and exploring the optimal architecture considering adversarial robustness, can be valuable. Since these practical studies often use complex models and datasets, verification methods might be hard to scale and using precise empirical metric can be important.

# Appendix A

# Deep neural network architectures

Simple and Simple-BN architectures are explained in detail in Table A.1. For WRN 28, we modify the number of output channels and pooling window size to fit with smaller input dimension of CIFAR-10 and SVHN compared to ImageNet. For VGG and WRN used for TinyImageNet, we use the architectures defined in TorchVision, and only modify final pooling and fully connected layer's dimension to fit downscaled TinyImageNet ($3 \times 64 \times 64$ with 200 classes).

Table A.1: Description of architectures used in this paper. Convolution layers are specified as (output channel, input channel, kernel height, kernel width, stride, padding). Maxpool layers are in (kernel height, kernel width, stride, padding), and fully connected (FC) layers are in (output channel, input channel).

| Model Type | Description ($w$: width scaling factor) |
|---|---|
| Simple | Conv1 : ($w \times 8$, 3, 3, 3, 1, 1)<br>Conv2 : ($w \times 8$, $w \times 8$, 3, 3, 1, 1)<br>MaxPool: (2, 2, 2, 0)<br>Conv3 : ($w \times 16$, $w \times 8$, 3, 3, 1, 1)<br>Conv4 : ($w \times 16$, $w \times 16$, 3, 3, 1, 1)<br>MaxPool: (2, 2, 2, 0)<br>FC1 : ($w \times 128$, $w \times 16 \times 8 \times 8$)<br>FC2 : (10, $w \times 128$) |
| Simple-BN | Convolution and FC layers are same as in Simple, but Batch Normalization layer follows each Convolution layer. |

As a default setting, a model is train for 100 epochs using Stochastic Gradient

Descent (SGD) with momentum of 0.9 for CIFAR-10 and SVHN. For Simple-BN and WRN 28, we use starting learning rate of 0.1 and decay it by factor of 10 for every 40 epochs. For Simple, we start with learning rate of 0.01. Models for TinyImageNet are trained with Adam ($\beta_1 = 0.9, \beta_2 = 0.99$), with starting learning rate of 0.001. Learning rate decay is applied in the same manner. Default batch size is 128, unless GPU memory is insufficient. Different training conditions deviating from the default setting, including specific regularizations, are described in Appendix B.

For the datasets, we use CIFAR-10 [25], SVHN [32], and TinyImageNet (a down-sampled dataset from Russakovsky et al. (2015) [40]). The images are normalized to the range $[0, 1]$ for both training and testing, and further pre-processing includes random crop and flips during training. We randomly split 10% of training samples for validation purpose. For Chapter 4, we additionally use MNIST [27] and follow pre-processing of Wong and Kolter (2017) for MNIST and CIFAR-10.

# Appendix B

# Experiment settings for Chapter 3

## B.1 Compensation methods for attacks

Accuracy against perturbed examples generated by the compensated attacks is measured in a cascading manner, in which samples that survived the previous stage (i.e., an attack fails on that sample to find an adversarial perturbation) are subjected to the next compensation method. For a single-step attack, such as FGSM or R-FGSM, we cascade compensation methods in following steps:

1. Apply an attack without any compensation methods.
2. Apply a compensation method for the zero loss phenomenon (default: change target labels to the second most likely classes)
3. Apply a compensation method for the non-differentiability phenomenon (default: BPDA with softplus as a substitute for ReLU and $L_p$ norm pooling with $p = 5$ for max pooling)
4. Apply both compensation methods in 2 and 3 together

This scheme results in 4 effective evaluations, and baseline attacks with stochasticity (e.g. R-FGSM) are set to have 4 random starts (i.e., a sample has to survive all four attempts to be considered accurate; in other words, this is equivalent to 4 cascading attacks but without compensation methods) for fair comparison. For iterative attacks, such as PGD, we cascade as:

1. Apply an attack without any compensation methods (e.g., plain PGD).
2. Apply PGD with an initialization method proposed in Sec 2.3 (default: PGD + Eigen)
3. Apply a compensation method for the zero loss phenomenon with plain PGD (default: change target labels to the second most likely classes)
4. Apply both compensation methods in 2 and 3 together
5. Apply a compensation method for the non-differentiability phenomenon along with compensation methods in 2 and 3 (default: BPDA with softplus as a substitute for ReLU and $L_p$ norm pooling with $p = 5$ for max pooling)

Note that for iterative attacks, we do not test for every possible combination of compensation methods for the three phenomena. Resulting scheme has 5 effective evaluations, and baseline attacks are set to have 5 random starts. For PGD attacks in subsequent experiments, we use total 9 back-propagations (9 iterations without initialization methods of Sec 2.3 or 7 iterations with those initialization methods) as a default value. When a compensation method is not effective for a given model, just using another random start can be more effective than using that compensation method. In such case, accuracy against the baseline attacks can be lower than accuracy against the compensated attacks.

