UCLA
UCLA Electronic Theses and Dissertations

Title
On the Robustness of Neural Network: Attacks and Defenses

Permalink
https://escholarship.org/uc/item/3k2780bg

Author
Cheng, Minhao

Publication Date
2021

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

UNIVERSITY OF CALIFORNIA
Los Angeles

On the Robustness of Neural Network: Attacks and Defenses

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Computer Science

by

Minhao Cheng

2021

© Copyright by
Minhao Cheng
2021

ABSTRACT OF THE DISSERTATION

On the Robustness of Neural Network:
Attacks and Defenses

by

Minhao Cheng
Doctor of Philosophy in Computer Science
University of California, Los Angeles, 2021
Professor Cho-Jui Hsieh, Chair

Neural networks provide state-of-the-art results for most machine learning tasks. Unfortunately, neural networks are vulnerable to adversarial examples. That is, a slightly modified example could be easily generated and fool a well-trained image classifier based on deep neural networks (DNNs) with high confidence. This makes it difficult to apply neural networks in security-critical areas.

To find such examples, we first introduce and define adversarial examples. In the first part, we then discuss how to build adversarial attacks in both image and discrete domains. For image classification, we introduce how to design an adversarial attacker in three different settings. Among them, we focus on the most practical setup for evaluating the adversarial robustness of a machine learning system with limited access: the hard-label black-box attack setting for generating adversarial examples, where limited model queries are allowed and only the decision is provided to a queried data input. For the discrete domain, we first talk about its difficulty and introduce how to conduct the adversarial attack on two applications.

While crafting adversarial examples is an important technique to evaluate the robustness

of DNNs, there is a huge need for improving the model robustness as well. Enhancing model robustness under new and even adversarial environments is a crucial milestone toward building trustworthy machine learning systems. In the second part, we talk about the methods to strengthen the model's adversarial robustness. We first discuss attack-dependent defense. Specifically, we first discuss one of the most effective methods for improving the robustness of neural networks: adversarial training and its limitations. We introduce a variant to overcome its problem. Then we take a different perspective and introduce attack-independent defense. We summarize the current methods and introduce a framework-based vicinal risk minimization. Inspired by the framework, we introduce self-progressing robust training. Furthermore, we discuss the robustness trade-off problem and introduce a hypothesis and propose a new method to alleviate it.

The dissertation of Minhao Cheng is approved.

Amit Sahai

Mani Srivastava

Kai-Wei Chang

Cho-Jui Hsieh, Committee Chair

University of California, Los Angeles

2021

To my parents

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

First and foremost, I would like to thank my advisor Cho-Jui Hsieh. When I started my Ph.D. study, I have little knowledge about machine learning and struggled to find a research direction. From the first day, Cho has shown and guided me how to conduct machine learning, find new ideas and formalize the idea into a successful project. It is impossible for me to finish my Ph.D. study with him definitely. I am really grateful and enjoy the time working with him.

I would also thank all of my collaborators throughout the years for the work we have done and my growth as a researcher. Especially, I want to thank Pin-Yu Chen for guiding me into the field of adversarial machine learning and being a caring mentor for my internship at IBM Research. I thank Xiaocheng Tang, Wei Wei, Jinfeng Yi, Sijia Liu. All of you are tremendously knowledgeable and taught me so much.

Many thanks as well to both the current and past members of Cho's Lab both in UC Davis and UCLA. I am grateful to spend a fruitful and enjoyable time with you. Special thanks to my close collaborators Huan Zhang, Xuanqing Liu, Yao Li, Xiangning Chen, and Ruochen Wang. I wish you have great success in the future career!

Finally, my family and friends have been the strongest support throughout my Ph.D. study. It is your encouragement and unconditional support that enabled me to pursue this journey. I would thank my cousin Hongyang Chen who let me first know what scientific research is and endless support from the beginning to the end of my graduate study. Also, I cannot express more gratitude to my parents so that I dedicate this thesis to them. It is you that encouraged me to pursue my dreams from my childhood, help me overcome any difficulties when I was growing up, and continues to be my greatest source of strength.

VITA

2015	B.S. (Computer Science and Technology), University of Electronic Science and Technology of China.
2015–2018	Teaching Assistant, Computer Science Department, UC Davis.
2015–2018	Research Assistant, Computer Science Department, UC Davis.
2018–present	Teaching Assistant, Computer Science Department, UCLA.
2018–present	Research Assistant, Computer Science Department, UCLA.

CHAPTER 1

Introduction

It has been shown that neural networks achieve state-of-art results in nearly every task in both computer vision and natural language processing. Moreover, extensive use of deep learning-based applications can be seen in safety and security-critical environments, such as self-driving cars, malware detection, drones, and robotics where the security requirement is crucial. These developments make security aspects of machine learning increasingly important.

However, recently, it has been shown that neural networks are vulnerable to adversarial examples (SVI16). For example, a slightly modified image can be easily generated and fool a well-trained image classifier based on DNNs with high confidence (GSS15; CW17; ACW18). As shown in Figure 1.1, a bagle image could be turned into a piano classified by neural networks model by only adding a very small human imperceptible perturbation. This problem may get worse if a stop-sign could be recognized as an irrelevant object in the self-driving car system. Similar results could be observed in other domains as well. Table 1.1 has shown a neural network-based text classification model could be easily fooled with only changing a single character. The original text is classified as world news with 57% confidence. However, after changing d in "mood" to P, the classification result becomes Sci/Tech news.

Consequently, the inherent weakness of lacking robustness to adversarial examples for DNNs brings out serious security concerns. Since then, a lot of methods have been proposed to produce those adversarial examples and improve the model's abilities to counter such examples. Specifically, given a victim neural network model and a correctly classified example, an adversarial attack aims to compute a small perturbation such that with this perturbation

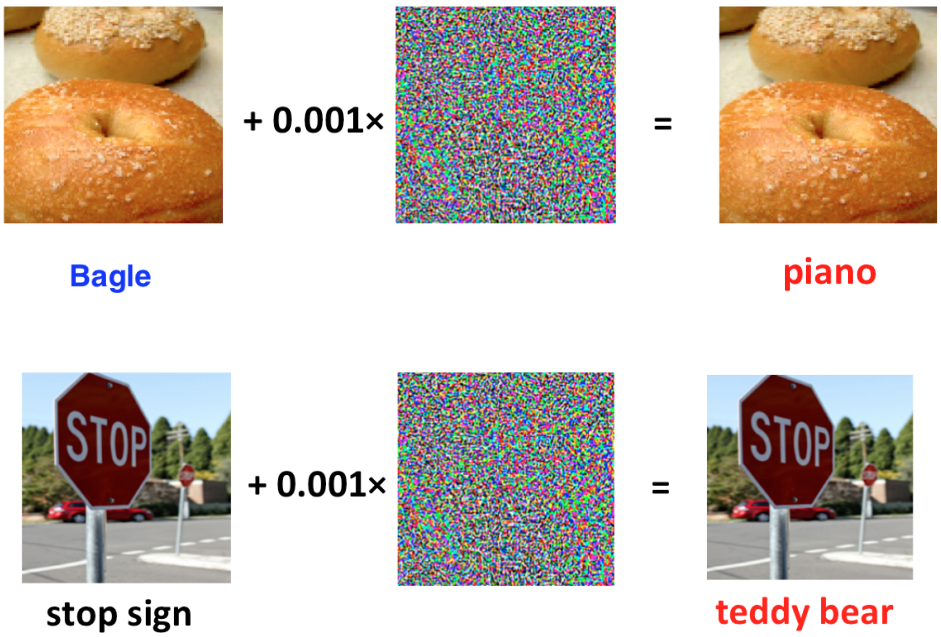


Figure 1.1: Adversarial examples in image classification

Table 1.1: Adversarial examples in text classification

South Africa's historic Soweto township marks its 100th birthday on Tuesday in a mood of optimism. 57% **World**
South Africa's historic Soweto township marks its 100th birthday on Tuesday in a mooP of optimism. 95% **Sci/Tech**

added, the original example will be misclassified. Many adversarial attacks have been proposed in the literature. In the first part, we discuss how to conduct adversarial attacks in different settings. We first start with adversarial attacks on image classification in Chapter 2. Most of them consider the white-box setting, where the attacker has full knowledge about the victim model, and thus gradient-based optimization can be used for the attack. Popular Examples include C&W (CW17) and PGD (MMS18) attacks. On the other hand, some more recent attacks have considered the probability black-box setting where the attacker does not know the victim model's structure and weights, but can iteratively query the model and get the corresponding probability output. In this setting, although gradient (of output probability to the input layer) is not computable, it can still be estimated using finite differences, and many attacks are based on this (CZS17; IEA18; TTC19; JLM18). In this thesis, we also introduce a more piratical setting called hard-label black-box attack, where the attacker can only make queries to acquire the corresponding hard-label decision instead of the probability outputs.

While finding adversarial examples on image domains has been widely discussed, models designed for different tasks are not born equal: some tasks are strictly harder to attack than others. For example, attacking an image is much easier than attacking a text string, since image space is continuous and the adversary can make arbitrarily small changes to the input. Therefore, even if most of the pixels of an image have been modified, the perturbations can still be imperceptible to humans when the accumulated distortion is small. In contrast, text strings live in a discrete space, and word-level manipulations may significantly change the meaning of the text. In this scenario, an adversary should change as few words as possible, and hence this limitation induces a sparse constraint on word-level changes. Likewise, attacking a classifier should also be much easier than attacking a model with sequence outputs. This is because different from the classification problem that has a finite set of discrete class labels, the output space of sequences may have an almost infinite number of possibilities. If we treat each sequence as a label, a targeted attack needs to find a specific one over an enormous number of possible labels, leading to a nearly zero volume in search space. In Chapter 3, we

introduce the challenge and method to evaluate the model's adversarial robustness. Moreover, all the above-mentioned work focus on the static setting, i.e., the input does not depend on the model's output so that one agent's input depends on the other agent's output, which makes the input undecidable in the beginning. Therefore, an adversarial sentence or example is not enough to conduct an attack in dialogue systems. Instead, we, in Section 3.2, propose novel ways to construct an adversarial agent, which can bait the target agent to step to a wrong state and make a bad decision. It is still unknown how to evaluate a model with the interactive input space such as a dialog system or an intelligent agent.

While crafting adversarial examples is an important technique to evaluate the robustness of DNNs, there is a huge need for improving the model's robustness as well. Enhancing model robustness under new and even adversarial environments is a crucial milestone toward building trustworthy machine learning systems. Therefore, in the second part, we discuss the adversarial defense methods. In general, adversarial defense methods could be divided into two categories: attack-dependent defense and attack-independent defenses. For attack-dependent defense, we start with adversarial training-based methods, one of the state-of-the-art robust training algorithms in Chapter 4, where we discuss its limitations and possible solution to overcome them. Specifically, we introduce CAT, a framework that adaptively customizes the perturbation level and the corresponding label for each training sample in adversarial training. Albeit effective, attack-dependent methods have the following limitations: (i) *poor scalability* – the process of generating adversarial examples incurs considerable computation overhead. For instance, our experiments show that, with the same computation resources, standard adversarial training (with 7 attack iterations per sample in every minibatch) of Wide ResNet on CIFAR-10 consumes 10 times more clock time per training epoch when compared with standard training; (ii) *attack specificity* – adversarially trained models are usually most effective against the same attack they trained on, and the robustness may not generalize well to other types of attacks (TB19; KSH19); (iii) *preference toward wider network* – adversarial training is more effective when the networks have sufficient capacity

(e.g., having more neurons in network layers) (MMS18). In Chapter 5, we discuss about the attack-independent attack. In Section 5.1, we first summarize a lot of data augmentation methods used to improve the model's robustness. We then introduce a general framework that formulates robust training objectives via vicinity risk minimization (VRM), which includes many robust training methods as special cases in Section 5.2. Inspired by this framework, we then introduce SPROUT which is short for self-progressing robust training. It is worth noting that the robust training methodology of SPROUT is fundamentally different from adversarial training, as SPROUT features self-adjusted label distribution during training instead of attack generation. In addition to our proposed parametrized label smoothing technique for progressive adjustment of training label distribution, SPROUT also adopts Gaussian augmentation and Mixup (ZCD18) to further enhance robustness. We show that they offer a complementary gain in robustness. However, adversarial defenses often suffer from inferior performance on clean data (ZYJ19; BGH19). This observation has led prior work to extrapolate that a trade-off between robustness and accuracy may be inevitable, particularly for image classification tasks (ZYJ19; TSE19). However, (YRZ20) recently suggests that it is possible to learn classifiers both robust and highly accurate on real image data. The current state of adversarial training methods falls short of this prediction, and the discrepancy remains poorly understood. In Chapter 6, we conduct an in-depth study on understanding the trade-off between robustness and clean accuracy in adversarial training and introduce *Adversarial Masking*, a new hypothesis stating that a widely used technique, batch normalization (BN), has a significant impact on the trade-off between robustness and natural accuracy. Specifically, we break down BN into normalization and rescaling operations and find that the rescaling operation has a significant impact on the robustness trade-off while normalization only has marginal influence. Built upon this observation, we hypothesize that adversarial masking (*i.e.*, the combination of the rescaling operation and the follow-up ReLU activation function) acts as a feature masking layer that can magnify or block feature maps to influence the performance of robust or clean generalization. In this hypothesis,

different rescaling parameters in BN contribute to different adversarial maskings learned through training. By using a simple linear combination of two adversarial maskings, rather than using robust features learned by adversarial training (MMS18; IST19; ZYJ19), we show that a well-balanced trade-off can be readily achieved.

This thesis is organized as follows. In the first part, we talk about how to evaluate adversarial robustness by conducting adversarial attacks. First, in Chapter 2, we start with introducing the problem formulation and different attack settings. We take a deep dive into the hard-label black-box attack where we introduce two optimization-based attacks: Opt-attack and Sign-Opt. In Chapter 3, we extend the attack into the more challenging discrete domain and introduce Seq2sick and AdvAgent to evaluate model robustness on seq2seq model and goal-oriented dialog system. In the second part, we discuss how to enhance model robustness by conducting adversarial defenses. We talk about attack-dependent and attack-independent defense respectively in Chapter 4 and Chapter 5. In Chapter 6, we introduce AdvMask, a hypothesis to explain the adversarial trade-off on generalization, and RobMask, a better-designed normalization technique to boost model generalization. In the end, we conclude the thesis with several future directions in Chapter 7.

Part I

Adversarial Attacks

In this part, we introduce how to evaluate the model's vulnerability towards adversarial examples by conducting adversarial attacks. In Chapter 2, we start with adversarial attacks on image classification tasks. In Section 2.1, we first give an introduction and definition of adversarial example and attack. And then we discuss three attack setting white-box (Section 2.2), soft-label black-box (Section 2.3) and hard-label black-box (Section 2.4) ordered by the available information to attacker. Specifically, we introduce two optimization-based hard-label black-box attacks in detail. We also show the experimental results of adversarial attacks in Section 2.5 and their convergence proofs in Section 2.6. Other than image classification, we extend the discussion on the more challenging discrete domain in Chapter 3. In Section 3.1, We introduce Seq2sick to evaluate the robustness of seq2seq model in NLP tasks such as machine translation or text summarization. In Section 3.2, we take a step further to introduce AdvAgent to evaluate and enhance the robustness of a goal-oriented dialog system.

CHAPTER 2

Adversarial Attack on Image Classification

2.1 Problem Setting

In this section, we introduce and define the adversarial examples and attacks. For classification task, we consider attacking a K-way multi-class classification model in the thesis. Given the classification model $f : \mathbb{R}^d \to \{1, \ldots, K\}$ and an original example $\boldsymbol{x}_0$, the goal is to generate an **adversarial example** $\boldsymbol{x}$ such that

$$\boldsymbol{x} \text{ is close to } \boldsymbol{x}_0 \quad \text{and} \quad \operatorname*{argmax}_i f(\boldsymbol{x})_i \neq \operatorname*{argmax}_i f(\boldsymbol{x}_0)_i \tag{2.1}$$

i.e., $\boldsymbol{x}$ has a different prediction with $\boldsymbol{x}_0$ by model f.

That is called **untargeted attack** since we only search for an input $\boldsymbol{x}$ so that $f(\boldsymbol{x}) \neq f(\boldsymbol{x}_0)$ and $\boldsymbol{x}$, $\boldsymbol{x}_0$ are close. The closeness of $\boldsymbol{x}$, $\boldsymbol{x}_0$ is defined in some distance metrics according different tasks. We defer the discussion in Sec 2.1.1. However, a more powerful attack requires to fool the classifier to any specified class t, i.e target class , instead of any class. Usually, $t \neq \operatorname{argmax}_i f(\boldsymbol{x}_0)$. Instead, we define **targeted attack** as we want to generated an adversarial example $\boldsymbol{x}$ such that

$$\boldsymbol{x} \text{ is close to } \boldsymbol{x}_0 \quad \text{and} \quad \operatorname*{argmax}_i f(\boldsymbol{x})_i = t \tag{2.2}$$

i.e., $\boldsymbol{x}$ is classified as a target class t by model f.

2.1.1 Distance Metric

In the definition of adversarial examples, we require use of a distance metric to quantify similarity. A common choice of the metric would be L_p norm. Formally,

$$\|\boldsymbol{x}\|_p = (|\boldsymbol{x}_1|^p + |\boldsymbol{x}_2|^p + \cdots + |\boldsymbol{x}_n|^p + \boldsymbol{x}_{n+1}|^p + \dots)^{1/p} \tag{2.3}$$

Among them, L_2 and L_∞ are the most used metric in the image domain. To be specific,

- L_2 distance measures the standard Euclidean (rootmean-square) distance between $\boldsymbol{x}$ and $\boldsymbol{x}_0$. The L_2 distance can remain small when there are many small changes to many pixels. This distance metric was used in the initial adversarial example work (GSS14).

- L_∞ distance measures the maximum change to any of the coordinates:

$$\|\boldsymbol{x}\|_\infty = \sup(|\boldsymbol{x}_1|, |\boldsymbol{x}_2|, \dots) \tag{2.4}$$

  For images, we can imagine there is a maximum budget,and each pixel is allowed to be changed by up to this limit, with no limit on the number of pixels that are modified.

In other domains such as text related domain like Natural Language Processing (NLP), the story becomes different. For example, since image space is continuous and the adversary can make arbitrarily small changes to the input, therefore, even if most of the pixels of an image have been modified, the perturbations can still be imperceptible to humans when the accumulated distortion is small. In contrast, text strings live in a discrete space, and word-level manipulations may significantly change the meaning of the text. In this scenario, an adversary should change **as few words as possible**, and hence this limitation induces a sparse constraint on word-level changes.

However, there is no certain conclusion to judge which distance metric is optimal. Constructing and evaluating a good distance metric is an important research direction need to

be explored.

2.2 White-box Adversarial Attacks

In this section, we discuss the most discussed attack algorithms in the literature: **white-box** setting.