## B.2 Experiment settings for Section 3.1.1

Table B.1: Training hyperparamters of models used in Section 3.1.1

| Dataset | Architecture | Training condition |
|---|---|---|
| CIFAR-10, SVHN | Simple | Epochs: 100, Batch size: 128<br>Optimizer: SGD with momentum of 0.9<br>Learning rate: start with 0.01, decay by factor of 10 every 40 epochs<br>Regularization: fixed weight decay of $5 \times 10^{-4}$ |
| | Simple-BN, WRN 28 | Epochs: 100, Batch size: 128<br>Optimizer: SGD with momentum of 0.9<br>Learning rate: start with 0.1, decay by factor of 10 every 40 epochs<br>Regularization: fixed weight decay of $5 \times 10^{-4}$ |
| TinyImageNet | VGG 11 | Epochs: 100, Batch size: 128 (96 for scale factor 4)<br>Optimizer: Adam ($\beta_1 = 0.9, \beta_2 = 0.99$)<br>Learning rate: start with $10^{-4}$. decay by factor of 10 every 40 epochs<br>Regularization: none |
| | VGG-BN 11 | Epochs: 100, Batch size: 128 (96 for scale factor 4)<br>Optimizer: Adam ($\beta_1 = 0.9, \beta_2 = 0.99$)<br>Learning rate: start with $10^{-3}$. decay by factor of 10 every 40 epochs<br>Regularization: none |
| | WRN 50 | Epochs: 100, Batch size: 128<br>Optimizer: Adam ($\beta_1 = 0.9, \beta_2 = 0.99$)<br>Learning rate: start with $10^{-3}$. decay by factor of 10 every 40 epochs<br>Regularization: fixed weight decay of $5 \times 10^{-4}$ |

## B.3 Experiment settings for Section 3.1.2

For weight pruning, we initially train a WRN 28 model with width scale factor of 10. The model is trained for 100 epochs using SGD with momentum of 0.9 as an optimizer, with starting learning rate of 0.1, which is decayed by factor of 10 every 40 epochs. We use early stopping based on the validation accuracy. Batch size is fixed to 128. To compare the impact of using weight decay, we train two models with and without weight decay of $5 \times 10^{-4}$ with otherwise same training conditions as stated.

We iteratively remove weights with small magnitude as in typical weight pruning. To be specific, in each pruning iteration, we remove 25% of total weights from convolution layers based on their magnitude, and finetune for 10 epochs with learning rate of 0.001. Otherwise, finetuning conditions are same as training conditions.

However, note that the optimizer is initialized at each pruning iteration so that the momentum term from previous iteration (which contains information on removed weights) does not affect current finetuning. We iterate this process for 9 cycles, and the proportion of surviving weights at the final stage is 7.5%.

## B.4 Experiment settings for Section 3.2

Details of training conditions for the models used in Section 3.2 are presented in Table B.2.