In the white-box setting, the classifier f is exposed to the attacker. For neural networks, under this assumption, back-propagation can be conducted on the target model because both network structure and weights are known by the attacker. For classification models in neural networks, it is usually assumed that model prediction is $\text{argmax}_i(Z(\boldsymbol{x})_i)$ or $\text{argmax}_i\, f(\boldsymbol{x})_i$, where $Z(\boldsymbol{x}) \in \mathbb{R}^K$ is the final (logit) layer output, and $Z(\boldsymbol{x})_i$ is the prediction score for the i-th class. The objectives in (2.1) can then be naturally formulated as the following optimization problem:

$$\underset{\boldsymbol{x}}{\text{argmin}}\ \{\text{Dis}(\boldsymbol{x}, \boldsymbol{x}_0) + c\mathcal{L}(Z(\boldsymbol{x}))\} := h(\boldsymbol{x}), \tag{2.5}$$

where $\text{Dis}(\cdot,\cdot)$ is some distance measurement (e.g., L_2, L_1 or L_∞ norm in Euclidean space), $\mathcal{L}(\cdot)$ is the loss function corresponding to the goal of the attack, and c is a balancing parameter. For *untargeted attack*, where the goal is to make the target classifier misclassify, the loss function can be defined as

$$\mathcal{L}(Z(\boldsymbol{x})) = \max\{[Z(\boldsymbol{x})]_{y_0} - \max_{i \neq y_0}[Z(\boldsymbol{x})]_i, -\kappa\}, \tag{2.6}$$

where y_0 is the original label predicted by the classifier, κ is the margin (usually set to be 1 or 0) of the hinge loss. For *targeted attack*, where the goal is to turn it into a specific target class t, the loss function can also be defined accordingly as

$$\mathcal{L}(Z(\boldsymbol{x})) = \max\{[Z(\boldsymbol{x})]_t - \max_{i \neq t}[Z(\boldsymbol{x})]_i, -\kappa\}, \tag{2.7}$$

For Non-overlapping Attack, we let $\mathbf{s} = \{s_1, \ldots, s_M\}$ be the original output sequence, where s_i denotes the location of the i-th word in the output vocabulary ν. $\{z_1, \ldots, z_M\}$ indicates the logit layer outputs of the adversarial example. In the non-overlapping attack, the output of adversarial example should be entirely different from the original output $\mathbf{S}$, i.e.,

$$s_t \neq \text{argmax}_{y \in \nu}\, z_t^{(y)}, \quad \forall t = 1, \ldots, M,$$

which is equivalent to

$$z_t^{(s_t)} < \max_{y \in \nu,\ y \neq s_t} z_t^{(y)}, \quad \forall t = 1, \ldots, M.$$

Given this observation, we can define a hinge-like loss function L to generate adversarial examples in the non-overlapping attack, i.e.,

$$L_{\text{non-overlapping}} = \sum\nolimits_{t=1}^{M} \max\{-\epsilon,\ z_t^{(s_t)} - \max_{y \neq s_t}\{z_t^{(y)}\}\}, \tag{2.8}$$

where $\epsilon \geq 0$ denotes the confidence margin parameter. Generally speaking, a larger ϵ will lead to a more confident output and a higher success rate, but with the cost of more iterations and longer running time.

For Targeted Keywords Attack, we do not specify the positions of the targeted keywords in the output sentence. Instead, it is more natural to design a loss function that allows the targeted keywords to become the top-1 prediction at any positions. The attack is considered as successful only when ALL the targeted keywords appear in the output sequence. Therefore, the more targeted keywords there are, the harder the attack is. To illustrate our method, we start from the simpler case with only one targeted keyword k_1. To ensure that the target keyword word's logit $z_t^{(k_1)}$ be the largest among all the words at a position t, we design the

following loss function:

$$L = \min_{t\in[M]}\{\max\{-\epsilon,\ \max_{y\neq k_1}\{z_t^{(y)}\} - z_t^{(k_1)}\}\}, \tag{2.9}$$

which essentially searches the minimum of the hinge-like loss terms over all the possible locations $t \in [M]$. When there exist more than one targeted keywords $K = \{k_1, k_2, \ldots, k_{|K|}\}$, where k_i denotes the i-th word in output vocabulary ν, we follow the same idea to define the loss function as follows:

$$L_{\text{keywords}} = \sum_{i=1}^{|K|} \min_{t\in[M]}\{\max\{-\epsilon, \max_{y\neq k_i}\{z_t^{(y)}\} - z_t^{(k_i)}\}\}. \tag{2.10}$$

However, the loss defined in (2.10) suffers from the "keyword collision" problem. When there are more than one keyword, it is possible that multiple keywords compete at the same position to attack. To address this issue, we define a mask function m to mask off the position if it has been already occupied by one of the targeted keywords:

$$m_t(x) = \begin{cases} +\infty & \text{if } \operatorname{argmax}_{i\in\nu} z_t^{(i)} \in K \\ x & \text{otherwise} \end{cases} \tag{2.11}$$

In other words, if any of the keywords appear at position t as the top-1 word, we ignore that position and only consider other positions for the placement of remaining keywords. By incorporating the mask function, the final loss for targeted keyword attack becomes:

$$\sum_{i=1}^{|K|} \min_{t\in[M]}\{m_t(\max\{-\epsilon,\ \max_{y\neq k_i}\{z_t^{(y)}\} - z_t^{(k_i)}\})\}. \tag{2.12}$$

Therefore, attacking a machine learning model can be posed as solving this optimization problem (CW17; CSZ18), which is also known as the **C&W attack** or the **EAD attack** depending on the choice of the distance measurement. To solve (2.5), one can apply any

gradient-based optimization algorithm such as SGD or Adam, since the gradient of $\mathcal{L}(Z(\boldsymbol{x}))$ can be computed via back-propagation.

At the same time $\mathcal{L}$ could be defined by using other loss function as well. **Fast Gradient Sign Method:** (GSS14) proposed an algorithm called fast gradient sign method (FGSM) to craft adversarial examples. Originated from an L_∞ constraint on the maximal distortion, FGSM uses the sign of the gradient w.r.t the input image $\boldsymbol{x}_0$ to generate adversarial examples. The formula for generating FGSM adversarial example is shown below:

$$\boldsymbol{x} = \Pi_{\boldsymbol{x}\in\mathcal{B}(\boldsymbol{x}_0,\epsilon)}\left\{\boldsymbol{x}_0 + \alpha\cdot\operatorname{sign}\Big(\nabla_{\boldsymbol{x}} f(\boldsymbol{x}_0)\Big)\right\},$$

where $\nabla_{\boldsymbol{x}} f(\boldsymbol{x}_0)$ denotes the gradient of the classifier w.r.t the input image $\boldsymbol{x}_0$. α represents the step-size of the one step distortion, $\mathcal{B}(\boldsymbol{x}_0,\epsilon)$ denotes the ℓ_p-norm ball centered at $\boldsymbol{x}_0$ with radius ϵ and Π_Φ is the projection to the set Φ. The final adversarial image $\boldsymbol{x}^{adv}$ will be a point within the ϵ-ball around the original image $\boldsymbol{x}_0$. Larger ϵ makes it easier to get successful attack but the adversarial image will be further from the original image.

The Basic Iterative Method and **Projected-Gradient Descent Attack**: The Basic Iterative Method (BIM) (KGB16) and the projected gradient descent attack (PGD) (MMS18) could be seen as a iterative variants of FGSM. The PGD attack updates in the direction that decreases the probability of the original class most, then projects the result back to the ϵ-ball of the input. PGD can be viewed as iterative-FGSM. An advantage of PGD attack over C&W attack is that it allows direct control of distortion level by changing ϵ, while for C&W attack, one can only do so indirectly via hyper-parameter tuning.

Starting from $\boldsymbol{x}^0 = \boldsymbol{x}_0$, PGD attack conducts projected gradient descent iteratively to update the adversarial example:

$$\boldsymbol{x}^{t+1} = \Pi_{\boldsymbol{x}\in\mathcal{B}(\boldsymbol{x}_0,\epsilon)}\left\{\boldsymbol{x}^t + \alpha\cdot\operatorname{sign}\Big(\nabla_{\boldsymbol{x}} f(\boldsymbol{x}^t)\Big)\right\},$$

where α is the step size. The number of iterations depends on the data.

The ability of computing gradient also enables many different attacks in the white-box setting. For example, eq (2.5) can also be turned into a constrained optimization problem, which can then be solved by projected gradient descent (PGD) (MMS18). Other algorithms such as Deepfool (MFF16) also solve similar optimization problems to construct adversarial examples.

2.3 Soft-label Black-box Attacks

In this section, we discuss a practical attack setting called soft-label Black-box attack.

In real-world systems, usually the underlying machine learning model will not be revealed and thus white-box attacks cannot be applied. This motivates the study of attacking machine learning models in the **black-box setting**, where attackers do not have any information about the function f. And the only valid operation is to make queries to the model and acquire the corresponding output $f(\boldsymbol{x})$. The first approach for black-box attack is using transfer attack (PMG17) – instead of attacking the original model f, attackers try to construct a substitute model $\hat{f}$ to mimic f and then attack $\hat{f}$ using white-box attack methods. This approach has been well studied and analyzed in (LCL16; BHL17). However, recent papers have shown that attacking the substitute model usually leads to much larger distortion and low success rate (CZS17). Therefore, instead, (CZS17) considers the **soft-label black-box** setting, where attackers can use $\boldsymbol{x}$ to query the softmax layer output in addition to the final classification result. It generate adversarial examples based on the approximated gradient based on the approximated gradient.

Zeroth Order Optimization Based Attack (ZOO): (CZS17) proposed to use a finite difference method to approximate the gradient of loss w.r.t the input image. Then C&W

attack is applied to generate the adversarial image. The formula of estimating the gradient is:

$$\frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}_{(i)}} \approx \frac{f(\boldsymbol{x} + h\boldsymbol{e}_i) - f(\boldsymbol{x} - h\boldsymbol{e}_i)}{2h},$$

where h is a small constant and $\boldsymbol{e}_i$ is a standard basis vector with only the ith component as 1, and i ranges from 1 to the dimension of input image.

The time used to estimate the gradient grows with the dimension of the input image. When the dimension of the image is large, the author introduced several techniques to scale-up the method. The method is able to craft adversarial examples in reasonable time for large Deep Neural Networks trained on ImageNet consists of large natural images.

NES Attack: (IEA18) introduced a score-based adversarial attack method, which uses natural evolutionary strategies (NES) to estimate the gradient of loss w.r.t the input image then generates adversarial examples based on the estimated gradient.

Algorithm 1 NES Gradient Estimate

Input: classifier $f(\cdot)$, image $\boldsymbol{x}$, variance σ.

Output: $\nabla_{\boldsymbol{x}} f(\boldsymbol{x})$

for $i = 1$ to T **do**

    $\boldsymbol{u}_i \leftarrow \mathcal{N}(\boldsymbol{0}_D, \boldsymbol{I}_{D \cdot D})$

    $\boldsymbol{g} \leftarrow \boldsymbol{g} + f(\boldsymbol{x} + \sigma \cdot \boldsymbol{u}_i) \cdot \boldsymbol{u}_i$

    $\boldsymbol{g} \leftarrow \boldsymbol{g} - f(\boldsymbol{x} - \sigma \cdot \boldsymbol{u}_i) \cdot \boldsymbol{u}_i$

Return $\frac{1}{2n\sigma}\boldsymbol{g}$

Note that $D = w \cdot h \cdot c$ is the dimension of the input image.

The author also extends the method to partial-information setting, where only part of the probabilities or top-k sorted labels are given.

In this case, they can reconstruct the loss function (2.6) and evaluate it as long as the objective function $h(\boldsymbol{x})$ exists for any $\boldsymbol{x}$. Thus a zeroth order optimization approach can

be directly applied to minimize $h(\boldsymbol{x})$. (TTC18) further improves the query complexity of (CZS17) by introducing an autoencoder-based approach to reduce query counts and an adaptive random gradient estimation to balance query counts and distortion.

2.4 Hard-label Black-box Attacks

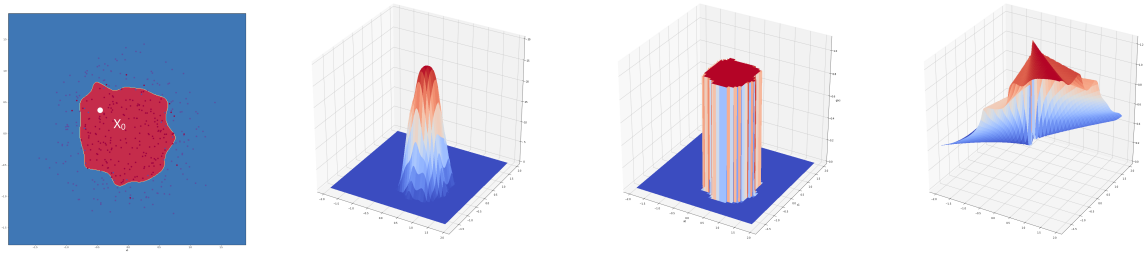
In this section, we discuss a stricter and more practical attack setting called hard-label black-box attack where only the top-1 predicted label is available to attackers.

2.4.1 Difficulty of Hard-label Black-box Attacks

The **hard-label black-box setting** refers to cases where real-world ML systems only provide limited prediction results of an input query. Specifically, only the final decision (top-1 predicted label) instead of probability outputs is known to an attacker.

Attacking in this setting is indeed very challenging. In Figure 2.1a, we show a simple 3-layer neural network's decision boundary. Note that the $\mathcal{L}(Z(\boldsymbol{x}))$ term is continuous as in Figure 2.1b because the logit layer output is real-valued functions. However, in the hard-label black-box setting, only $f(\cdot)$ is available instead of $Z(\cdot)$. Since $f(\cdot)$ can only be a one-hot vector, if we plug-in f into the loss function, $\mathcal{L}(f(\boldsymbol{x}))$ (as shown in Figure 2.1c) will be discontinuous and with discrete outputs.

Optimizing this function will require combinatorial optimization or search algorithms, which is challenging given the high dimensionality of the problem. The only two current approaches (BRB17; IEA18) are based on random-walk on the boundary and random trails on the loss function. Although these "Boundary attack" and "Limited attack" can find adversarial examples with comparable distortion with white-box attacks, they need lots of queries to explore the high-dimensional space and lack convergence guarantees. We show that our optimization-based algorithm can significantly reduce the number of queries, and has guaranteed convergence in the number of iterations (queries) when the objective function



(a) Decision boundary of $f(\boldsymbol{x})$ (b) $\mathcal{L}(Z(\boldsymbol{x}))$ (c) $\mathcal{L}(f(\boldsymbol{x}))$ (d) $g(\boldsymbol{\theta})$

Figure 2.1: (a) A neural network classifier. (b) illustrates the loss function of C&W attack, which is continuous and hence can be easily optimized. (c) is the C&W loss function in the hard-label setting, which is discrete and discontinuous. (d) our proposed attack objective $g(\boldsymbol{\theta})$ for this problem, which is continuous and easier to optimize. See detailed discussions in Section 3.

is lipschitz smooth.

2.4.2 Opt-attack: A Query-Efficient Hard-label Black-box based on Optimization Approach

Now we introduce a novel way to re-formulate hard-label black-box attack as another optimization problem, show how to evaluate the function value using hard-label queries, and then apply a zeroth order optimization algorithm to solve it.

A Boundary-based Re-formulation For a given example $\boldsymbol{x}_0$, true label y_0 and the hard-label black-box function $f : \mathbb{R}^d \to \{1, \ldots, K\}$, we define our objective function $g : \mathbb{R}^d \to \mathbb{R}$ depending on the type of attack:

$$\textbf{Untargeted attack:} \quad g(\boldsymbol{\theta}) = \min_{\lambda>0} \lambda \quad \texttt{s.t} \quad f(\boldsymbol{x}_0 + \lambda \frac{\boldsymbol{\theta}}{||\boldsymbol{\theta}||}) \neq y_0 \tag{2.13}$$

$$\textbf{Targeted attack (given target } t\textbf{):} \quad g(\boldsymbol{\theta}) = \min_{\lambda>0} \lambda \quad \texttt{s.t} \quad f(\boldsymbol{x}_0 + \lambda \frac{\boldsymbol{\theta}}{||\boldsymbol{\theta}||}) = t \tag{2.14}$$

In this formulation, $\boldsymbol{\theta}$ represents the search direction and $g(\boldsymbol{\theta})$ is the distance from $\boldsymbol{x}_0$ to the nearest adversarial example along the direction $\boldsymbol{\theta}$. The difference between (2.13) and (2.14) corresponds to the different definitions of "successfulness" in untargeted and targeted

attack, where the former one aims to turn the prediction into any incorrect label and the later one aims to turn the prediction into the target label. For untargeted attack, $g(\boldsymbol{\theta})$ also corresponds to the distance to the decision boundary along the direction $\boldsymbol{\theta}$. In image problems the input domain of f is bounded, so we will impose corresponding upper/lower bounds in the definition of (2.13) and (2.14).

Instead of searching for an adversarial example, we search the direction $\boldsymbol{\theta}$ to minimize the distortion $g(\boldsymbol{\theta})$, which leads to the following optimization problem:

$$\min_{\boldsymbol{\theta}} \; g(\boldsymbol{\theta}). \tag{2.15}$$

Finally, the adversarial example can be found by $\boldsymbol{x}^* = \boldsymbol{x}_0 + g(\boldsymbol{\theta}^*)\frac{\boldsymbol{\theta}^*}{\|\boldsymbol{\theta}^*\|}$, where $\boldsymbol{\theta}^*$ is the optimal solution of (2.15).

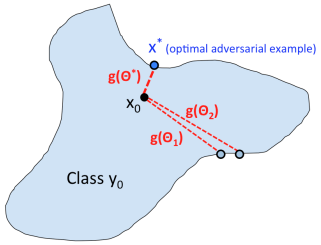


Figure 2.2: Illustration

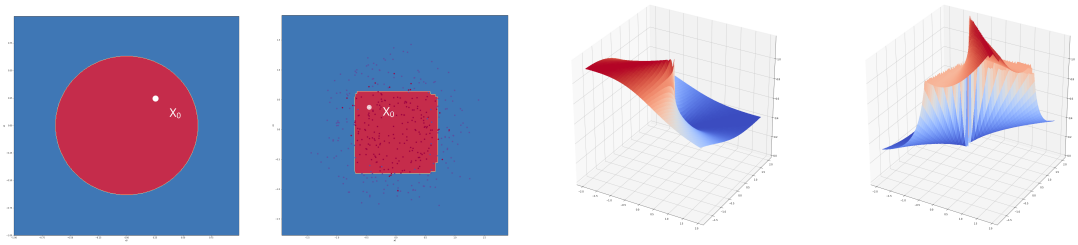
Note that unlike the C&W or PGD objective functions, which are discontinuous step functions in the hard-label setting, $g(\boldsymbol{\theta})$ maps input direction to real-valued output (distance to decision boundary), which is usually continuous – a small change of $\boldsymbol{\theta}$ usually leads to a small change of $g(\boldsymbol{\theta})$, as can be seen from Figure 2.2.

Moreover, we give three examples of $f(\boldsymbol{x})$ defined in two dimension input space and their corresponding $g(\boldsymbol{\theta})$. In Figure 2.3a, we have a continuous classification function defined as follows

$$f(\boldsymbol{x}) = \begin{cases} 1, & \text{if } \|\boldsymbol{x}\|_2^2 \geq 0.4 \\ 0, & \text{otherwise.} \end{cases}$$

In this case, as shown in Figure 2.3c, $g(\boldsymbol{\theta})$ is continuous. Moreover, in Figure 2.3b and Figure 2.1a, we show decision boundaries generated by GBDT and neural network classifier, which are not continuous. However, as showed in Figure 2.3d and Figure 2.1d, even if the classifier function is not continuous, $g(\boldsymbol{\theta})$ is still continuous. This makes it easy to apply zeroth order method to solve (2.15).

Compute $g(\boldsymbol{\theta})$ up to certain accuracy. We are not able to evaluate the gradient of



(a) Decision boundary of continuous function

(b) Decision boundary of GBDT

(c) $g(\boldsymbol{\theta})$ of (a)

(d) $g(\boldsymbol{\theta})$ of (b)

Figure 2.3: Examples of decision boundary of classification function $f(\boldsymbol{x})$ and corresponding $g(\boldsymbol{\theta})$.

g, but we can evaluate the function value of g using the hard-label queries to the original function f. For simplicity, we focus on untargeted attack here, but the same procedure can be applied to targeted attack as well.

First, we discuss how to compute $g(\boldsymbol{\theta})$ directly without additional information. This is used in the initialization step of our algorithm. For a given normalized $\boldsymbol{\theta}$, we do a coarse-grained search and then a binary search. In coarse-grained search, we query the points $\{\boldsymbol{x}_0 + \alpha\boldsymbol{\theta}, \boldsymbol{x}_0 + 2\alpha\boldsymbol{\theta}, \dots\}$ one by one until we find $f(\boldsymbol{x} + i\alpha\boldsymbol{\theta}) \neq y_0$. This means the boundary lies between $[\boldsymbol{x}_0 + (i-1)\alpha\boldsymbol{\theta}, \boldsymbol{x}_0 + i\alpha\boldsymbol{\theta}]$. We then enter the second phase and conduct a binary search to find the solution within this region (same with line 11–17 in Algorithm 2). Note that there is an upper bound of the first stage if we choose $\boldsymbol{\theta}$ by the direction of $\boldsymbol{x} - \boldsymbol{x}_0$ with some $\boldsymbol{x}$ from another class. This procedure is used to find the initial $\boldsymbol{\theta}_0$ and corresponding $g(\boldsymbol{\theta}_0)$ in our optimization algorithm. We omit the detailed algorithm for this part since it is similar to Algorithm 2.

Next, we discuss how to compute $g(\boldsymbol{\theta})$ when we know the solution is very close to a reference point v. This is used in all the function evaluations in our optimization algorithm, since the current solution is usually close to the previous solution, and when we estimate the gradient using (2.16), the queried direction will only be a slight modification of the previous one. In this case, we first increase or decrease v in the local region to find the interval that

contains the nearby boundary (e.g, $f(\boldsymbol{x}_0 + v\boldsymbol{\theta}) = y_0$ and $f(\boldsymbol{x}_0 + v'\boldsymbol{\theta}) \neq y_0$), then conduct a binary search to find the final value of g. Our procedure for computing the g value is presented in Algorithm 2.

Algorithm 2 Compute $g(\boldsymbol{\theta})$ locally

Input: Hard-label model f, original image x_0, query direction $\boldsymbol{\theta}$, previous solution v, increase/decrease ratio $\alpha = 0.01$, stopping tolerance ϵ (maximum tolerance of computed error)
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}/||\boldsymbol{\theta}||$
if $f(\boldsymbol{x}_0 + v\boldsymbol{\theta}) = y_0$ **then**
    $v_{left} \leftarrow v, v_{right} \leftarrow (1+\alpha)v$
    while $f(\boldsymbol{x}_0 + v_{right}\boldsymbol{\theta}) = y_0$ **do**
        $v_{right} \leftarrow (1+\alpha)v_{right}$
else
    $v_{right} \leftarrow v, v_{left} \leftarrow (1-\alpha)v$
    while $f(\boldsymbol{x}_0 + v_{left}\boldsymbol{\theta}) \neq y_0$ **do**
        $v_{left} \leftarrow (1-\alpha)v_{left}$
Binary Search within $[v_{left}, v_{right}]$
while $v_{right} - v_{left} > \epsilon$ **do**
    $v_{mid} \leftarrow (v_{right} + v_{left})/2$
    if $f(\boldsymbol{x}_0 + v_{mid}\boldsymbol{\theta}) = y_0$ **then**
        $v_{left} \leftarrow v_{mid}$
    else
        $v_{right} \leftarrow v_{mid}$
return v_{right}

Hard-label Black-box Attacks with L_∞ norm constraint Although we could let $\|\boldsymbol{\theta}\| = \|\boldsymbol{\theta}\|_\infty$ in (2.13) and (2.14) directly, $g(\boldsymbol{\theta})$ will be harder to optimize in practice because of introducing the max term in $\|\cdot\|_\infty$. Instead, with an L_∞ constraint ε , we design a smooth approximation loss as follows:

$$\textbf{Untargeted attack:} \quad g(\boldsymbol{\theta}) = \min_{\lambda}\{\sum_{i=1}^{d}(\max\{\lambda\frac{|\boldsymbol{\theta}_i|}{\|\boldsymbol{\theta}\|_\infty} - \varepsilon, 0\})^2\} \texttt{ s.t } f(\boldsymbol{x}_0 + \lambda\frac{\boldsymbol{\theta}}{\|\boldsymbol{\theta}\|_\infty}) \neq y_0$$

$$\textbf{Targeted attack:} \quad g(\boldsymbol{\theta}) = \min_{\lambda}\{\sum_{i=1}^{d}(\max\{\lambda\frac{|\boldsymbol{\theta}_i|}{\|\boldsymbol{\theta}\|_\infty} - \varepsilon, 0\})^2\} \texttt{ s.t } f(\boldsymbol{x}_0 + \lambda\frac{\boldsymbol{\theta}}{\|\boldsymbol{\theta}\|_\infty}) = t.$$

Here $\boldsymbol{\theta}_i$ is the i-th coordinate of $\boldsymbol{\theta}$. Notably, when $\lambda \leq \varepsilon$, we have $g(\boldsymbol{\theta}) = 0$. That's to say, we have obtained a legitimate $\boldsymbol{\theta}$ to make a valid adversarial example $\boldsymbol{x}_0 + \lambda^* \frac{\boldsymbol{\theta}}{\|\boldsymbol{\theta}\|_\infty}$.

Zeroth Order Optimization To solve the optimization problem (2.15) for which we can only evaluate function value instead of gradient, zeroth order optimization algorithms can be naturally applied. In fact, after the reformulation, the problem can be potentially solved by any zeroth order optimization algorithm, like zeroth order gradient descent, genetic algorithm (ASC19) or coordinate descent (see (CSV09) for a comprehensive survey).

Here we propose to solve (2.1) using Randomized Gradient-Free (RGF) method proposed in (NS17; GL13). In practice, we found it outperforms zeroth-order coordinate descent. At each iteration, the gradient is estimated by

$$\hat{\boldsymbol{g}} = \frac{g(\boldsymbol{\theta} + \beta \boldsymbol{u}) - g(\boldsymbol{\theta})}{\beta} \cdot \boldsymbol{u} \tag{2.16}$$

where $\boldsymbol{u}$ is a random Gaussian vector, and $\beta > 0$ is a smoothing parameter (we set $\beta = 0.005$ in all our experiments). The solution is then updated by $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \hat{\boldsymbol{g}}$ with a step size η. The procedure is summarized in Algorithm 3.

Algorithm 3 RGF for hard-label black-box attack

Input: Hard-label model f, original image x_0, initial $\boldsymbol{\theta}_0$.
for $t = 0, 1, 2, \ldots, T$ **do**
    Randomly choose $\boldsymbol{u}_t$ from a zero-mean Gaussian distribution
    Evaluate $g(\boldsymbol{\theta}_t)$ and $g(\boldsymbol{\theta}_t + \beta \boldsymbol{u})$ using Algorithm 2
    Compute $\hat{\boldsymbol{g}} = \frac{g(\boldsymbol{\theta}_t + \beta \boldsymbol{u}) - g(\boldsymbol{\theta}_t)}{\beta} \cdot \boldsymbol{u}$
    Update $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \hat{\boldsymbol{g}}$
return $\boldsymbol{x}_0 + g(\boldsymbol{\theta}_T)\boldsymbol{\theta}_T$

Also, if $g(\boldsymbol{\theta})$ is Lipschitz-smooth, we are able to bound the number of iterations needed with $O(\frac{d}{\delta^2})$ for our algorithm to achieve stationary points.

Theoretical Analysis If $g(\boldsymbol{\theta})$ can be computed exactly, it has been proved in (NS17) that RGF in Algorithm 3 requires at most $O(\frac{d}{\delta^2})$ iterations to converge to a point with $\|\nabla g(\boldsymbol{\theta})\|^2 \leq \delta^2$. However, in our algorithm the function value $g(\boldsymbol{\theta})$ cannot be computed exactly; instead, we compute it up to ϵ-precision, and this precision can be controlled by binary threshold in Algorithm 2. We thus extend the proof in (NS17) to include the case of approximate function value evaluation, as described in the following theorem.

Theorem 1. *In Algorithm 3, suppose g has Lipschitz-continuous gradient with constant $L_1(g)$. If the error of function value evaluation is controlled by $\epsilon \sim O(\beta\delta^2)$ and $\beta \leq O(\frac{\delta}{dL_1(g)})$, then in order to obtain $\frac{1}{N+1}\sum_{k=0}^{N} E_{\mathcal{U}_k}(\|\nabla g(\boldsymbol{\theta}_k)\|^2) \leq \delta^2$, the total number of iterations is at most $O(\frac{d}{\delta^2})$.*

Detailed proofs can be found in Section 2.6.0.1. Note that the binary search procedure could obtain the desired function value precision in $O(\log \delta)$ steps. By using the same idea with Theorem 1 and following the proof in (NS17), we could also achieve $O(\frac{d^2}{\delta^3})$ complexity when $g(\boldsymbol{\theta})$ is non-smooth but Lipschitz continuous.

Implementation details There are several implementation details when we apply this algorithm. First, for high-dimensional problems, we found the estimation in (2.16) is very noisy. Therefore, instead of using one vector, we sample q vectors from Gaussian distribution and average their estimators to get $\hat{\boldsymbol{g}}$. We set $q = 20$ in all the experiments. Second, instead of using a fixed step size (suggested in theory), we use a backtracking line-search approach to find step size at each step. This leads to additional query counts, but makes the algorithm more stable and eliminates the need to hand-tuning the step size. Third, instead of using a random direction $\boldsymbol{\theta}$ as initialization, we sample t vectors from Gaussian distribution and choose the one with smallest $g(\boldsymbol{\theta})$ as our initialization. It helps us to find a good initialization direction and thus get a smaller distortion in the end with limited number of additional queries. We set $t = 100$ in all the experiments.

2.4.3 Sign-OPT: Using Gradient Sign to Further Gain Query Efficiency

In this part, we introduce an algorithm that hugely improves the query complexity over Opt attack. Our algorithm is based on the following key ideas: (i) one does not need very accurate values of directional derivative in order to make the algorithm converge, and (ii) there exists an **imperfect but informative estimation** of directional derivative of g that can be computed by a single query.

A single query oracle As mentioned before, the previous approach requires computing $g(\boldsymbol{\theta} + \epsilon \boldsymbol{u}) - g(\boldsymbol{\theta})$ which consumes a lot of queries. However, based on the definition of $g(\cdot)$, we can compute the sign of this value $\text{sign}(g(\boldsymbol{\theta} + \epsilon \boldsymbol{u}) - g(\boldsymbol{\theta}))$ using a single query. Considering the untargeted attack case, the sign can be computed by

$$\text{sign}(g(\boldsymbol{\theta}+\epsilon\mathbf{u})-g(\boldsymbol{\theta})) = \begin{cases} +1, & f(x_0 + g(\boldsymbol{\theta})\frac{(\boldsymbol{\theta}+\epsilon\mathbf{u})}{\|\boldsymbol{\theta}+\epsilon\mathbf{u}\|}) = y_0, \\ -1, & \text{Otherwise.} \end{cases} \tag{2.17}$$

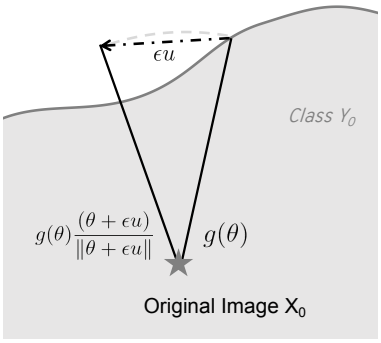


Figure 2.4: Illustration

This is illustrated in Figure 2.4. Essentially, for a new direction $\boldsymbol{\theta} + \epsilon\mathbf{u}$, we test whether a point at the original distance $g(\boldsymbol{\theta})$ from x_0 in this direction lies inside or outside the decision boundary, i.e. if the produced perturbation will result in a wrong prediction by classifier. If the produced perturbation is outside the boundary i.e. $f(x_0 + g(\boldsymbol{\theta})\frac{(\boldsymbol{\theta}+\epsilon\mathbf{u})}{\|\boldsymbol{\theta}+\epsilon\mathbf{u}\|}) \neq y_0$, the new direction has a smaller distance to decision boundary, and thus giving a smaller value of g. It indicates that $\mathbf{u}$ is a descent direction to minimize g.

Sign-OPT attack By sampling random Gaussian vector Q times, we can estimate the imperfect gradient by

$$\hat{\nabla} g(\boldsymbol{\theta}) \approx \hat{\boldsymbol{g}} := \sum_{q=1}^{Q} \operatorname{sign}(g(\boldsymbol{\theta} + \epsilon \mathbf{u}_q) - g(\boldsymbol{\theta}))\mathbf{u}_q, \tag{2.18}$$

which only requires Q queries. We then use this imperfect gradient estimate to update our search direction $\boldsymbol{\theta}$ as $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \hat{\mathbf{g}}$ with a step size η and use the same search procedure to compute $g(\boldsymbol{\theta})$ up to a certain accuracy. The detailed procedure is shown in Algorithm 4.

Algorithm 4 Sign-OPT attack

Input: Hard-label model f, original image x_0, initial $\boldsymbol{\theta}_0$

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

Randomly sample $u_1, \ldots, u_Q$ from a Gaussian or Uniform distribution

Evaluate $g(\boldsymbol{\theta}_t)$

~~$\hat{\boldsymbol{g}} = \frac{g(\boldsymbol{\theta}_t + \beta \boldsymbol{u}) - g(\boldsymbol{\theta}_t)}{\beta} \cdot \boldsymbol{u}$~~ $\Rightarrow \mathbf{sign}\left(\frac{g(\boldsymbol{\theta}_t + \beta \boldsymbol{u}) - g(\boldsymbol{\theta}_t)}{\beta}\right) \cdot \boldsymbol{u}$

Update $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \eta \hat{\boldsymbol{g}}$

Evaluate $g(\boldsymbol{\theta}_t)$ using the same search algorithm in Algorithm 2

We note that (LCC19) designed a Zeroth Order SignSGD algorithm for soft-label black box attack (not hard-label setting). They use $\hat{\nabla} g(\boldsymbol{\theta}) \approx \hat{\boldsymbol{g}} := \sum_{q=1}^{Q} \operatorname{sign}(g(\boldsymbol{\theta} + \epsilon \mathbf{u}_q) - g(\boldsymbol{\theta})\mathbf{u}_q)$ and shows that it could achieve a comparable or even better convergence rate than zeroth order stochastic gradient descent by using only sign information of gradient estimation. Although it is possible to combine ZO-SignSGD with our proposed single query oracle for solving hard-label attack, their estimator will take sign of the whole vector and thus ignore the direction of $\mathbf{u}_q$, which leads to slower convergence in practice (please refer to Section 4.4 and Figure 5(b) for more details).

To the best of our knowledge, no previous analysis can be used to prove convergence of Algorithm 4. In the following, we show that Algorithm 4 can in fact converge and furthermore, with similar convergence rate compared with (LCC19) despite using a different gradient estimator.

Assumption 1. *Function $g(\theta)$ is L-smooth with a finite value of L.*

Assumption 2. *At any iteration step t, the gradient of the function g is upper bounded by* $\|\nabla g(\boldsymbol{\theta}_t)\|_2 \leq \sigma$.

Theorem 2. *Suppose that the conditions in the assumptions hold, and the distribution of gradient noise is unimodal and symmetric. Then, Sign-OPT attack with learning rate* $\eta_t = O(\frac{1}{Q\sqrt{dT}})$ *and* $\epsilon = O(\frac{1}{dT})$ *will give following bound on* $\mathbb{E}[\|\nabla g(\boldsymbol{\theta})\|_2]$*:*

$$\mathbb{E}[\|\nabla g(\boldsymbol{\theta})\|_2] = O(\frac{\sqrt{d}}{\sqrt{T}} + \frac{d}{\sqrt{Q}}).$$

The proof can be found in subsubsection 2.6.0.2. The main difference with the original analysis provided by (LCC19) is that they only only deal with sign of each element, while our analysis also takes the magnitudes of each element of $\boldsymbol{u}_q$ into account.

Other gradient estimations Note that the value $\text{sign}(g(\boldsymbol{\theta} + \epsilon\boldsymbol{u}) - g(\boldsymbol{\theta}))$ computed by our single query oracle is actually the sign of the directional derivative:

$$\text{sign}(\langle \nabla g(\boldsymbol{\theta}), \boldsymbol{u} \rangle) = \text{sign}(\lim_{\epsilon \to \infty} \frac{g(\boldsymbol{\theta} + \epsilon\boldsymbol{u}) - g(\boldsymbol{\theta})}{\epsilon}) = \text{sign}(g(\boldsymbol{\theta} + \epsilon\boldsymbol{u}) - g(\boldsymbol{\theta})) \text{ for a small } \epsilon.$$

Therefore, we can use this information to estimate the original gradient. The Sign-OPT approach in the previous section uses $\sum_q \text{sign}(\langle \nabla g(\boldsymbol{\theta}), \boldsymbol{u}_q \rangle)\boldsymbol{u}_q$ as an estimation of gradient. Let $y_q := \text{sign}(\langle \nabla g(\boldsymbol{\theta}), \boldsymbol{u}_q \rangle)$, a more accurate gradient estimation can be cast as the following constraint optimization problem:

$$\text{Find a vector } \boldsymbol{z} \text{ such that } \text{sign}(\langle \boldsymbol{z}, \boldsymbol{u}_q \rangle) = y_q \quad \forall q = 1, \ldots, Q.$$

Therefore, this is equivalent to a hard constraint SVM problem where each $\boldsymbol{u}_q$ is a training sample and y_q is the corresponding label. The gradient can then be recovered by solving the following quadratic programming problem:

$$\min_{\boldsymbol{z}} \ \boldsymbol{z}^T\boldsymbol{z} \quad \text{s.t.} \quad \boldsymbol{z}^T\boldsymbol{u}_q \geq y_q, \quad \forall q = 1, \ldots, Q. \tag{2.19}$$

By solving this problem, we can get a good estimation of the gradient. As explained earlier, each y_q can be determined with a single query. Therefore, we propose a variant of Sign-OPT, which is called SVM-OPT attack. The detailed procedure is shown in Algorithm 5. We will present an empirical comparison of our two algorithms in subsubsection 2.5.0.4.

Algorithm 5 SVM-OPT attack

Input: Hard-label model f, original image $\boldsymbol{x}_0$, initial $\boldsymbol{\theta}_0$
for $t = 1, 2, \ldots, T$ **do**
  Sample $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_Q$ from Gaussian or orthogonal basis
  Solve $\boldsymbol{z}$ defined by (2.19)
  Update $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \eta \boldsymbol{z}$
  Evaluate $g(\boldsymbol{\theta}_t)$ using search algorithm in (CLC19)

2.5 Experimental Results

We evaluate the SIGN-OPT algorithm for attacking black-box models in a hard-label setting on three different standard datasets - MNIST (LBB98), CIFAR-10 (KH09) and ImageNet-1000 (DDS09) and compare it with existing methods. For fair and easy comparison, we use the CNN networks provided by (CW17), which have also been used by other previous hard-label attacks as well. Specifically, for both MNIST and CIFAR-10, the model consists of nine layers in total - four convolutional layers, two max-pooling layers and two fully-connected layers. Further details about implementation, training and parameters are available on (CW17). As reported in (CW17) and (CLC19), we were able to achieve an accuracy of 99.5% on MNIST and 82.5% on CIFAR-10. We use the pretrained Resnet-50 (HZR16a) network provided by torchvision (MR10) for ImageNet-1000, which achieves a Top-1 accuracy of 76.15%.

In our experiments, we found that Sign-OPT and SVM-OPT perform quite similarly in terms of query efficiency. Hence we compare only Sign-OPT attack with previous approaches and provide a comparison between Sign-OPT and SVM-OPT in subsubsection 2.5.0.4. We compare the following attacks:

- **Sign-OPT attack** (black box): The approach presented in (CSC19).
- **Opt-based attack** (black box): The method proposed in (CLC19) where they use Randomized Gradient-Free method to optimize the same objective function.
- **Boundary attack** (black box): The method proposed in (BRB17). This is compared only in L_2 setting as it is designed for the same. We use the implementation provided

in Foolbox (`https://github.com/bethgelab/foolbox`).