Table B.2: Training hyperparameters of models used in Section 3.2

| Dataset | Architecture | Regularization ($\lambda$ if applicable) | Other training conditions |
|---|---|---|---|
| CIFAR-10 | Simple (width=4) | None | Epochs: 100, Batch size: 128, Optimizer: SGD (momentum=0.9), Learning rate: start with 0.01, decay by 10 / 40 epochs |
| | | Weight decay, $\lambda = 5 \times 10^{-4}$ | |
| | | Weight decay excess | |
| | | Spectral normalization | |
| | | Orthonormal, $\lambda = 1 \times 10^{-4}$ | |
| | | Input gradiet, $\lambda = 1.0$ | |
| | | Adversarial training, for $L_\infty$ norm: $\epsilon = \frac{8}{255}$, 7 iterations for $L_2$ norm: $\epsilon = 1.0$, 7 iterations Apply fixed weight decay of $\lambda = 5 \times 10^{-4}$ | |
| | Simple-BN (width=4) | None | Learning rate: start with 0.1 Otherwise same as above |
| | | Weight decay, $\lambda = 5 \times 10^{-4}$ | |
| | | Weight decay excess | |
| | | Orthonormal, $\lambda = 1 \times 10^{-3}$ | |
| | | Input gradiet, $\lambda = 1.0$ | |
| | | Adversarial training, for $L_\infty$ norm: $\epsilon = \frac{8}{255}$, 7 iterations for $L_2$ norm: $\epsilon = 1.0$, 7 iterations Apply fixed weight decay of $\lambda = 5 \times 10^{-4}$ | |
| | WRN 28 (width=2) | None | |
| | | Weight decay, $\lambda = 5 \times 10^{-4}$ | |
| | | Weight decay excess | |
| | | Spectral normalization | |
| | | Orthonormal, $\lambda = 1 \times 10^{-3}$ | |
| | | Input gradiet, $\lambda = 1.0$ | |
| | | Adversarial training, for $L_\infty$ norm: $\epsilon = \frac{8}{255}$, 7 iterations for $L_2$ norm: $\epsilon = 1.0$, 7 iterations Apply fixed weight decay of $\lambda = 5 \times 10^{-4}$ | |
| SVHN | Simple-BN (width=4) | None | Epochs: 100, Batch size: 128, Optimizer: SGD (momentum=0.9), Learning rate: start with 0.1, decay by 10 / 40 epochs |
| | | Weight decay, $\lambda = 5 \times 10^{-4}$ | |
| | | Weight decay excess | |
| | | Spectral normalization | Optimizer: Adam ($\beta_1 = 0.9, \beta_2 = 0.99$) Learning rate: start with $10^{-4}$ Otherwise same as above. |
| | | Orthonormal, $\lambda = 1 \times 10^{-3}$ | Same as those for 'None' (see above) |
| | | Input gradiet, $\lambda = 1.0$ | |
| | | Adversarial training, for $L_\infty$ norm: $\epsilon = \frac{8}{255}$, 7 iterations for $L_2$ norm: $\epsilon = 1.0$, 7 iterations Apply fixed weight decay of $\lambda = 5 \times 10^{-4}$ | Optimizer: Adam ($\beta_1 = 0.5, \beta_2 = 0.5$) Learning rate: start with $10^{-3}$ Otherwise same as above. |
| | WRN 28 (width=2) | None | Epochs: 100, Batch size: 128, Optimizer: SGD (momentum=0.9), Learning rate: start with 0.1, decay by 10 / 40 epochs |
| | | Weight decay, $\lambda = 5 \times 10^{-4}$ | |
| | | Weight decay excess | |
| | | Spectral normalization | |
| | | Orthonormal, $\lambda = 1 \times 10^{-3}$ | |
| | | Input gradiet, $\lambda = 1.0$ | |
| | | Adversarial training, for $L_\infty$ norm: $\epsilon = \frac{8}{255}$, 7 iterations for $L_2$ norm: $\epsilon = 1.0$, 7 iterations Apply fixed weight decay of $\lambda = 5 \times 10^{-4}$ | Optimizer: Adam ($\beta_1 = 0.5, \beta_2 = 0.9$) Learning rate: start with $10^{-3}$ Otherwise same as above. |
| TinyImageNet | WRN 50 (width=1) | None | Epochs: 100, Batch size: 128, Optimizer: Adam ($\beta_1 = 0.9, \beta_2 = 0.99$), Learning rate: start with 0.1, decay by 10 / 40 epochs |
| | | Weight decay, $\lambda = 5 \times 10^{-4}$ | |
| | | Weight decay excess | |
| | | Input gradiet, $\lambda = 0.01$ | Batch size: 64 Otherwise same as above. |
| | | Input gradiet, $\lambda = 0.05$ | |

# Appendix C

# Experiment settings for Chapter 4

Here we elaborate on the experimental setup for Chapter 4. We introduce each model considered, and methods to obtain the lower bound of each model. We follow data preprocessing of Wong and Kolter (2017) for MNIST and CIFAR-10 dataset, which additionally includes normalization in case of CIFAR-10; the size of perturbation $\epsilon$ is scaled according to the normalization so that the pixel level perturbation size (which assumes 0-255 RGB image encoded with 8-bit) can be preserved.

- MNIST-A, $\epsilon = 0.4$ : this model uses `'small'` model of Wong and Kolter (LP$_d$-CNN$_A$ of Tjeng et al.), and is trained with the code publicly available made by Wong and Kolter (`https://github.com/locuslab/convex_adversarial`). Training hyperparameters are: `cascade=1, epochs=200, schedule_length=20, norm_type=l1_median, norm_eval=l1, starting_epsilon=0.01, verbose=200, batch_size=20, test_batch_size=10, eps=0.4`. To obtain the lower bound of this model, we use verification method with MILP, with publicly available code provided by Tjeng et al. (`https://github.com/vtjeng/MIPVerify.jl`). We measure for untargeted adversarial robustness with $\epsilon = 0.4$ in $L_\infty$ norm, and find that MILP can provide *exact* robustness (that is, there is no gap between upper and lower bounds obtained by MILP) for this model.