- **Guessing Smart Attack** (black box): The method proposed in (BDL18). This attack enhances boundary attack by biasing sampling towards three priors. Note that one of the priors assumes access to a similar model as the target model and for a fair comparison we do not incorporate this bias in our experiments. We use the implementation provided at `https://github.com/ttbrunner/biased_boundary_attack`.

- **C&W attack** (white box): One of the most popular methods in the white-box setting proposed in (CW17). We use C&W L_2 norm attack as a baseline for the white-box attack performance.

For each attack, we randomly sample 100 examples from validation set and generate adversarial perturbations for them. For untargeted attack, we only consider examples that are correctly predicted by model and for targeted attack, we consider examples that are already not predicted as target label by the model. To compare different methods, we mainly use *median distortion* as the metric. Median distortion for x queries is the median adversarial perturbation of all examples achieved by a method using less than x queries. Since all the hard-label attack algorithms will start from an adversarial exmample and keep reduce the distortion, if we stop at any time they will always give an adversarial example and medium distortion will be the most suitable metric to compare their performance. Besides, we also show *success rate (SR)* for x queries for a given threshold (ϵ), which is the percentage of number of examples that have achieved an adversarial perturbation below ϵ with less than x queries. We evaluate success rate on different thresholds which depend on the dataset being used. For comparison of different algorithms in each setting, we chose the same set of examples across all attacks.

Implementation details: To optimize Algorithm 4, we estimate the step size η using the same line search procedure implemented in (CLC19). At the cost of a relatively small number of queries, this provides significant speedup in the optimization. Similar to (CLC19), $g(\theta)$ in

last step of Algorithm 4 is approximated via binary search. The initial θ_0 in Algorithm 4 is calculated by evaluating $g(\theta)$ on 100 random directions and taking the best one. We provide our implementation publicly[1].

2.5.0.1 Untargeted attack

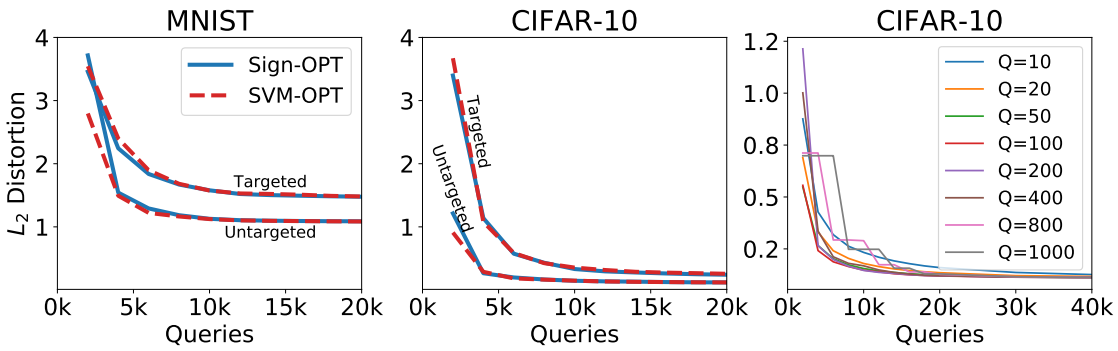


Figure 2.5: Median L_2 distortion vs Queries. First two: Comparison of Sign-OPT and SVM-OPT attack for MNIST and CIFAR-10. Third: Performance of Sign-OPT for different values of Q.

In this attack, the objective is to generate an adversary from an original image for which the prediction by model is different from that of original image. Figure 2.6 provides an elaborate comparison of different attacks for L_2 case for the three datasets. Sign-OPT attack consistently outperforms the current approaches in terms of queries. Not only is Sign-OPT more efficient in terms of queries, in most cases it converges to a lower distortion than what is possible by other hard-label attacks. Furthermore, we observe Sign-OPT converges to a solution comparable with C&W white-box attack (better on CIFAR-10, worse on MNIST, comparable on ImageNet). This is significant for a hard-label attack algorithm since we are given very limited information.

We highlight some of the comparisons of Boundary attack, OPT-based attack and Sign-OPT attack (L_2 norm-based) in Table 2.2. Particularly for ImageNet dataset on ResNet-50

[1]https://github.com/cmhcbb/attackbox

model, Sign-OPT attack reaches a median distortion below 3.0 in less than $30k$ queries while other attacks need more than $200k$ queries for the same.

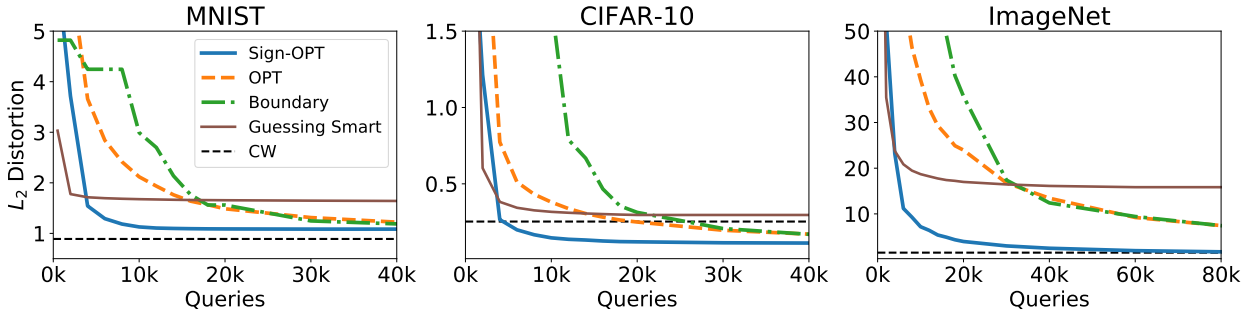


Figure 2.6: Untargeted attack: Median distortion vs Queries for different datasets.

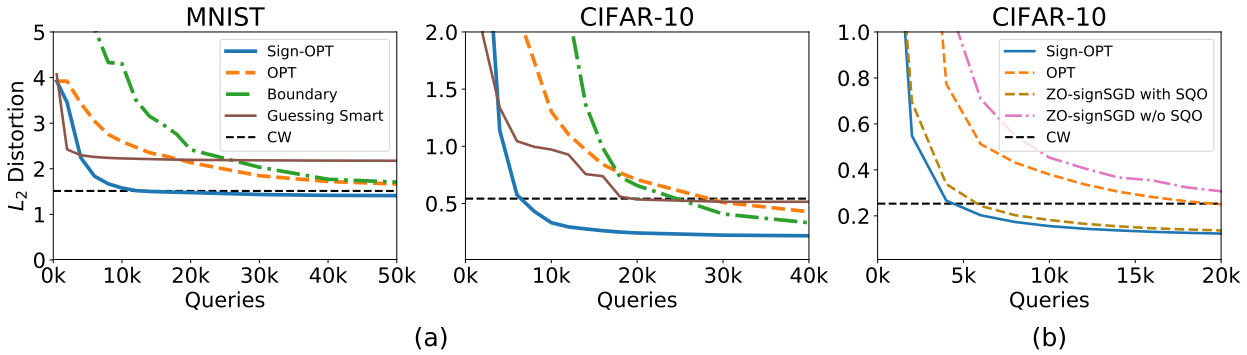


Figure 2.7: (a) Targeted Attack: Median distortion vs Queries of different attacks on MNIST and CIFAR-10. (b) Comparing Sign-OPT and ZO-SignSGD with and without single query oracle (SQO).

2.5.0.2 Targeted attack

In targeted attack, the goal is to generate an adversarial perturbation for an image so that the prediction of resulting image is the same as a specified target. For each example, we randomly specify the target label, keeping it consistent across different attacks. We calculate the initial θ_0 in Algorithm 4 using 100 samples in target label class from training dataset and this θ_0 is the same across different attacks. Figure 2.9 shows some examples of adversarial examples generated by Sign-OPT attack and the Opt-based attack. The first two rows show

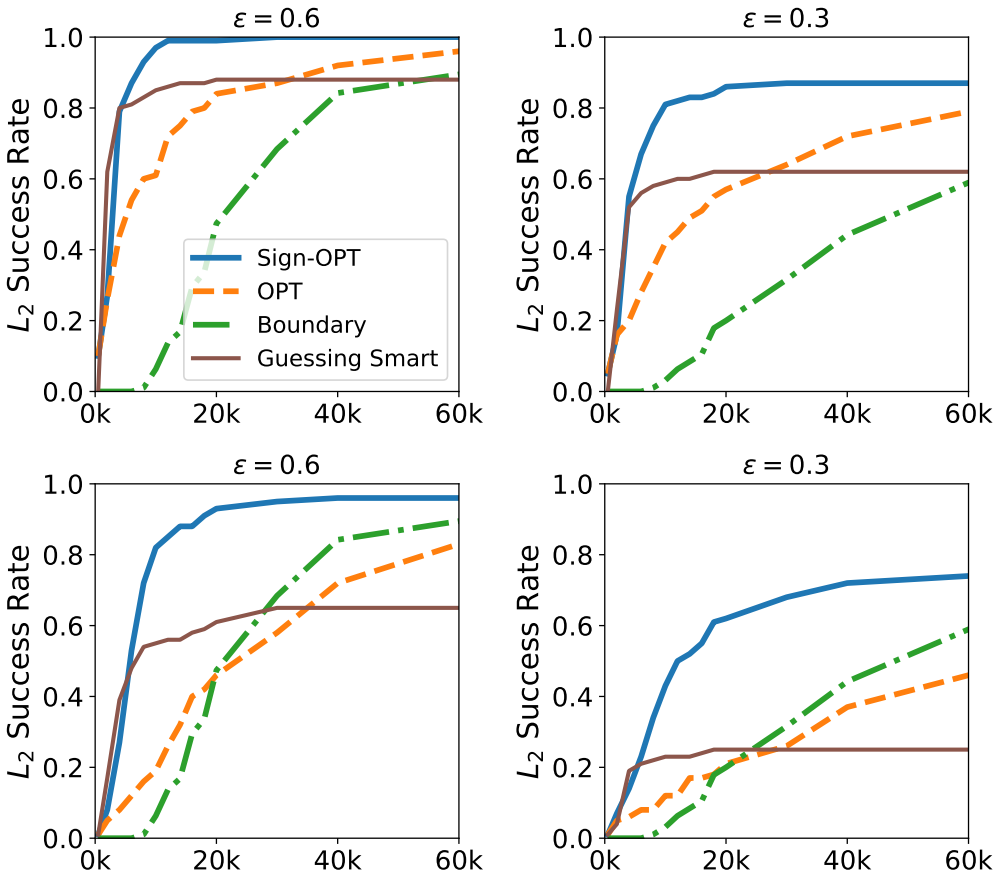


Figure 2.8: Success Rate vs Queries for CIFAR-10 (L_2 norm-based attack). First two and last two depict untargeted and targeted attacks respectively. Success rate threshold is at the top of each plot.

comparison of Sign-OPT and Opt attack respectively on an example from MNIST dataset. The figures show adversarial examples generated at almost same number of queries for both attacks. Sign-OPT method generates an L_2 adversarial perturbation of 0.94 in $\sim 6k$ queries for this particular example while Opt-based attack requires $\sim 35k$ for the same. Figure 2.7 displays a comparison among different attacks in targeted setting. In our experiments, average distortion achieved by white box attack C&W for MNIST dataset is 1.51, for which Sign-OPT requires $\sim 12k$ queries while others need $> 120k$ queries. We present a comparison of success rate of different attacks for CIFAR-10 dataset in Figure 2.8 for both targeted and untargeted cases.

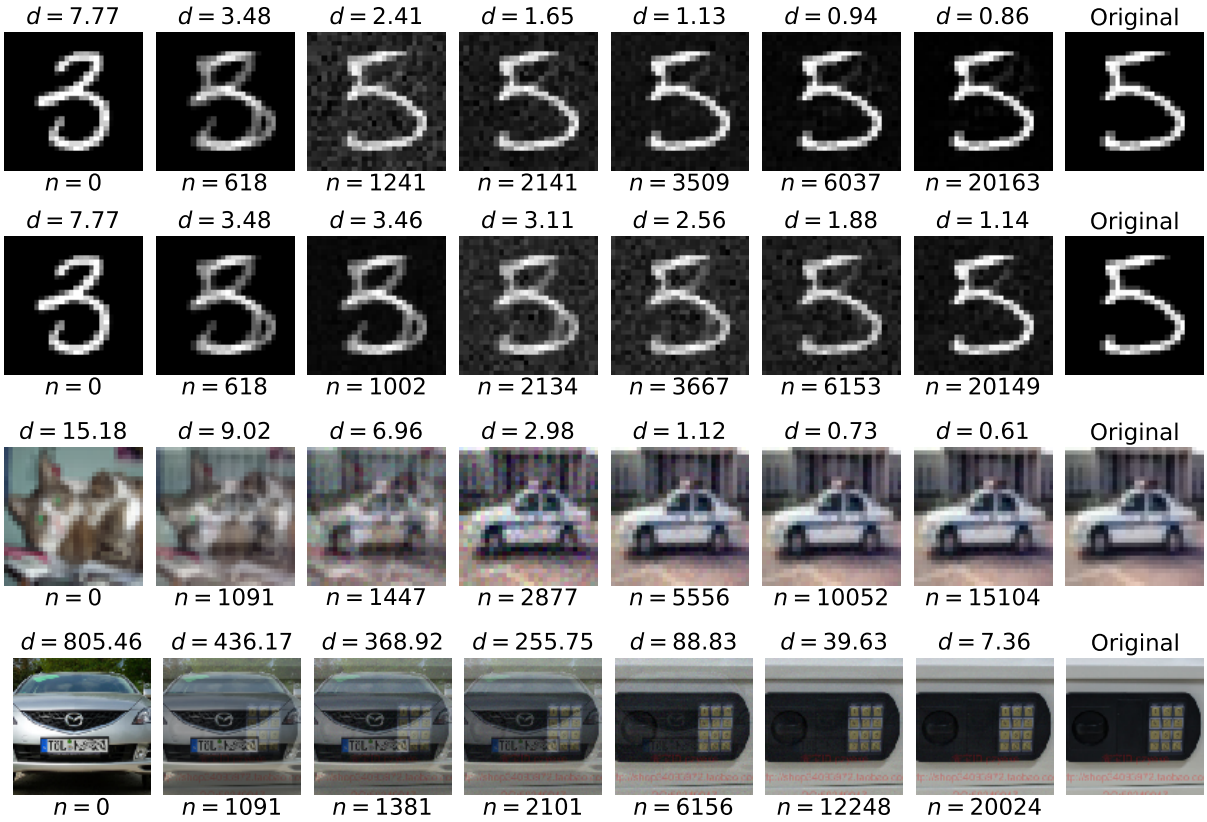


Figure 2.9: Example of Sign-OPT targeted attack. L_2 distortions and queries used are shown above and below the images. First two rows: Example comparison of Sign-OPT attack and OPT attack. Third and fourth rows: Examples of Sign-OPT attack on CIFAR-10 and ImageNet

2.5.0.3 The power of single query oracle

In this subsection, we conduct several experiments to prove the effectiveness of our proposed single query oracle in hard-label adversarial attack setting. ZO-SignSGD algorithm (LCC19) is proposed for soft-label black box attack and we extend it into hard-label setting. A straightforward way is simply applying ZO-SignSGD to solve the hard-label objective proposed in (CLC19), estimate the gradient using binary search as (CLC19) and take its sign. In Figure 5(b), we clearly observe that simply combining ZO-SignSGD and (CLC19) is not efficient. With the proposed single query sign oracle, we can also reduce the query count of this method, as demonstrated in Figure 5(b). This verifies the effectiveness of single query oracle, which can universally improve many different optimization methods in the hard-label attack setting. To be noted, there is still improvement on Sign-OPT over ZO-SignSGD with single query oracle because instead of directly taking the sign of gradient estimation, our algorithm utilizes the scale of random direction u as well. In other words, signSGD's gradient norm is always 1 while our gradient norm takes into account the magnitude of u. Therefore, our signOPT optimization algorithm is fundamentally different (LCC19) or any other proposed signSGD varieties. Our method can be viewed as a new zeroth order optimization algorithm that features fast convergence in signSGD.

Table 2.1: Results of (L_2-norm based) untargeted attack on gradient boosting decision tree.

	HIGGS		MNIST	
	Avg L_2	# queries	Avg L_2	# queries
Opt-attack	0.3458	4,229	0.6113	5,125
	0.2179	11,139	0.5576	11,858
	0.1704	29,598	0.5505	32,230

2.5.0.4 Comparison between Sign-OPT and SVM-OPT

In our experiments, we found that the performance in terms of queries of both these attacks is remarkably similar in all settings (both L_2/L_∞ & Targeted/Untargeted) and datasets. We

present a comparison for MNIST and CIFAR-10 (L_2 norm-based) for both targeted and untargeted attacks in Figure 2.5. We see that the median distortion achieved for a given number of queries is quite on part for both Sign-OPT and SVM-OPT.

Number of queries per gradient estimate: In Figure 2.5, we show the comparison of Sign-OPT attack with different values of Q. Our experiments suggest that Q does not have an impact on the convergence point reached by the algorithm. Although, small values of Q provide a noisy gradient estimate and hence delayed convergence to an adversarial perturbation. Large values of Q, on the other hand, require large amount of time per gradient estimate. After fine tuning on a small set of examples, we found that $Q = 200$ provides a good balance between the two. Hence, we set the value of $Q = 200$ for all our experiments in this section.

2.5.0.5 Attacking Gradient Boosting Decision Tree (GBDT)

To evaluate our method's ability to attack models with discrete decision functions, we conduct our untargeted attack on gradient booting decision tree (GBDT). In this experiment, we use two standard datasets: HIGGS (BSW14) for binary classification and MNIST (LBB98) for multi-class classification. We use popular LightGBM framework to train the GBDT models and use suggested parameters in `https://github.com/Koziev/MNIST_Boosting`. To be more specific, for MNIST model, it has 100 trees and the max number of leaves in each tree is 100. For Higgs model, it has 255 trees and the max number of leaves in each tree is 500. And we don't limit the max depth on both models. We could achieve 0.8457 AUC for HIGGS and 98.09% accuracy for MNIST. The results of untargeted attack on GBDT are given in Table 2.1.

As shown in Table 2.1, by using around 30K queries, we could get a small distortion on both datasets, which firstly uncovers the vulnerability of GBDT models. Tree-based methods are well-known for its good interpretability. And because of that, they are widely used in the industry. However, we show that even with good interpretability and a similar

Table 2.2: L_2 Untargeted attack - Comparison of average L_2 distortion achieved using a given number of queries for different attacks. SR stands for success rate.

	MNIST			CIFAR10			ImageNet (ResNet-50)		
	#Queries	Avg L_2	SR($\epsilon = 1.5$)	#Queries	Avg L_2	SR($\epsilon = 0.5$)	#Queries	Avg L_2	SR($\epsilon = 3.0$)
Boundary attack	4,000	4.24	1.0%	4,000	3.12	2.3%	4,000	209.63	0%
	8,000	4.24	1.0%	8,000	2.84	7.6%	30,000	17.40	16.6%
	14,000	2.13	16.3%	12,000	0.78	29.2%	160,000	4.62	41.6%
OPT attack	4,000	3.65	3.0%	4,000	0.77	37.0%	4,000	83.85	2.0%
	8,000	2.41	18.0%	8,000	0.43	53.0%	30,000	16.77	14.0%
	14,000	1.76	36.0%	12,000	0.33	61.0%	160,000	4.27	34.0%
Guessing Smart	4,000	1.74	41.0%	4,000	0.29	75.0%	4,000	16.69	12.0%
	8,000	1.69	42.0%	8,000	0.25	80.0%	30,000	13.27	12.0%
	14,000	1.68	43.0%	12,000	0.24	80.0%	160,000	12.88	12.0%
Sign-OPT attack	4,000	1.54	46.0%	4,000	0.26	73.0%	4,000	23.19	8.0%
	8,000	1.18	84.0%	8,000	0.16	90.0%	30,000	2.99	50.0%
	14,000	1.09	94.0%	12,000	0.13	95.0%	160,000	1.21	90.0%
C&W (white-box)	-	0.88	99.0%	-	0.25	85.0%	-	1.51	80.0%

prediction accuracy with convolution neural network, the GBDT models are vulnerable under our Opt-attack. This result raises a question about tree-based models' robustness, which will be an interesting direction in the future.

2.6 Proofs

2.6.0.1 Convergence guarantee for OPT-attack

If $g(\boldsymbol{\theta})$ can be computed exactly, it has been proved in (NS17) that RGF in Algorithm 3 requires at most $O(\frac{d}{\delta^2})$ iterations to converge to a point with $\|\nabla g(\boldsymbol{\theta})\|^2 \leq \delta^2$. However, in our algorithm the function value $g(\boldsymbol{\theta})$ cannot be computed exactly; instead, we compute it up to ϵ-precision, and this precision can be controlled by binary threshold in Algorithm 2. We thus extend the proof in (NS17) to include the case of approximate function value evaluation, as described in the following theorem.

Theorem 3. *In Algorithm 3, suppose g has Lipschitz-continuous gradient with constant $L_1(g)$ and g^* (optimal value) is finite. If the error of function value evaluation is controlled by $\epsilon = O(\beta\delta^2)$ and $\beta \leq \frac{\delta}{dL_1(g)}$, then in order to obtain $\frac{1}{N+1}\sum_{k=0}^{N} E_{\mathcal{U}_k}(\|\nabla g(\boldsymbol{\theta}_k)\|^2) \leq \delta^2$, the upper bound of total number of iterations is $O(\frac{d}{\delta^2})$.*

Note that the binary search procedure could obtain the desired function value precision in $O(\log \delta)$ steps. By using the same idea with Theorem 1 and following the proof in (NS17), we could also achieve $O(\frac{d^2}{\delta^3})$ complexity when $g(\boldsymbol{\theta})$ is non-smooth but Lipschitz continuous.

Because there is a stopping criterion in Algorithm 2, we couldn't achieve the exact $g(\boldsymbol{\theta})$. Instead, we could get $\tilde{g}$ with ϵ error, i.e., $g(\boldsymbol{\theta}) - \epsilon \leq \tilde{g}(\boldsymbol{\theta}) \leq g(\boldsymbol{\theta}) + \epsilon$. Also, we define $\hat{\boldsymbol{g}}(\boldsymbol{\theta}) = \frac{\tilde{g}(\boldsymbol{\theta}+\beta \boldsymbol{u}) - \tilde{g}(\boldsymbol{\theta})}{\beta} \cdot \boldsymbol{u}$ to be the noise gradient estimator.

Following (Nes11), we define the Guassian smoothing approximation over $g(\theta)$, i.e,

$$g_\beta(\theta) = \frac{1}{\kappa} \int_E g(\theta + \beta u) e^{-\frac{1}{2}||u||^2} du. \tag{2.20}$$

Also, we have the upper bounds for the moments $M_p = \frac{1}{\kappa} \int_E ||u||^p e^{-\frac{1}{2}||u||^2} du$ from (Nes11) Lemma 1.

For $p \in [0, 2]$, we have

$$M_p \leq d^{p/2}. \tag{2.21}$$

If $p \geq 2$, we have two-sided bounds

$$n^{p/2} \leq M_p \leq (p+n)^{p/2}. \tag{2.22}$$

Proof of Theorem 1 Suppose f has a lipschitz-continuous gradient with constant $L_1(g)$, then

$$|g(y) - g(x) - \langle \nabla g(x), y - x \rangle| \leq \frac{1}{2} L_1(g) ||x - y||^2 \tag{2.23}$$

We could bound $E_u(||\hat{g}(\boldsymbol{\theta})||^2)$ as follows,

Since

$$\begin{aligned}(\tilde{g}(\boldsymbol{\theta} + \beta u) - \tilde{g}(\boldsymbol{\theta}))^2 &= [\tilde{g}(\boldsymbol{\theta} + \beta u) - \tilde{g}(\boldsymbol{\theta}) - \beta \langle \nabla g(\boldsymbol{\theta}), u \rangle + \beta \langle \nabla g(\boldsymbol{\theta}), u \rangle]^2 \\ &\leq 2(g(\boldsymbol{\theta} + \beta u) - g(\boldsymbol{\theta}) + \epsilon_{\boldsymbol{\theta}+\beta u} - \epsilon_{\boldsymbol{\theta}} - \beta \langle \nabla g(\boldsymbol{\theta}), u \rangle)^2 + 2\beta^2 \langle \nabla g(\boldsymbol{\theta}), u \rangle^2 \end{aligned} \tag{2.24}$$

Because $|\epsilon_{\boldsymbol{\theta}+\beta u} - \epsilon_{\boldsymbol{\theta}}| \leq 2\epsilon$,

$$[\tilde{g}(\boldsymbol{\theta} + \beta u) - \tilde{g}(\boldsymbol{\theta})]^2 \leq 2(\frac{\beta^2}{2} L_1(g) ||u||^2)^2 + 4\beta^2 L_1(g) ||u||^2 \epsilon + 8\epsilon^2 + 2\beta^2 \langle \nabla g(\boldsymbol{\theta}), u \rangle^2 \tag{2.25}$$

Take expectation over u, and with Theorem 3 in (Nes11), which is $E_u(||g'(\boldsymbol{\theta}, u) \cdot u||^2) \leq$

$(d+4)||\nabla g(\boldsymbol{\theta})||^2$

$$\begin{aligned} E_u(||\hat{g}(\boldsymbol{\theta})||^2) &\leq \frac{\beta^2}{2}L_1^2(g)E_u(||u||^6) + 2E_u(||g'(\boldsymbol{\theta},u)\cdot u||^2) + 4L_1(g)\epsilon E_u(||u||^4) + 8\frac{\epsilon^2}{\beta^2}E_u(||u||^2) \\ &\leq \frac{\beta^2}{2}L_1^2(g)(d+6)^3 + 2(d+4)||\nabla g(\boldsymbol{\theta})||^2 + 4\epsilon L_1(g)(d+4)^2 + 8\frac{\epsilon^2}{\beta^2}d \end{aligned} \tag{2.26}$$

With $\epsilon = O(\delta^2\beta)$, we could bound $E_u(||\tilde{g}(\boldsymbol{\theta})||^2)$

$$E_u(||\hat{g}(\boldsymbol{\theta})||^2) \leq \frac{\beta^2}{2}L_1^2(g)(d+6)^3 + 2(d+4)||\nabla g(\boldsymbol{\theta})||^2 + 4\beta L_1(g)(d+4)^2\delta^2 + 8d\delta^4 \tag{2.27}$$

And with

$$||\nabla g(\boldsymbol{\theta})||^2 \leq 2||\nabla g_\beta(\boldsymbol{\theta})||^2 + \frac{\beta^2}{2}L_1^2(g)(d+4)^2 \tag{2.28}$$

Which is proved in (Nes11) Lemma 4.

Therefore, since $(n+6)^3 + 2(n+4)^3 \leq 3(n+5)^3$, we could get

$$\begin{aligned} E_u(||\hat{g}(\boldsymbol{\theta})||^2) &\leq \frac{\beta^2}{2}L_1^2(g)(d+6)^3 + 2(d+4)||\nabla g(\boldsymbol{\theta})||^2 + 2(d+4)||\nabla g(\boldsymbol{\theta})||^2 \\ &\quad + 4\beta L_1(g)(d+4)^2\delta^2 + 8d\delta^4 \\ &\leq \frac{\beta^2}{2}L_1^2(g)(d+6)^3 + 2(d+4)(2||\nabla g_\beta(\boldsymbol{\theta})||^2 + \frac{\beta^2}{2}L_1^2(g)(d+4)^2) \\ &\quad + 4\beta L_1(g)(d+4)^2\delta^2 + 8d\delta^4 \\ &\leq 4(d+4)||\nabla g_\beta(x)||^2 + \frac{3\beta^2}{2}L_1^2(g)(d+5)^3 + 4\beta L_1(g)(d+4)^2\delta^2 + 8d\delta^4 \end{aligned} \tag{2.29}$$

Therefore, since $g_\beta(\boldsymbol{\theta})$ has Lipshcitz-continuous gradient:

$$|g_\beta(\boldsymbol{\theta}_{k+1}) - g_\beta(\boldsymbol{\theta}_k) + \alpha\langle\nabla g_\beta(\boldsymbol{\theta}_k), \hat{g}_\beta(\boldsymbol{\theta}_k)\rangle| \leq \frac{1}{2}\alpha^2 L_1(g_\beta)||\hat{g}_\beta(\boldsymbol{\theta}_k)||^2 \tag{2.30}$$

So that

$$g_\beta(\boldsymbol{\theta}_{k+1}) \leq g_\beta(\boldsymbol{\theta}_k) - \alpha\langle\nabla g_\beta(\boldsymbol{\theta}_k), \hat{g}_\beta(\boldsymbol{\theta}_k)\rangle + \frac{1}{2}\alpha^2 L_1(g_\beta)||\hat{g}_\beta(\boldsymbol{\theta}_k)||^2 \tag{2.31}$$

Since

$$\begin{aligned} E_u(\hat{g}(\boldsymbol{\theta}_k)) &= \frac{1}{\kappa}\int_E \frac{g(\boldsymbol{\theta}+\beta u) - g(\boldsymbol{\theta}) + \epsilon_{\boldsymbol{\theta}+\beta u} - \epsilon_{\boldsymbol{\theta}}}{\beta} u e^{-\frac{1}{2}||u||^2} du \\ &= \nabla g_\beta(\boldsymbol{\theta}_k) + \frac{1}{\kappa}\int_E \frac{\epsilon_{\boldsymbol{\theta}+\beta u} - \epsilon_{\boldsymbol{\theta}}}{\beta} u e^{-\frac{1}{2}||u||^2} du \\ &\leq \nabla g_\beta(\boldsymbol{\theta}_k) + \frac{2\epsilon}{\beta}n^{1/2}\cdot\mathbb{1} \end{aligned} \tag{2.32}$$

where $\mathbb{1}$ is a all-one vector. Taking the expectation in u_k, we obtain

$$\begin{aligned} E_{u_k}(g_\beta(\boldsymbol{\theta}_{k+1})) &\leq g_\beta(\boldsymbol{\theta}_k) - \alpha_k||\nabla g_\beta(\boldsymbol{\theta}_k)||^2 + \alpha_k\langle\nabla g_\beta(\boldsymbol{\theta}_k), \frac{2\epsilon}{\beta}d^{1/2}\cdot\mathbb{1}\rangle \\ &\quad + \frac{1}{2}\alpha_k^2 L_1(g_\beta)E_{u_k}||\hat{g}_\beta(\boldsymbol{\theta}_k)||^2 \\ E_{u_k}(g_\beta(\boldsymbol{\theta}_{k+1})) &\leq g_\beta(\boldsymbol{\theta}_k) - \alpha_k||\nabla g_\beta(\boldsymbol{\theta}_k)||^2 + \alpha_k\frac{2\epsilon}{\beta}n^{1/2}||\nabla g_\beta(\boldsymbol{\theta}_k)|| \\ &\quad + \frac{1}{2}\alpha_k^2 L_1(g)(4(d+4)||\nabla g_\beta(\boldsymbol{\theta}_k)||^2 + \frac{3\beta^2}{2}L_1^2(g)(d+5)^3 \\ &\quad + 4\beta L_1(g)(d+4)^2\delta^2 + 8d\delta^4) \end{aligned} \tag{2.33}$$

Choosing $\alpha_k = \hat{\alpha} = \frac{1}{4(d+4)L_1(g)}$, we obtain

$$\begin{aligned} E_{u_k}(g_\beta(\boldsymbol{\theta}_k+1)) \leq g_\beta(\boldsymbol{\theta}_k) - \frac{1}{2}\hat{\alpha}||\nabla g_\beta(\boldsymbol{\theta}_k)||^2 + \hat{\alpha}\frac{2\epsilon}{\beta}d^{1/2}||\nabla g_\beta(\boldsymbol{\theta}_k)|| + \frac{3\beta^2}{64}L_1(g)\frac{(d+5)^3}{(d+4)^2} \\ + \frac{\beta}{8}\delta^2 + \frac{d}{4(d+4)^2L_1(g)}\delta^4 \end{aligned} \tag{2.34}$$

Since $(d+5)^3 \leq (d+8)(d+4)^2$, taking expectation over $\mathcal{U}_k$, where $\mathcal{U}_k = \{u_1, u_2, \ldots, u_k\}$, we get

$$\begin{aligned} \phi_{k+1} \leq &\phi_k - \frac{1}{2}\hat{\alpha}E_{\mathcal{U}_k}(||\nabla g_\beta(\boldsymbol{\theta}_k)||^2) + \frac{3\beta^2(d+8)}{64}L_1(g) + \frac{\beta}{8}\delta^2 + \frac{d}{4(d+4)^2L_1(g)}\delta^4 \\ &+ \hat{\alpha}d^{1/2}E_{\mathcal{U}_k}(||\nabla g_\beta(\boldsymbol{\theta}_k)||)\delta^2 \end{aligned} \tag{2.35}$$

Where $\phi_k = E_{\mathcal{U}_{k-1}(g(\boldsymbol{\theta}_k))}, k \geq 1$ and $\phi_0 = g(\boldsymbol{\theta}_0)$.

Assuming $g(x) \geq g^*$, summing over k and divided by N+1, we get

$$\begin{aligned} \frac{1}{N+1}\sum_{k=0}^{N}E_{\mathcal{U}_k}(||\nabla g_\beta(\boldsymbol{\theta}_k)||^2) \leq 8(d+4)L_1(g)[\frac{g(x_0)-g^*}{N+1} + \frac{3\beta^2(d+8)}{16}L_1(g) + \frac{\beta}{8}\delta^2 \\ + \frac{d}{4(d+4)^2L_1(g)}\delta^4 + \frac{1}{N+1}\sum_{k=0}^{N}E_{\mathcal{U}_k}(||\nabla g_\beta(\boldsymbol{\theta}_k)||)\delta^2] \end{aligned} \tag{2.36}$$

Clearly, $\frac{1}{N+1}\sum_{k=0}^{N}E_{\mathcal{U}_k}(||\nabla g_\beta(\boldsymbol{\theta}_k)||) \leq \delta^2$.

Since $\vartheta_k^2 = E_{\mathcal{U}_k}(||\nabla g(\boldsymbol{\theta}_k)||^2) \leq 2E_{\mathcal{U}_k}(||\nabla g_\beta(\boldsymbol{\theta}_k)||^2) + \frac{\beta^2(d+4)^2}{2}L_1^2(g)$, ϑ_k^2 is in the same order of $E_{\mathcal{U}_k}(||\nabla g_\beta(\boldsymbol{\theta}_k)||^2)$. In order to get $\frac{1}{N+1}\sum_{k=0}^{N}\vartheta_k^2 \leq \delta^2$, we need to choose $\beta \leq \frac{\delta}{dL_1(g)}$, then N is bounded by $O(\frac{d}{\delta^2})$

2.6.0.2 Convergence guarantee for Sign-OPT attack

Define following notations:

$$\hat{\nabla} g(\boldsymbol{\theta}_t; \boldsymbol{u}_q) \coloneqq \text{sign}(g(\boldsymbol{\theta}_t + \epsilon \boldsymbol{u}_q) - g(\boldsymbol{\theta}_t))\boldsymbol{u}_q$$
$$\dot{\nabla} g(\boldsymbol{\theta}_t; \boldsymbol{u}_q) \coloneqq \frac{1}{\epsilon}(g(\boldsymbol{\theta}_t + \epsilon \boldsymbol{u}_q) - g(\boldsymbol{\theta}_t))\boldsymbol{u}_q$$
$$\bar{\nabla} g(\boldsymbol{\theta}_t; \boldsymbol{u}_q) \coloneqq \text{sign}(\frac{1}{\epsilon}(g(\boldsymbol{\theta}_t + \epsilon \boldsymbol{u}_q) - g(\boldsymbol{\theta}_t))\boldsymbol{u}_q)$$

Thus we could write the corresponding estimate of gradients as follow:

$$\hat{\boldsymbol{g}}_t = \frac{1}{Q}\sum_{q=1}^{Q} \text{sign}(g(\boldsymbol{\theta}_t + \epsilon \mathbf{u}_q) - g(\boldsymbol{\theta}_t))\boldsymbol{u}_q = \frac{1}{Q}\sum_{q=1}^{Q} \hat{\nabla} g(\boldsymbol{\theta}_t; \boldsymbol{u}_q)$$
$$\dot{\boldsymbol{g}}_t = \frac{1}{Q}\sum_{q=1}^{Q} \frac{1}{\epsilon}(g(\boldsymbol{\theta}_t + \epsilon \boldsymbol{u}_q) - g(\boldsymbol{\theta}_t))\boldsymbol{u}_q = \frac{1}{Q}\sum_{q=1}^{Q} \dot{\nabla} g(\boldsymbol{\theta}_t; \boldsymbol{u}_q)$$
$$\bar{\boldsymbol{g}}_t = \frac{1}{Q}\sum_{q=1}^{Q} \text{sign}(\frac{1}{\epsilon}(g(\boldsymbol{\theta}_t + \epsilon \boldsymbol{u}_q) - g(\boldsymbol{\theta}_t))\boldsymbol{u}_q) = \frac{1}{Q}\sum_{q=1}^{Q} \bar{\nabla} g(\boldsymbol{\theta}_t; \boldsymbol{u}_q)$$

Clearly, we have $\bar{\nabla} g(\boldsymbol{\theta}_t; \boldsymbol{u}_q) = \text{sign}(\dot{\nabla} g(\boldsymbol{\theta}_t; \boldsymbol{u}_q))$ and we could relate $\bar{\nabla} g(\boldsymbol{\theta}_t; \boldsymbol{u}_q)$ and $\hat{\nabla} g(\boldsymbol{\theta}_t; \boldsymbol{u}_q)$ by writing $\hat{\nabla} g(\boldsymbol{\theta}_t; \boldsymbol{u}_q) = G_q \odot \bar{\nabla} g(\boldsymbol{\theta}_t; \boldsymbol{u}_q)$ where where $G_q \in \mathbb{R}^d$ is absolute value of vector $\boldsymbol{u}_q$ (i.e. $G_q = (|\boldsymbol{u}_{q,1}|, |\boldsymbol{u}_{q,2}|, \cdots, |\boldsymbol{u}_{q,d}|)^T$).

Note that Zeroth-order gradient estimate $\dot{\nabla} g(\boldsymbol{\theta}_t; \boldsymbol{u}_q)$ is a biased approximation to the true gradient of g. Instead, it becomes unbiased to the gradient of the randomized smoothing function $g_\epsilon(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{u}}[g(\boldsymbol{\theta} + \epsilon \boldsymbol{u})]$ (DBW12).

Our analysis is based on the following two assumptions:

Assumption 1 function g is L-smooth with a finite value of L.

Assumption 2 At any iteration step t, the gradient of the function g is upper bounded by $\|\nabla g(\boldsymbol{\theta}_t)\|_2 \leq \sigma$.

To prove the convergence of proposed method, we need the information on variance of the update $\dot{\nabla} g(\boldsymbol{\theta}_t; \boldsymbol{u}_q)$. Here, we introduce a lemma from previous works.