- MNIST-B, $\epsilon = 0.3$ : this model is `'large'` model of Wong and Kolter (LP$_d$-CNN$_B$ of Tjeng et al.), and is directly obtained from repository of Wong and Kolter (`'mnist_large_0_3.pth'`). The lower bound of *robustness* is directly taken from Tjeng et al.; because they measure lower and upper bounds of *error*, we take (100% − `upper bound of error`) from Tjeng et al. as the lower bound of robustness.

- CIFAR-A, $\epsilon = \frac{2}{255}$ : this model is `'small'` model for CIFAR-10 of Wong and Kolter (LP$_d$-CNN$_A$ of Tjeng et al.), and is directly obtained from repository of Wong and Kolter (`'cifar_small_2px.pth'`). The lower bound of robustness is directly taken from Tjeng et al., as MNIST-B.

- CIFAR-B, $\epsilon = \frac{8}{255}$ : this model is `'ResNet'` model for CIFAR-10 of Wong and Kolter (LP$_d$-RES of Tjeng et al.), and is directly obtained from repository of Wong and Kolter (`'cifar_resnet_8px.pth'`). The lower bound of robustness is directly taken from Tjeng et al., as MNIST-B.

Adversarial accuracy is measured using a PGD attack with stated $\epsilon$ for each model in $L_\infty$ norm. Baseline attacks use 5 random starts to have the same number of evaluations as compensated attacks.

# Bibliography

[1] Anish Athalye, Logan Engstrom, Andrew Ilyas, and Kevin Kwok. Synthesizing robust adversarial examples, 2017.

[2] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples, 2018.

[3] Jonathan T. Barron. Continuously differentiable exponential linear units, 2017.

[4] Battista Biggio and Fabio Roli. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition*, 84:317–331, Dec 2018. ISSN 0031-3203. doi: 10.1016/j.patcog.2018.07.023. URL `http://dx.doi.org/10.1016/j.patcog.2018.07.023`.

[5] Battista Biggio, Giorgio Fumera, Ignazio Pillai, and Fabio Roli. A survey and experimental evaluation of image spam filtering techniques. *Pattern Recognition Letters*, 32(10):1436–1446, 2011.

[6] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. *2017 IEEE Symposium on Security and Privacy (SP)*, May 2017. doi: 10.1109/sp.2017.49. URL `http://dx.doi.org/10.1109/sp.2017.49`.

[7] Nicholas Carlini and David Wagner. Audio adversarial examples: Targeted attacks on speech-to-text. *2018 IEEE Security and Privacy Workshops (SPW)*, May 2018. doi: 10.1109/spw.2018.00009. URL `http://dx.doi.org/10.1109/SPW.2018.00009`.

[8] Moustapha Cisse, Piotr Bojanowski, Edouard Grave, Yann Dauphin, and Nicolas Usunier. Parseval networks: Improving robustness to adversarial examples, 2017.

[9] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus), 2015.

[10] Zhun Deng, Cynthia Dwork, Jialiang Wang, and Yao Zhao. Architecture selection via the trade-off between accuracy and robustness, 2019.

[11] Guneet S. Dhillon, Kamyar Azizzadenesheli, Jeremy D. Bernstein, Jean Kossaifi, Aran Khanna, Zachary C. Lipton, and Animashree Anandkumar. Stochastic activation pruning for robust adversarial defense. In *International Conference*

*on Learning Representations*, 2018. URL `https://openreview.net/forum?id=H1uR4GZRZ`.

[12] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust physical-world attacks on deep learning models, 2017.

[13] Roger Fletcher. *Practical methods of optimization*. John Wiley & Sons, 2013.

[14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

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

[16] Shupeng Gui, Haotao Wang, Chen Yu, Haichuan Yang, Zhangyang Wang, and Ji Liu. Model compression with adversarial robustness: A unified optimization framework, 2019.

[17] Chuan Guo, Mayank Rana, Moustapha Cisse, and Laurens van der Maaten. Countering adversarial images using input transformations. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=SyJ7ClWCb`.

[18] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, 2015.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2016. doi: 10.1109/cvpr.2016.90. URL `http://dx.doi.org/10.1109/CVPR.2016.90`.