Lemma 1 The variance of Zeroth-Order gradient estimate $\dot{\nabla} g(\boldsymbol{\theta}_t; \boldsymbol{u}_q)$ is upper bounded by

$$\mathbb{E}\big[\|\dot{\nabla} g(\boldsymbol{\theta}_t; \boldsymbol{u}_q) - \nabla g_\epsilon(\boldsymbol{\theta}_t)\|_2^2\big] \leq \frac{4(Q+1)}{Q}\sigma^2 + \frac{2}{Q}C(d,\epsilon),$$

where $C(d,\epsilon) \coloneqq 2d\sigma^2 + \epsilon^2 L^2 d^2/2$

Proof of Lemma 1 This lemma could be proved by using proposition 2 in (LCC19) with b = 1 and q = Q. When b = 1 there is no difference between with/without replacement, and we opt for with replacement case to obtain above bound. □

By talking Q = 1, we know that $\mathbb{E}\big[\|\dot{\nabla} g(\boldsymbol{\theta}_t; \boldsymbol{u}_q) - \nabla g_\epsilon(\boldsymbol{\theta}_t)\|_2^2\big]$ is upper bounded. And by Jensen's inequality, we also know that the

$$\mathbb{E}\big[|(\dot{\nabla} g(\boldsymbol{\theta}_t; \boldsymbol{u}_q) - \nabla g_\epsilon(\boldsymbol{\theta}_t))_l|\big] \leq \sqrt{\mathbb{E}\big[((\dot{\nabla} g(\boldsymbol{\theta}_t; \boldsymbol{u}_q) - \nabla g_\epsilon(\boldsymbol{\theta}_t))_l^2\big]} \coloneqq \delta_l, \tag{2.37}$$

where δ_l denotes the upper bound of lth coordinate of $\mathbb{E}\big[|\dot{\nabla} g(\boldsymbol{\theta}_t; \boldsymbol{u}_q) - \nabla g_\epsilon(\boldsymbol{\theta}_t)|\big]$, and δ_l is finite since $\mathbb{E}\big[\|\dot{\nabla} g(\boldsymbol{\theta}_t; \boldsymbol{u}_q) - \nabla g_\epsilon(\boldsymbol{\theta}_t)\|_2^2\big]$ is upper bounded.

Next, we want to show the Prob[sign$((\bar{\boldsymbol{g}}_t)_l) \neq$ sign$((\nabla g_\epsilon(\boldsymbol{\theta}_t))_l)$] by following lemma.

Lemma 2 $|(\nabla g_\epsilon(\boldsymbol{\theta}_t))_l|$Prob[sign$((\bar{\boldsymbol{g}}_t)_l) \neq$ sign$((\nabla g_\epsilon(\boldsymbol{\theta}_t))_l)$] $\leq \frac{\delta_l}{\sqrt{Q}}$

Proof of Lemma 2 We first relax Prob[sign$((\dot{\nabla} g(\boldsymbol{\theta}_t; \boldsymbol{u}_q))_l) \neq$ sign$(\nabla g_\epsilon(\boldsymbol{\theta}_t))_l$] by Markov inequality:

$$\begin{aligned}
\text{Prob}[\text{sign}((\dot{\nabla} g(\boldsymbol{\theta}_t; \boldsymbol{u}_q))_l) \neq \text{sign}((\nabla g_\epsilon(\boldsymbol{\theta}_t))_l)] &\leq \text{Prob}[|\dot{\nabla} g(\boldsymbol{\theta}_t; \boldsymbol{u}_q)_l)| \geq |\nabla g_\epsilon(\boldsymbol{\theta}_t)_l|] \\
&\leq \frac{\mathbb{E}\big[|(\dot{\nabla} g(\boldsymbol{\theta}_t; \boldsymbol{u}_q) - \nabla g_\epsilon(\boldsymbol{\theta}_t))_l|\big]}{|\nabla g_\epsilon(\boldsymbol{\theta}_t)_l|} \\
&\leq \frac{\delta_l}{|\nabla g_\epsilon(\boldsymbol{\theta}_t)_l|},
\end{aligned}$$

where the last inequality comes from eq (2.37). Recall that $(\dot{\nabla} g(\boldsymbol{\theta}_t; \boldsymbol{u}_q))_l)$ is an unbiased estimation to $(\nabla g_\epsilon(\boldsymbol{\theta}_t))_l$. Under the assumption that the noise distribution is unimodal and symmetric, from (BWA18) Lemma D1, we will have

$$\text{Prob}[\text{sign}((\dot{\nabla} g(\boldsymbol{\theta}_t; \boldsymbol{u}_q))_l) \neq \text{sign}(\nabla g_\epsilon(\boldsymbol{\theta}_t))_l] \coloneqq M \leq \begin{cases} \frac{2}{9}\frac{1}{S^2}, & \text{S } \geq \frac{2}{\sqrt{3}} \\ \frac{1}{2} - \frac{S}{2\sqrt{3}}, & otherwise \end{cases} < \frac{1}{2},$$

where $S \coloneqq |\nabla g_\epsilon(\boldsymbol{\theta}_t)_l|/\delta_l$.

Note that this probability bound applies uniformly to all $q \in Q$ regardless of the magnitude $|(\boldsymbol{u}_q)_l|$. That is,

$$\text{Prob}[\text{sign}(\sum_{q=1}^{Q} |(\boldsymbol{u}_q)_l| \text{sign}((\dot{\nabla} g(\boldsymbol{\theta}_t; \boldsymbol{u}_q))_l) \neq \text{sign}(\nabla g_\epsilon(\boldsymbol{\theta}_t))_l] =$$

$$\text{Prob}[\text{sign}((\sum_{q=1}^{Q} \text{sign}(\dot{\nabla} g(\boldsymbol{\theta}_t; \boldsymbol{u}_q))_l) \neq \text{sign}(\nabla g_\epsilon(\boldsymbol{\theta}_t))_l]. \tag{2.38}$$

This is true as when all $|(\boldsymbol{u}_q)_l| = 1$, $\text{Prob}[\text{sign}((\sum_{q=1}^{Q} \text{sign}(\dot{\nabla} g(\boldsymbol{\theta}_t; \boldsymbol{u}_q))_l) \neq \text{sign}(\nabla g_\epsilon(\boldsymbol{\theta}_t))_l]$ is equivalent to majority voting of each estimate q yielding correct sign. This is the same as sum of Q bernoulli trials (i.e. binomial distribution) with error rate M. And since error probability M is independent of sampling of $|(\boldsymbol{u}_q)_l|$, calculating
$\text{Prob}[\text{sign}(\sum_{q=1}^{Q} |(\boldsymbol{u}_q)_l| \text{sign}((\dot{\nabla} g(\boldsymbol{\theta}_t; \boldsymbol{u}_q))_l) \neq \text{sign}(\nabla g_\epsilon(\boldsymbol{\theta}_t))_l]$ could be thought as taking Q bernoulli experiments and then independently draw a weight from unit length for each of Q experiment. Since the weight is uniform, we will have expectation of weights on correct counts and incorrect counts are the same and equal to 1/2. Therefore, the probability of $\text{Prob}[\text{sign}(\sum_{q=1}^{Q} |(\boldsymbol{u}_q)_l| \text{sign}((\dot{\nabla} g(\boldsymbol{\theta}_t; \boldsymbol{u}_q))_l) \neq \text{sign}(\nabla g_\epsilon(\boldsymbol{\theta}_t))_l]$ is still the same as original non-weighted binomial distribution. Notice that by our notation, we will have $\text{sign}(\dot{\nabla} g(\boldsymbol{\theta}_t; \boldsymbol{u}_q)_l) = \bar{\nabla} g(\boldsymbol{\theta}_t; \boldsymbol{u}_q)_l$ thus $\frac{1}{Q} \sum_{q=1}^{Q} \text{sign}(\dot{\nabla} g(\boldsymbol{\theta}_t; \boldsymbol{u}_q))_l = (\bar{\boldsymbol{g}}_t)_l$. Let Z counts the number of estimates $\dot{\nabla} g(\boldsymbol{\theta}_t; \boldsymbol{u}_q)_l$ yielding correct sign of $\nabla g_\epsilon(\boldsymbol{\theta}_t)_l$. Probability in eq (2.38) could be written as:

$$\text{Prob}[\text{sign}(\text{sign}((\bar{\boldsymbol{g}}_t)_l) \neq \text{sign}(\nabla g_\epsilon(\boldsymbol{\theta}_t))_l] = P[Z \leq \frac{Q}{2}].$$

Following the derivation of theorem 2b in (BWA18), we could get

$$P[Z \leq \frac{Q}{2}] \leq \frac{1}{\sqrt{Q} S}$$

$$\Rightarrow |(\nabla g_\epsilon(\boldsymbol{\theta}_t))_l| \text{Prob}[\text{sign}((\bar{\boldsymbol{g}}_t)_l) \neq \text{sign}((\nabla g_\epsilon(\boldsymbol{\theta}_t))_l)] \leq \frac{\delta_l}{\sqrt{Q}} \tag{2.39}$$

□

We also need few more lemmas on properties of function g.

Lemma 3 $g_\epsilon(\boldsymbol{\theta}_1) - g_\epsilon(\boldsymbol{\theta}_T) \leq g_\epsilon(\boldsymbol{\theta}_1) - g^* + \epsilon^2 L$

Proof of Lemma 3 The proof can be found in (LKC18) Lemma C. □

Lemma 4 $\mathbb{E}[\|\nabla g(\boldsymbol{\theta})\|_2] \leq \sqrt{2}\mathbb{E}[\|\nabla g_\epsilon(\boldsymbol{\theta})\|_2] + \frac{\epsilon L d}{\sqrt{2}}$, where $g^* = \min_{\boldsymbol{\theta}} g(\boldsymbol{\theta})$.

Proof of Lemma 4 The proof can be found in (LCC19). □

Theorem 1 Suppose that the conditions in the assumptions hold, and the distribution of gradient noise is unimodal and symmetric. Then, Sign-OPT attack with learning rate $\eta_t = O(\frac{1}{Q\sqrt{dT}})$ and $\epsilon = O(\frac{1}{dT})$ will give following bound on $\mathbb{E}[\|\nabla g(\boldsymbol{\theta})\|_2]$

$$\mathbb{E}[\|\nabla g(\boldsymbol{\theta})\|_2] = O(\frac{\sqrt{d}}{\sqrt{T}} + \frac{d}{\sqrt{Q}})$$

Proof of Theorem 1 From L-smoothness assumption we could have

$$\begin{aligned}
g_\epsilon(\boldsymbol{\theta}_{t+1}) &\leq g_\epsilon(\boldsymbol{\theta}_t) + \langle \nabla g_\epsilon(\boldsymbol{\theta}_t), \boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t \rangle + \frac{L}{2}\|\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t\|_2^2 \\
&= g_\epsilon(\boldsymbol{\theta}_t) - \eta_k \langle \nabla g_\epsilon(\boldsymbol{\theta}_t), \hat{\boldsymbol{g}}_t \rangle + \frac{L}{2}\eta_t^2 \|\hat{\boldsymbol{g}}_t\|_2^2 \\
&= g_\epsilon(\boldsymbol{\theta}_t) - \eta_t \odot \bar{G}_t \|\nabla g_\epsilon(\boldsymbol{\theta}_t)\|_1 + \frac{dL}{2}\eta_t^2 \odot {\bar{G}_t}^2 \\
&+ 2\eta_t \odot \bar{G}_t \sum_{l=1}^{d} |(\nabla g_\epsilon(\boldsymbol{\theta}_t))_l| \text{Prob}[\text{sign}((\bar{\boldsymbol{g}}_t)_l) \neq \text{sign}((\nabla g_\epsilon(\boldsymbol{\theta}_t))_l)],
\end{aligned}$$

where $\bar{G}_t$ is defined as $(\bar{G}_t)_l = \sum_{q=1}^{Q} (G_q)_l \bar{\nabla} g(\boldsymbol{\theta}_t; \boldsymbol{u}_q)_l = \sum_{q=1}^{Q} |(\boldsymbol{u}_q)_l| \bar{\nabla} g(\boldsymbol{\theta}_t; \boldsymbol{u}_q)_l$. Continue the inequality,

$$
\begin{aligned}
& g_\epsilon(\boldsymbol{\theta}_t) - \eta_t \odot \bar{G}_t \|\nabla g_\epsilon(\boldsymbol{\theta}_t)\|_1 + \frac{dL}{2} \eta_t^2 \odot \bar{G}_t{}^2 \\
& + 2\eta_t \odot \bar{G}_t \sum_{l=1}^{d} |(\nabla g_\epsilon(\boldsymbol{\theta}_t))_l| \text{Prob}[\text{sign}((\bar{\boldsymbol{g}}_t)_l) \neq \text{sign}((\nabla g_\epsilon(\boldsymbol{\theta}_t))_l)] \\
& \leq g_\epsilon(\boldsymbol{\theta}_t) - \eta_t \odot \bar{G}_t \|\nabla g_\epsilon(\boldsymbol{\theta}_t)\|_1 + \frac{dL}{2} \eta_t^2 \odot \bar{G}_t{}^2 + 2\eta_t \odot \bar{G}_t \sum_{l=1}^{d} \frac{\delta_l}{\sqrt{Q}} \\
& \leq g_\epsilon(\boldsymbol{\theta}_t) - \eta_t \odot \bar{G}_t \|\nabla g_\epsilon(\boldsymbol{\theta}_t)\|_1 + \frac{dL}{2} \eta_t^2 \odot \bar{G}_t{}^2 + 2\eta_t \odot \bar{G}_t \frac{\|\delta_l\|_1}{\sqrt{Q}} \\
& \leq g_\epsilon(\boldsymbol{\theta}_t) - \eta_t \odot \bar{G}_t \|\nabla g_\epsilon(\boldsymbol{\theta}_t)\|_1 + \frac{dL}{2} \eta_t^2 \odot \bar{G}_t{}^2 + 2\eta_t \odot \bar{G}_t \frac{\sqrt{d}\sqrt{\|\delta_l\|_2^2}}{\sqrt{Q}} \\
& = g_\epsilon(\boldsymbol{\theta}_t) - \eta_t \odot \bar{G}_t \|\nabla g_\epsilon(\boldsymbol{\theta}_t)\|_1 + \frac{dL}{2} \eta_t^2 \odot \bar{G}_t{}^2 + 2\eta_t \odot \bar{G}_t \frac{\sqrt{d}\sqrt{\mathbb{E}\big[((\dot{\nabla} g(\boldsymbol{\theta}_t; \boldsymbol{u}_q) - \nabla g_\epsilon(\boldsymbol{\theta}_t)))_l^2\big]}}{\sqrt{Q}}
\end{aligned}
$$

Thus we will have,

$$
\begin{aligned}
& g_\epsilon(\boldsymbol{\theta}_{t+1}) - g_\epsilon(\boldsymbol{\theta}_t) \leq -\eta_t \odot \bar{G}_t \|\nabla g_\epsilon(\boldsymbol{\theta}_t)\|_1 + \frac{dL}{2} \eta_t^2 \odot \bar{G}_t{}^2 \\
& + 2\eta_t \odot \bar{G}_t \frac{\sqrt{d}\sqrt{\mathbb{E}\big[((\dot{\nabla} g(\boldsymbol{\theta}_t; \boldsymbol{u}_q) - \nabla g_\epsilon(\boldsymbol{\theta}_t)))_l^2\big]}}{\sqrt{Q}} \\
& \Rightarrow \eta_t \odot \bar{G}_t \|\nabla g_\epsilon(\boldsymbol{\theta}_t)\|_1 \leq g_\epsilon(\boldsymbol{\theta}_t) - g_\epsilon(\boldsymbol{\theta}_{t+1}) + \frac{dL}{2} \eta_t^2 \odot \bar{G}_t{}^2 \\
& + 2\eta_t \odot \bar{G}_t \frac{\sqrt{d}\sqrt{\mathbb{E}\big[((\dot{\nabla} g(\boldsymbol{\theta}_t; \boldsymbol{u}_q) - \nabla g_\epsilon(\boldsymbol{\theta}_t)))_l^2\big]}}{\sqrt{Q}} \\
& \Rightarrow \hat{\eta}_t \|\nabla g_\epsilon(\boldsymbol{\theta}_t)\|_1 \leq g_\epsilon(\boldsymbol{\theta}_t) - g_\epsilon(\boldsymbol{\theta}_{t+1}) + \frac{dL}{2} \hat{\eta}_t{}^2 + 2\hat{\eta}_t \sqrt{d} \frac{\sqrt{\mathbb{E}\big[((\dot{\nabla} g(\boldsymbol{\theta}_t; \boldsymbol{u}_q) - \nabla g_\epsilon(\boldsymbol{\theta}_t)))_l^2\big]}}{\sqrt{Q}},
\end{aligned}
$$

where we define $\hat{\eta_t} \coloneqq \eta_t \odot \bar{G}_t$. Sum up all inequalities for all ts and take expectation on both side, we will have

$$\begin{aligned}\sum_{t=1}^{T} \hat{\eta_t} \mathbb{E}[\|\nabla g_\epsilon(\boldsymbol{\theta}_t)\|_1] &\leq \mathbb{E}[g_\epsilon(\boldsymbol{\theta}_1) - g_\epsilon(\boldsymbol{\theta}_T)] + \frac{dL}{2} \sum_{t=1}^{T} {\hat{\eta_t}}^2 \\ &+ \sum_{t=1}^{T} 2\hat{\eta_t}\sqrt{d}\sqrt{\mathbb{E}\big[((\dot{\nabla} g(\boldsymbol{\theta}_t; \boldsymbol{u}_q) - \nabla g_\epsilon(\boldsymbol{\theta}_t)))_l^2\big]} \\ &\leq \mathbb{E}[g_\epsilon(\boldsymbol{\theta}_1) - g_\epsilon(\boldsymbol{\theta}_T)] + \frac{dL}{2} \sum_{t=1}^{T} {\hat{\eta_t}}^2 + \sum_{t=1}^{T} 2\hat{\eta_t}\sqrt{d}\sqrt{\frac{4(Q+1)}{Q}\sigma^2 + \frac{2}{Q}C(d,\epsilon)}\end{aligned}$$

by Lemma 1.