[20] Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Girshick. Mask r-cnn. *2017 IEEE International Conference on Computer Vision (ICCV)*, Oct 2017. doi: 10.1109/iccv.2017.322. URL `http://dx.doi.org/10.1109/ICCV.2017.322`.

[21] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015.

[22] Sandy Huang, Nicolas Papernot, Ian Goodfellow, Yan Duan, and Pieter Abbeel. Adversarial attacks on neural network policies, 2017.

[23] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[24] Daniel Jakubovitz and Raja Giryes. Improving dnn robustness to adversarial attacks using jacobian regularization, 2018.

[25] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[26] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world, 2016.

[27] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11): 2278–2324, 1998.

[28] Ji Lin, Chuang Gan, and Song Han. Defensive quantization: When efficiency meets robustness. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=ryetZ20ctX`.

[29] 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.

[30] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=B1QRgziT-`.

[31] Takeru Miyato, Shin-Ichi Maeda, Masanori Koyama, and Shin Ishii. Virtual adversarial training: A regularization method for supervised and semi-supervised learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41 (8):1979–1993, Aug 2019. ISSN 1939-3539. doi: 10.1109/tpami.2018.2858821. URL `http://dx.doi.org/10.1109/TPAMI.2018.2858821`.

[32] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

[33] Travis E Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

[34] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples, 2016.

[35] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. *2016 IEEE Symposium on Security and Privacy (SP)*, May 2016. doi: 10.1109/sp.2016.41. URL `http://dx.doi.org/10.1109/sp.2016.41`.

[36] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '17, page 506–519, New York, NY,

USA, 2017. Association for Computing Machinery. ISBN 9781450349444. doi: 10.1145/3052973.3053009. URL `https://doi.org/10.1145/3052973.3053009`.

[37] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017.

[38] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Certified defenses against adversarial examples, 2018.

[39] Andrew Slavin Ross and Finale Doshi-Velez. Improving the adversarial robustness and interpretability of deep neural networks by regularizing their input gradients, 2017.

[40] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

[41] Pouya Samangouei, Maya Kabkab, and Rama Chellappa. Defense-GAN: Protecting classifiers against adversarial attacks using generative models. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=BkJ3ibb0-`.

[42] Vikash Sehwag, Shiqi Wang, Prateek Mittal, and Suman Jana. On pruning adversarially robust neural networks. *ArXiv*, abs/2002.10509, 2020.

[43] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K. Reiter. Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 1528–1540, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4139-4. doi: 10.1145/2976749.2978392. URL `http://doi.acm.org/10.1145/2976749.2978392`.

[44] Aman Sinha, Hongseok Namkoong, and John Duchi. Certifying some distributional robustness with principled adversarial training, 2017.

[45] Dong Su, Huan Zhang, Hongge Chen, Jinfeng Yi, Pin-Yu Chen, and Yupeng Gao. Is robustness the cost of accuracy? - a comprehensive study on the robustness of 18 deep image classification models. *ArXiv*, abs/1808.01688, 2018.

[46] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S. Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12): 2295–2329, Dec 2017. ISSN 1558-2256. doi: 10.1109/jproc.2017.2761740. URL `http://dx.doi.org/10.1109/JPROC.2017.2761740`.

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

[48] Vincent Tjeng, Kai Y. Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=HyGIdiRqtm`.

[49] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses, 2017.

[50] Eric Wong and J. Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope, 2017.

[51] Cihang Xie, Jianyu Wang, Zhishuai Zhang, Yuyin Zhou, Lingxi Xie, and Alan Yuille. Adversarial examples for semantic segmentation and object detection. *2017 IEEE International Conference on Computer Vision (ICCV)*, Oct 2017. doi: 10.1109/iccv.2017.153. URL `http://dx.doi.org/10.1109/ICCV.2017.153`.

[52] Shaokai Ye, Xue Lin, Kaidi Xu, Sijia Liu, Hao Cheng, Jan-Henrik Lambrechts, Huan Zhang, Aojun Zhou, Kaisheng Ma, and Yanzhi Wang. Adversarial robustness vs. model compression, or both? *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, Oct 2019. doi: 10.1109/iccv.2019.00020. URL `http://dx.doi.org/10.1109/ICCV.2019.00020`.

[53] Yuichi Yoshida and Takeru Miyato. Spectral norm regularization for improving the generalizability of deep learning, 2017.

[54] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks, 2016.

[55] Bolei Zhou, Agata Lapedriza, Aditya Khosla, Aude Oliva, and Antonio Torralba. Places: A 10 million image database for scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.