Substitute Lemma 3 into above inequality, we get

$$\sum_{t=1}^{T} \hat{\eta_t} \mathbb{E}[\|\nabla g_\epsilon(\boldsymbol{\theta}_t)\|_1] \leq g_\epsilon(\boldsymbol{\theta}_1) - g^* + \epsilon^2 L + \frac{dL}{2} \sum_{t=1}^{T} {\hat{\eta_t}}^2 + \sum_{t=1}^{T} 2\hat{\eta_t}\sqrt{d}\sqrt{\frac{4(Q+1)}{Q}\sigma^2 + \frac{2}{Q}C(d,\epsilon)}.$$

Since $\|\cdot\|_2 \leq \|\cdot\|_1$ and we could divide $\sum_{t=1}^{T} \hat{\eta_t}$ on both side to get

$$\begin{aligned}\sum_{t=1}^{T} \frac{\hat{\eta_t}}{\sum_{t=1}^{T} \hat{\eta_t}} \mathbb{E}[\|\nabla g_\epsilon(\boldsymbol{\theta}_t)\|_2] &\leq \frac{g_\epsilon(\boldsymbol{\theta}_1) - g^* + \epsilon^2 L}{\sum_{t=1}^{T} \hat{\eta_t}} + \frac{dL}{2} \frac{\sum_{t=1}^{T} {\hat{\eta_t}}^2}{\sum_{t=1}^{T} \hat{\eta_t}} \\ &+ \sum_{t=1}^{T} \frac{2\sqrt{d}}{\sqrt{Q}} \sqrt{4(Q+1)\sigma^2 + 2C(d,\epsilon)}.\end{aligned}$$

Define a new random variable R with probability $P(R=t) = \frac{\eta_t}{\sum_{t=1}^{T} \eta_t}$, we will have

$$\mathbb{E}[\|\nabla g_\epsilon(\boldsymbol{\theta}_R)\|_2] = \mathbb{E}[\mathbb{E}_R[\|\nabla g_\epsilon(\boldsymbol{\theta}_R)\|_2]] = \mathbb{E}\Big[\sum_{t=1}^{T} P(R=t)\|\nabla g_\epsilon(\boldsymbol{\theta}_t)\|_2\Big].$$

Substitute all the quantities into Lemma 4, we will get

$$\begin{aligned}\mathbb{E}[\|\nabla g(\boldsymbol{\theta})\|_2] &\leq \frac{\sqrt{2}(g_\epsilon(\boldsymbol{\theta}_1) - g^* + \epsilon^2 L)}{\sum_{t=1}^{T} \hat{\eta_t}} + \frac{dL}{\sqrt{2}} \frac{\sum_{t=1}^{T} {\hat{\eta_t}}^2}{\sum_{t=1}^{T} \hat{\eta_t}} \\ &+ \frac{\epsilon L d}{\sqrt{2}} + \sum_{t=1}^{T} \frac{2\sqrt{2}\sqrt{d}}{\sqrt{Q}} \sqrt{4(Q+1)\sigma^2 + 2C(d,\epsilon)}.\end{aligned}$$

By choosing $\epsilon = O(\frac{1}{dT})$ and $\eta_t = O(\frac{1}{Q\sqrt{dT}})$, then the convergence rate as shown in above is $O(\frac{d}{T} + \frac{d}{\sqrt{Q}})$. $\square$

CHAPTER 3

Adversarial Attacks on Discrete Domain

3.1 Seq2Sick: Evaluating the Robustness of Sequence-to-Sequence Models with Adversarial Examples

Models designed for different tasks are not born equal: some tasks are strictly harder to attack than others. For example, attacking an image is much easier than attacking a text string, since image space is continuous and the adversary can make arbitrarily small changes to the input. Therefore, even if most of the pixels of an image have been modified, the perturbations can still be imperceptible to humans when the accumulated distortion is small. In contrast, text strings live in a discrete space, and word-level manipulations may significantly change the meaning of the text. In this scenario, an adversary should change as few words as possible, and hence this limitation induces a sparse constraint on word-level changes. Likewise, attacking a classifier should also be much easier than attacking a model with sequence outputs. This is because different from the classification problem that has a finite set of discrete class labels, the output space of sequences may have an almost infinite number of possibilities. If we treat each sequence as a label, a targeted attack needs to find a specific one over an enormous number of possible labels, leading to a nearly zero volume in search space. This may explain why most existing works on adversarial attacks focus on the image classification task since its input space is continuous and its output space is finite.

In this chapter, we study a harder problem of crafting adversarial examples for sequence-to-sequence (seq2seq) models (SVL14). This problem is challenging since it combines both

aforementioned difficulties, i.e., discrete inputs and sequence outputs with an almost infinite number of possibilities. We choose this problem not only because it is challenging, but also because seq2seq models are widely used in many safety and security sensitive applications, e.g., machine translation (BCB14), text summarization (RCW15), and speech recognition (CJL16), thus measuring its robustness becomes critical. Specifically, we aim to examine the following questions in this study:

1. *Is it possible to slightly modify the inputs of seq2seq models while significantly change their outputs?*

2. *Are seq2seq models more robust than the well-evaluated CNN-based image classifiers?*

We provide an affirmative answer to the first question by developing an effective adversarial attack framework called Seq2Sick. It is an optimization-based framework that aims to learn an input sequence that is close enough to the original sequence (in terms of distance metrics in word embedding spaces or sentiment classification) while leads to the desired outputs with high confidence. To address the challenges caused by the discrete input space, we propose to use the projected gradient descent method combined with group lasso and gradient regularization. To address the challenges of almost infinite output space, we design some novel loss functions for the tasks of non-overlapping attack and targeted keyword attack. Our experimental results show that the proposed framework yields high success rates in both tasks. However, even if the proposed approach can successfully attack seq2seq models, our answer to the second question is "Yes". Compared with CNN-based classifiers that are highly sensitive to adversarial examples, seq2seq model is intrinsically more robust since it has discrete input space and the output space is exponentially large. As a result, adversarial examples of seq2seq models usually have larger distortions and are more perceptible than the adversarial examples crafted for CNN-based image classifiers.

3.1.1 Problem Setting

For sequential output task such machine translation and text summarization, we have to use a different strategy. Obviously, attacking a classifier should also be much easier than attacking a model with sequence outputs. It is because different from the classification problem that has a finite set of discrete class labels, the output space of sequences may have an almost infinite number of possibilities. If we treat each sequence as a label, a targeted attack needs to find a specific one over an enormous number of possible labels, leading to a nearly zero volume in search space.

We consider the sequence-to-sequence (seq2seq) model as follows. Let $\mathbf{x}_i \in \mathbb{R}^d$ be the embedding vector of each input word, N be the input sequence length, and M be the output sequence length. Let ω be the input vocabulary, and the output word $\mathbf{y}_j \in \nu$ where ν is the output vocabulary. The seq2seq model has an encoder-decoder framework that aims at mapping an input sequence of vectors $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)$ to the output sequence $\mathbf{Y} = \{\mathbf{y}_1, \ldots, \mathbf{y}_M\}$. Its encoder first reads the input sequence, then each RNN/LSTM cell computes $\mathbf{h}_t = f(\mathbf{x}_t,\ \mathbf{h}_{t-1})$, where $\mathbf{x}_t$ is the current input, $\mathbf{h}_{t-1}$ and $\mathbf{h}_t$ represent the previous and current cells' hidden States, respectively. The next step computes the context vector $\mathbf{c}$ using all the hidden layers of cells $\mathbf{h}_1, \ldots, \mathbf{h}_N$, i.e $\mathbf{c} = q(\mathbf{h}_1, \cdots, \mathbf{h}_N)$, where $q(\cdot)$ could be a linear or non-linear function. In this chapter, we follow the setting in (SVL14) that $\mathbf{c} = q(\mathbf{h}_1, \cdots, \mathbf{h}_N) = \mathbf{h}_N$.

Given the context vector $\mathbf{c}$ and all the previously words $\{\mathbf{y}_1, \ldots, \mathbf{y}_{t-1}\}$, the decoder is trained to predict the next word $\mathbf{y}_t$. Specifically, the t-th cell in the decoder receives its previous cell's output $\mathbf{y}_{t-1}$ and the context vector $\mathbf{c}$, and then outputs

$$z_t = g(\mathbf{y}_{t-1}, \mathbf{c}) \quad \text{and} \quad p_t = \text{softmax}(z_t), \tag{3.1}$$

where g is another RNN/LSTM cell function. $z_t := [z_t^{(1)}, z_t^{(2)}, \ldots, z_t^{(|\nu|)}] \in \mathbb{R}^{|\nu|}$ is a vector of the *logits* for each possible word in the output vocabulary ν.

Crafting adversarial examples against the seq2seq model can be formulated as an opti-

mization problem:

$$\min_{\boldsymbol{\delta}} \; L(\mathbf{X}+\boldsymbol{\delta}) + \lambda \cdot R(\boldsymbol{\delta}), \tag{3.2}$$

where $R(\cdot)$ indicates the regularization function to measure the magnitude of distortions. $L(\cdot)$ is the loss function to penalize the unsuccessful attack and it may take different forms in different attack scenarios. A common choice for $R(\boldsymbol{\delta})$ is the ℓ_2 penalty $\|\boldsymbol{\delta}\|_2^2$, but it is, as we will show later, not suitable for attacking seq2seq model. $\lambda > 0$ is the regularization parameter that balances the distortion and attack success rate – a smaller λ will make the attack more likely to succeed but with the price of larger distortion.

Here I introduce two kinds of attacks: *non-overlapping attack* and *targeted keywords attack*. The first attack requires that the output of the adversarial example shares no overlapping words with the original output. This task is strictly harder than untargeted attack, which only requires that the adversarial output to be different from the original output (ZDS17; ERL17). We ignore the task of untargeted attack since it is trivial for the proposed framework, which can easily achieve a 100% attack success rate, while (ERL17) could achieve 76.24% attack success rate for text summarization and 98.8% success rate for machine translation with 1 word change. Targeted keywords attack is an even more challenging task than non-overlapping attack. Given a set of targeted keywords, the goal of targeted keywords attack is to find an adversarial input sequence such that all the keywords must appear in its corresponding output

Non-overlapping Attack: To formally define the non-overlapping attack, we let $\mathbf{s} = \{s_1, \ldots, s_M\}$ be the original output sequence, where s_i denotes the location of the i-th word in the output vocabulary ν. $\{z_1, \ldots, z_M\}$ indicates the logit layer outputs of the adversarial example. In the non-overlapping attack, the output of adversarial example should be entirely different from the original output $\mathbf{S}$, i.e.,

$$s_t \neq \text{argmax}_{y \in \nu} \, z_t^{(y)}, \quad \forall t = 1, \ldots, M,$$

Targeted Keywords Attack Given a set of targeted keywords, the goal of targeted keywords attack is to generate an adversarial input sequence to ensure that all the targeted keywords appear in the output sequence. This task is important since it suggests adding a few malicious keywords can completely change the meaning of the output sequence. For example, in English to German translation, an input sentence "*policeman helps protesters to keep the assembly in order*" should generate an output sentence "*Polizist hilft Demonstranten, die Versammlung in Ordnung zu halten*". However, changing only one word from "hilft" to "verhaftet" in the output will significantly change its meaning, as the new sentence means "*police officer arrested protesters to keep the assembly in order*".

3.1.2 Handling Discrete Input Space

As mentioned before, the problem of "discrete input space" is one of the major challenges in attacking seq2seq model. Let $\mathbb{W}$ be the set of word embeddings of all words in the input vocabulary. A naive approach is to first learn $\mathbf{X} + \boldsymbol{\delta}^*$ in the continuous space by solving the problem (3.2), and then search for its nearest word embedding in $\mathbb{W}$. This idea has been used in attacking sequence classification models in (GWL18). Unfortunately, when applying this idea to targeted keywords attack, we report that all of the 100 attacked sequences on Gigaword dataset failed to generate the targeted keywords. The main reason is that by directly solving (3.2), the final solution will not be a feasible word embedding in $\mathbb{W}$, and its nearest neighbor could be far away from it due to the curse of dimensionality (Fri97).

To address this issue, we propose to add an additional constraint to enforce that $\mathbf{X} + \boldsymbol{\delta}$ belongs to the input vocabulary $\mathbb{W}$. The optimization problem then becomes

$$\begin{aligned} \min_{\boldsymbol{\delta}} \quad & L(\mathbf{X} + \boldsymbol{\delta}) + \lambda \cdot R(\boldsymbol{\delta}) \\ \text{s.t.} \quad & \mathbf{x}_i + \delta_i \in \mathbb{W} \quad \forall i = 1, \ldots, N \end{aligned} \tag{3.3}$$

We then apply projected gradient descent to solve this constrained problem. At each iteration, we project the current solution $\mathbf{x}_i + \delta_i$, where δ_i denotes the i-th column of $\boldsymbol{\delta}$, back into $\mathbb{W}$ to ensure that $\mathbf{X} + \boldsymbol{\delta}$ can map to a specific input word.

Group lasso Regularization: ℓ_2 norm has been widely used in the adversarial machine learning literature to measure distortions. However, it is not suitable for our task since almost all the learned $\{\delta_t\}_{t=1}^M$ using ℓ_2 regularization will be nonzero. As a result, most of the inputs words will be perturbed to another word, leading to an adversarial sequence that is significantly different from the input sequence.

To solve this problem, we treat each δ_t with d variables as a group, and use the group lasso regularization

$$R(\boldsymbol{\delta}) = \sum\nolimits_{t=1}^{N} \|\delta_t\|_2$$

to enforce the group sparsity: only a few groups (words) in the optimal solution $\boldsymbol{\delta}^*$ are allowed to be nonzero.

Gradient Regularization When attacking the seq2seq model, it is common to find that the adversarial example is located in a region with very few or even no embedding vector. This will negatively affect our projected gradient method since even the closest embedding from those regions can be far away.

To address this issue, we propose a gradient regularization to make $\mathbf{X} + \boldsymbol{\delta}$ close to the word embedding space. Our final objective function becomes:

$$\begin{aligned} &\min_{\boldsymbol{\delta}} L(\mathbf{X}+\boldsymbol{\delta}) + \lambda_1 \sum_{i=1}^{N} \|\delta_i\|_2 + \lambda_2 \sum_{i=1}^{N} \min_{\boldsymbol{w}_j \in \mathbb{W}} \{\left\|\mathbf{x}_i + \delta_i - \boldsymbol{w}_j\right\|_2\} \\ &\text{s.t. } \mathbf{x}_i + \delta_i \in \mathbb{W} \quad \forall i = 1, \ldots, N \end{aligned} \tag{3.4}$$

where the third term is our gradient regularization that penalizes a large distance to the nearest point in $\mathbb{W}$. The gradient of this term can be efficiently computed since it is only related to one $\boldsymbol{w}_j$ that has a minimum distance from $\mathbf{x}_i + \delta_i$. For the other terms, we use the proximal operator to optimize the group lasso regularization, and the gradient of the loss function L can be computed through back-propagation. The detailed steps of our approach, Seq2Sick, is presented in Algorithm 6. Our source code is publicly available at

Algorithm 6 Seq2Sick algorithm

Input: input sequence $\boldsymbol{x} = \{x_1, \ldots, x_N\}$, seq2seq model, target keyword $\{k_1, \ldots, k_T\}$
Output: adversarial sequence $\boldsymbol{x}^* = \boldsymbol{x} + \boldsymbol{\delta}^*$
Let $\mathbf{s} = \{s_1, \ldots, s_M\}$ denote the original output of $\boldsymbol{x}$.
Set the loss $L(\cdot)$ in (3.4) to be (2.8)
if Targeted Keyword Attack **then**
  Set the loss $L(\cdot)$ in (3.4) to be (2.12)
for $r = 1, 2, \ldots, T$ **do**
  back-propagation L to achieve gradient $\nabla_{\boldsymbol{\delta}} L(\boldsymbol{x} + \boldsymbol{\delta}_r)$
  for $i = 1, 2, \ldots, N$ **do**
    if $\left\|\delta_{r,i}\right\| > \eta\lambda_1$ **then**
      $\delta_{r,i} = \delta_{r,i} - \eta\lambda_1 \frac{\delta_{r,i}}{\left\|\delta_{r,i}\right\|}$
    else
      $\delta_{r,i} = 0$
  $y^{r+1} = \boldsymbol{\delta}^r + \eta \cdot \nabla_{\boldsymbol{\delta}} L(\boldsymbol{x} + \boldsymbol{\delta}^r)$
  $\boldsymbol{\delta}^{r+1} = \underset{\boldsymbol{x}+\boldsymbol{\delta}^{r+1} \in \mathbb{W}}{\operatorname{argmin}} \left\|y^{r+1} - \boldsymbol{\delta}^{r+1}\right\|$
$\boldsymbol{\delta}^* = \boldsymbol{\delta}^T$
$\boldsymbol{x}^* = \boldsymbol{x} + \boldsymbol{\delta}^*$
return $\boldsymbol{x}^*$

`https://github.com/cmhcbb/Seq2Sick`.

3.1.3 Experimental Results

We conduct experiments on two widely-used applications of seq2seq model: text summarization and machine translation.

Datasets We use three datasets DUC2003, DUC2004, and Gigaword, to conduct our attack for the text summarization task. Among them, DUC2003 and DUC2004 are widely-used datasets in documentation summarization. We also include a subset of randomly chosen samples from Gigaword to further evaluate the performance of our algorithm. For the machine translation task, we use 500 samples from WMT'16 Multimodal Translation task. The statistics about the datasets are shown in Table 3.1.

Table 3.1: Statistics of the datasets. "# Samples" is the number of test examples we used for robustness evaluations

Datasets	# samples	Average input lengths
Gigaword	1,000	30.1 words
DUC2003	624	35.5 words
DUC2004	500	35.6 words
Multi30k	500	11.5 words

Seq2seq models We implement both text summarization and machine translation models on OpenNMT-py. Specifically, we use a word-level LSTM encoder and a word-based attention decoder for both applications (BCB14). For the text summarization task, we use 380k training pairs from Gigaword dataset to train a seq2seq model. The architecture consists of a 2-layer stacked LSTM with 500 hidden units. We conduct experiments on two types of models, one uses the pre-trained 300-dimensional GloVe word embeddings and the other one is trained from scratch. We set the beam search size to be 5 as suggested. For the machine translation task, we train our model using 453k pairs from the Europal corpus of German-English WMT 15, common crawl and news-commentary. We use the hyper-parameters suggested by OpenNMT for both models, and have reproduced the performance reported in (RCW15) and (HNW16).

3.1.4 Empirical Results

Text Summarization For the non-overlapping attack, we use the proposed loss (2.8) in our objective function. A non-overlapping attack is treated as successful only if there is no common word at every position between output sequence and original sequence. We set $\lambda = 1$ in all non-overlapping experiments. Table 3.2 summarizes the experimental results. It shows that our algorithm only needs to change 2 or 3 words on average and can generate entirely different outputs for more than 80% of sentences. We have also included some adversarial

examples in Table 3.7. From these examples, we can only change one word to let output sequence look completely different with the original one and change the sentence's meaning completely.

Table 3.2: Results of non-overlapping attack in text summarization. **# changed** is how many words are changed in the input sentence. The high BLEU scores and low average number of changed words indicate that the crafted adversarial inputs are very similar to their originals, and we achieve high success rates to generate a summarization that differs with the original *at every position* for all three datasets.

Dataset	Success%	BLEU	# changed
Gigaword	86.0%	0.828	2.17
DUC2003	85.2%	0.774	2.90
DUC2004	84.2%	0.816	2.50

For the targeted keywords attack, we randomly choose some targeted keywords from the output vocabulary after removing the stop words like "a" and "the". A targeted keywords attack is treated as successful only if the output sequence contains all the targeted keywords. We set $\lambda_1 = \lambda_2 = 1$ in our objective function (3.4) in all our experiments. Table 3.3 summarizes the performance, including the overall success rate, average BLEU score (PRW02), and the average number of changed words in input sentences. Average BLEU score is defined by exponential average over BLEU 1,2,3,4, which is commonly used in evaluating the quality of text which has been machine-translated from one natural language to another. Also, we have included some adversarial examples crafted by our method in Table 3.8. In Table 3.8, some adversarial examples with 3 sets of keywords, where "##" stands for a two-digit number after standard preprocessing in text summarization. Through these examples, our method could generate totally irrelevant subjects, verbs, numerals and objects which could easily be formed as a complete sentence with only several word changes. Note that there are three important techniques used in our algorithm: projected gradient method, group lasso, and gradient regularization. Therefore, we conduct experiments to

verify the importance of each of these techniques.

Table 3.3: Results of targeted keywords attack in text summarization. $|K|$ is the number of keywords. We found that our method can make the summarization include 1 or 2 target keywords with a high success rate, while the changes made to the input sentences are relatively small, as indicated by the high BLEU scores and low average number of changed words. When $|K| = 3$, this task becomes more challenging, but our algorithm can still find many adversarial examples.

Datasest	$\|K\|$	Success%	BLEU	# changed
Gigaword	1	99.8%	0.801	2.04
	2	96.5%	0.523	4.96
	3	43.0%	0.413	8.86
DUC2003	1	99.6%	0.782	2.25
	2	87.6%	0.457	5.57
	3	38.3%	0.376	9.35
DUC2004	1	99.6%	0.773	2.21
	2	87.8%	0.421	5.1
	3	37.4%	0.340	9.3

Machine Translation We then conduct both non-overlapping and targeted keywords attacks to the English-German machine translation model. We first filter out stop words like "Ein"(a), "und"(and) in German vocabulary and randomly choose several nouns, verbs, adjectives or adverbs in German as targeted keywords. Similar to the text summarization experiments, we set $\lambda_1 = \lambda_2 = 1$ in our objective function. The success rates, BLEU scores, and the average number of words changed are reported in Table 3.4, with some adversarial examples shown in Table 3.6.

3.1.4.1 Analysis of Syntactic structure and Semantic Meaning Preservation

In our algorithm we aim to make adversarial examples having similar meaning to original examples by constraining the number of changed words and enforcing the changed words are

Table 3.4: Results of non-overlapping method and targeted keywords method in machine translation.

Method	Success%	BLEU	# changed
Non-overlap	89.4%	0.349	3.5
1-keyword	100.0%	0.705	1.8
2-keyword	91.0 %	0.303	4.0
3-keyword	69.6%	0.205	5.3

Table 3.5: Perplexity score for adversarial example

	DUC2003	DUC2004
Original	102.02	121.09
Non-overlap	114.02	149.15
1-keyword	159.54	199.01
2-keyword	352.12	384.80

close to the original words in the embedding space. However, depending on the implemented word embedding techniques, in general there is no guarantee that every word pair close in the embedding space have similar meanings. Therefore, we have conducted additional experiments to verify the syntactic and semantic quality of our generated adversarial examples. For syntactic structure part, as showed in Table 3.5, we measure the perplexity of generated adversarial sentences in DUC2003 and DUC2004 dataset. It shows that our examples keeps the original syntactic structure. For the semantic meaning part, We use DeepAI's online sentiment analysis API to test whether our attack changes the sentiment of 500 sentences from DUC2003 dataset in summarization task. The results show that **only 2.2% of adversarial examples have semantic meaning differ from the original sentences**. It proves that almost all adversarial examples keep the same semantic classification unchanged.

3.1.5 Analysis and Discussions

Observation from adversarial example As shown in Table 3.8, our targeted keyword attack wouldn't just directly replace the keyword with some word in the source input. However, the word changed in the adversarial example and the target keyword are co-occurrent in the training dataset. It infers that seq2seq model learns the relationship between changed word and target keyword. However, the model fails to decide where it should focus on, which is strongly related with attention layer used in the model. It encourages us to use self-attention such as transformer (VSP17) instead to extract all the attentions between any two words.When attacking subword transformer model, the target 1 keyword attack has 17% lower success rate and 0.13 lower BLEU score. It shows transformer model has a greater adversarial robustness.

Robustness of Seq2Seq Model Although our algorithm can achieve very good success rates ($84\% - 100\%$) in both non-overlapping and targeted keywords attacks with 1 or 2 keywords, we also recognize some strengths of the seq2seq model: (i) unlike CNN models where targeted attack can be conducted easily with almost 100% success rate and very small distortion that cannot be perceived by human eyes (CW17), it is harder to turn the entire seq2seq output into a particular sentence – some sentences are even impossible to generate by seq2seq models; and (ii) since the input space of seq2seq is discrete, it is easier for human to detect the differences between the adversarial sequence and the original one, even if we only change one or few words. Therefore, we conclude that, compared with the DNN models designed for other tasks such as image classification, seq2seq models are more robust to adversarial attacks. The main reason, as pointed out in the introduction, is that the seq2seq model has a finite and discrete input space and almost infinite output space, so it is more robust than visual classification models that have an infinite and continuous input space and a very small output space (e.g., 10 categories in MNIST and 1,000 categories in ImageNet).

Table 3.6: Machine translation adversarial examples. Upper 4 lines: non-overlap; Bottom 4 lines: targeted keyword "Hund sitzt"

Source input seq	A child is splashing in the water.
Adv input seq	A children is **unionists** in the water.
Source output seq	Ein Kind im Wasser.
Adv output seq	**Kinder sind in der Wasser @-@ <unk>**.
Source input seq	Two men wearing swim trunks jump in the air at a moderately populated beach.
Adv input seq	Two men wearing **dog Leon comes** in the air at a moderately populated beach.
Source output seq	Zwei Männer in Badehosen springen auf einem mäßig belebten Strand in die Luft.
Adv output seq	Zwei Männer tragen **Hund** , der in der Luft **sitzt** , hat <unk> <unk> .

3.2 AdvAgent: Evaluating and Enhancing the Robustness of Dialogue Systems

3.2.1 Competitive Negotiation Dialogues

We use the negotiation agent developed in (LYD17) as the running example in this chapter. Note that our algorithm can be generalized to other goal-oriented dialogue systems by designing a different scoring function according to the task.

In a competitive negotiation dialogue setting, two agents are negotiating with each other over a set of items. We adopt the same setting as (LYD17), in which case items can be categorized into either a ball, a hat or a book. Each agent is given the goal of the conversation (denoted by g), which contains the initial values and the quantities of each of the three items. Agents then negotiate to maximize the total value of their possessed items. Agents are allowed to negotiate up to a maximum of 10 turns. Scores will be granted to agents based on the total value of the items if they reach an agreement. If they choose not to agree, 0 score will be granted to both agents. A competitive negotiation dialogue example played by

Table 3.7: Text summarization adversarial examples using non-overlapping method. Surprisingly, it is possible to make the output sequence completely different by changing only one word in the input sequence.

Source input seq	among asia 's leaders , prime minister mahathir mohamad was notable as a man with a bold vision : a physical and social transformation that would push this nation into the forefront of world affairs .
Adv input seq	among **lynn** 's leaders , prime minister mahathir mohamad was notable as a man with a bold vision : a physical and social transformation that would push this nation into the forefront of world affairs.
Source output seq	asia 's leaders are a man of the world
Adv output seq	**a vision for the world**
Source input seq	under nato threat to end his punishing offensive against ethnic albanian separatists in kosovo , president slobodan milosevic of yugoslavia has ordered most units of his army back to their barracks and may well avoid an attack by the alliance , military observers and diplomats say
Adv input seq	under nato threat to end his punishing offensive against ethnic albanian separatists in kosovo , president slobodan milosevic of yugoslavia has **jean-sebastien** most units of his army back to their barracks and may well avoid an attack by the alliance , military observers and diplomats say.
Source output seq	milosevic orders army back to barracks
Adv output seq	**nato may not attack kosovo**
Source input seq	flooding on the yangtze river remains serious although water levels on parts of the river decreased today , according to the State headquarters of flood control and drought relief .
Adv input seq	flooding **that** the yangtze river **becomes** serious although water levels on parts of the river decreased today , according to the State headquarters of flood control and drought relief .
Source output seq	floods on yangtze river continue
Adv output seq	**flooding in water recedes in river**

Table 3.8: Text summarization examples with targeted keywords “police arrest ”

Source input seq	north korea is entering its fourth winter of chronic food shortages with its people malnourished and at risk of dying from normally curable illnesses , senior red cross officials said tuesday.
Adv input seq	north **detectives** is **apprehended** its fourth winter of chronic food shortages with its people malnourished and at risk of dying from normally curable illnesses , senior red cross officials said tuesday.
Source output seq	north korea enters fourth winter of food shortages
Adv output seq	north **police arrest** fourth winter of food shortages.
Source input seq	after a day of fighting , congolese rebels said sunday they had entered kindu , the strategic town and airbase in eastern congo used by the government to halt their advances.
Adv input seq	after a day of fighting , **nordic detectives** said sunday they had entered **UNK** , the strategic town and airbase in eastern congo used by the government to halt their advances.
Source output seq	congolese rebels say they have entered UNK.
Adv output seq	nordic **police arrest** ## in congo.
Source input seq	president boris yeltsin stayed home tuesday , nursing a respiratory infection that forced him to cut short a foreign trip and revived concerns about his ability to govern.
Adv input seq	president boris yeltsin stayed home tuesday , **cops cops** respiratory infection that forced him to cut short a foreign trip and revived concerns about his ability to govern.
Source output seq	yeltsin stays home after illness
Adv output seq	yeltsin stays home after **police arrest**

Input		
Human	3x**book** value 2 3x**hat** value 1 1x**ball** value 1	
Agent	3x**book** value 1 3x**hat** value 2 1x**ball** value 1	
Human	I'd like the books and the hats.	
Agent	I will take the books and hats.	
Human	I need the books and at least one other object.	
Agent	Ok, you can have one book and one ball	
Human	If I can have all books, I can leave you the rest.	
Agent	Deal	
Human	$\langle selection \rangle$	
Output		**Reward**
Human	3x**hat** 1x**ball**	7/10
Agent	3x**book**	6/10

Table 3.9: Competitive negotiation dialogue generated between agent and human.

human and agent could be found in Table 3.9.

We assess the robustness of a trained end-to-end negotiation agent used in (LYD17). In the negotiation chatbot setting, agents first chat using natural language and then make a selection based on what they have chatted with. We refer to the first phase as **negotiation phase** and the second phase as **decision phase**.

In the negotiation phase, conversation response at time t, x_t is generated word by word based on chat history $x_{0..t-1}$ and the goal of the conversation g. The conversation model is controlled by a speaking module θ and tokens are randomly sampled from probability distribution p_θ. This process continues recursively until an end-of-sentence token $\langle EOS \rangle$ or selection token $\langle selection \rangle$ token is generated. When $\langle EOS \rangle$ is encountered, the turn terminates and the conversation is handled to another agent. When $\langle selection \rangle$ is encountered, the negotiation phase terminates and the negotiation will reach the decision phase.

$$x_t \sim p_\theta(x_t | x_{0...t-1}, g) \tag{3.5}$$

In the decision phase, both agents will output a decision o based on a decision module probability distribution p'_θ. Agents' decisions will be based on conversation history $x_{0...T}$ up to the current time step T and the goal of the conversation g. Here O is a set of all legitimate

selections, which is defined to be a space of where each selection must be greater or equal than 0 and the sum of selections for the same item must be equal to its original quantity. Since we only have a few items, it is possible to enumerate all the possibilities to get the set O.

$$o^* = \underset{o \in O}{\operatorname{argmax}} \prod_i p'_\theta(o_i | x_{0...T}, g) \tag{3.6}$$

Agents will then collect rewards (i.e. scores) from the environment (which will be 0 if they output conflicted decisions, e.g. the total number of items are different from the initial amount). It is important to keep the agent producing sentences that are correct both grammatically and semantically and keeping them competitive at the same time. Therefore, a common strategy is to train agents using supervised learning to learn natural language and to use reinforcement learning to optimize models' performance using on goal-oriented learning. We measure two statistics **score** and **agreement**. **score** is the average score for each agent (0-10). **agreement** is the percentage of dialogues where both agents agreed on the same decision. To measure the extent of success of our adversarial agent, we use **advantage** which is easy to compute directly from adversarial agent score minus target agent score, i.e. $S_{adv} - S_{ori}$.

3.2.2 Proposed Black-box Attack Algorithms

We first build our adversarial agent in black-box setting. Black-box setting in goal-oriented dialogue system is defined where the target agent is unknown to the attacker, but it is possible to make queries to obtain the final decision made by the target agent. To be noted, our aim is to test the robustness of the target agent. Therefore, in the decision phase we let adversarial agent chooses the complementary of target agent's choice, so those two agents will always reach agreement. The adversarial agent thus only has the speaking module and there is no decision network needed. In this section we proposed two adversarial agents in the black-box

setting.

3.2.2.1 Reinforcement learning attack

Inspired by the procedure of goal-based reinforcement learning, we modified the reward function of our adversarial agent with the advantage instead of the score he got:

$$r^{adv} = S_{adv} - S_{ori} \tag{3.7}$$

where S_{adv} and S_{ori} are adversarial agent score and target agent score respectively. After a complete dialogue has been generated, we update adversarial agent's parameters based on the outcome of the negotiation.

To learn the adversarial agent's speaking network by reinforcement learning, we denote the subset of tokens generated by the adversarial agent as X^{adv}. In the completed dialogue, γ is the discount factor that rewards actions at the end of the dialogue more strongly, and μ is a running average of completed dialogue rewards so far. We define the future reward R for an action $x_t \in X^{adv}$ as follows:

$$R(x_t) = \sum_{x_t \in X^{adv}} \gamma^{T-t}(r^{adv} - \mu). \tag{3.8}$$

Then by a standard policy gradient algorithm, we could train our adversarial agent. Note that this attack doesn't require the knowledge on the target agent's structure/weights, and the experimental results demonstrate significant attack performance over regular agents.

3.2.2.2 Transfer attack

Transfer attack is a popular idea for attacking black-box models (PMG17). In dialogue systems, we can also consider the following transfer process: a sentence that leads to low r^{adv} in one dialogue might also lead to similar results in another dialogue. To implement this idea, we first collect a list of last sentences spoken by the adversarial agent from dialogues with high reward, denoted by L. In the conversations, we let our adversarial agent and the target agent negotiate n turns using the regular speaking module, and then plug in one sentence in

L at the $(n+1)$-th turn. Our experimental results show that this transfer attack does not work well in practice.

3.2.3 Proposed White-box Attack Algorithms

In the white-box setting, we assume that the attacker can access every part of the target agent, including the weights of both speaking and decision models, and the decision output in every dialogue. Similar to the black-box attacks, we let the adversarial agent choose the complementary of target agent's choices to ensure 100% agreement. By exploiting the knowledge of the target agent's model, white-box attacks can achieve much higher advantage than black-box attacks.

3.2.3.1 Force target agent to select at a fixed turn

To begin with, we consider a simplified strategy where we first let our adversarial agent and the target agent negotiate n turns using regular speaking module. For the $(n+1)$-th turn, we propose the following two ways to modify the output of regular speaking module to maximize the rewards of adversarial agent.

3.2.3.2 Reactive attack

The first strategy is that the adversarial agent produces a sentence that forces the target agent to say $\langle selection \rangle$. The conversation will then enter the decision phase. At the same time, the sentence produced by the adversarial agent should guide the target agent to make a bad selection that would be in favor of the adversarial agent. We call this method **reactive attack**.

We formulate this strategy as an optimization problem. Let $\hat{\mathbf{x}} = x_{t_n \ldots T-1}$ be the output sentence generated by adversarial agent in the speaking model after n -th turn. Specifically, we define $x_{0 \ldots T-1}$ as all the tokens in the dialogue history before $\langle selection \rangle$. $Z_r(x_{0 \ldots T-1})$

indicates the logit layer outputs for predicting x_T based on chat history $x_{0...T-1}$ in the speaking model. $Z_o(x_{0...T})$ indicates the logit layer outputs on conversation history $x_{0...T}$ in the decision model. Because we have a constraint to force the target agent to say the end-of-dialog token $\langle selection \rangle$, we could format this constraint as

$$[Z_r(x_{0...T-1})]_{k_{sel}} - \max_{i \neq k_{sel}} [Z_r(x_{0...T-1})]_i \geq 0 \tag{3.9}$$

where k_{sel} is the corresponding index of end-of-dialog token $\langle selection \rangle$.

At the same time, the score of output o should be in favor of our adversarial agent. Assume the original decision output is o',

$$L(\hat{\mathbf{x}}) = \max\{[Z_o(x_{0...T})]_{o'} - \max_{o \in \bar{O}} [Z_o(x_{0...T})]_o, -\kappa\} \tag{3.10}$$

where $\bar{O}$ is the set of outputs that score of adversarial agent is greater than target agent i.e. $\bar{O} = \{o \in \bar{O} | S_{adv}(o) > S_{ori}(o)\}$, and $\kappa \geq 0$ denotes the confidence margin parameter. Note that $\hat{\mathbf{x}}$ is a sub-sequence in $x_{0...T}$, so the right hand side of (3.10) is a function of $\hat{\mathbf{x}}$.

Combining these two equations together, we can get our final objective function:

$$\begin{aligned} &\min_{\hat{\mathbf{x}}} && L(\hat{\mathbf{x}}) \\ &\text{s.t.} && [Z_r(x_{0...T-1})]_{k_{sel}} - \max_{i \neq k_{sel}} [Z_r(x_{0...T-1})]_i \geq 0 \end{aligned} \tag{3.11}$$

Eq (3.11) is a discrete optimization problem since $\hat{x}$ is the sentence produced by adversarial agent.

In this chapter, we use a modified version of the greedy algorithm to optimize (3.11). Although the original algorithm proposed in (YCH18) only considered the unconstrained discrete problem, we show that the following slightly modified version performs well for solving (3.11). At each iteration, we try to replace each word in $\hat{x}$ by the special token $\langle PAD \rangle$. A word that achieves minimal loss after swapping with $\langle PAD \rangle$ is then selected as the word to be replaced. Then we try to replace the selected word with each word in the vocabulary. For all the trials that satisfy the constraint, we choose the one with minimal loss and conduct the actual change. We run this procedure iteratively to minimize (3.11). In the experiments, we only replace two words in $\hat{x}$ to ensure the fluency and correctness of the adversarial sentences.

3.2.3.3 Preemptive attack

The other attack strategy is to produce a sentence to guide the target agent to lower its demand in the reply instead of making target agent say end-of-dialog token. And after the reply from target agent, the adversarial agent speaks the end-of-dialogue token to enter the decision phase. Similar to the reactive attack, adversarial agent's score should be greater than target agent's score in the decision phase. Clearly, this strategy is more challenging than the previous one because there is an intermediate sentence spoken by the target agent before end-of-dialogue. We call this preemptive attack.

Let $\hat{\mathbf{x}} = x_{t_n \dots t_{n_T}}$ be the output sentence generated by adversarial agent in the speaking model after turn n, where t_n is the first word and t_{n_T} is the last word of the sentence. Similarly, we could formally turn the intuition into optimization problem as follows:

$$L(\hat{\mathbf{x}}) = \max\{[Z_o(x_{0\dots T})]_{o'} - \max_{o \in \bar{O}} [Z_o(x_{0\dots T})]_o, -\kappa\} \tag{3.12}$$

Since we do not need to force target agent to say end-of-dialogue, the problem becomes an unconstrained discrete optimization problem. We then directly apply the unconstrained version of greedy algorithm (YCH18) to solve it.

3.2.3.4 Force target agent to select at arbitrary turn

While we could let our adversarial agent and the target agent negotiate n turns, it is still unknown which n should be chosen to get the best performance. In other words, we aim to not only know what to say but also when to say to fool the target agent.

We propose two strategies to force target agent to make bad decisions at arbitrary turn. The details are presented in Algorithm 7. When it is the turn for adversarial agent to speak, we first try to apply reactive and preemptive attacks. If both attacks couldn't make the loss $L(\cdot)$ less than 0, there are two strategies: 1) just output the sentence generated by the regular speaking module (delayed attack), and 2) conduct transfer attack. The comparisons can be found in the experiments.

Algorithm 7 Arbitrary turn attack algorithm

Input: Target agent B, Input goal g
Output: Dialogue $x_{0...T}$, Agent score S_{adv} and S_{ori}
while $\langle selection \rangle$ is not generated **do**
  Set the loss $L(\cdot)$ to be (3.11)
  Optimize the Loss $L(\cdot)$
  if $L(\cdot) < 0$ **then**
    Add the output into the dialogue
  else
    Set the loss $L(\cdot)$ in to be (3.12)
    Optimize the Loss $L(\cdot)$
    if $L(\cdot) < 0$ **then**
      Add the output into the dialogue
    else
      if Transfer Attack **then**
        Randomly add a sentence in L (malicious sentences) into the dialogue.
      else
        Add the sentence generated by regular speaking model into the dialogue (delayed attack).
Generate o using dialogue $x_{0...T}$
Calculate S_{adv} and S_{ori}
Return: $x_{0...T}$,S_{adv},S_{ori}

3.2.4 Adversarial Training

Adversarial training is a popular method to improve the robustness of machine learning models (MDG16; MMS18). In this section, we use the agents designed in the previous sections to improve the robustness of the target agent.

In standard adversarial training for neural network models (GSS14; JL17), adversarial examples (images or sentences) generated by an attack are added to the training procedure to refine the model. Since our setting is interactive and there is no fixed data used in selfplay, we should conduct training with *adversarial agents* instead of adversarial examples. Moreover, as pointed out by (JL17), training on the examples generated by a single attack will lead to over-fitting to a particular attack, so we should do adversarial training iteratively.

Taking the black-box RL agent as an example, we consider the following min-max

formulation:

$$\min_{\theta^{ori}}\{\max_{\theta^{adv}} S_{adv} - S_{ori}\}, \tag{3.13}$$

where θ^{ori} is the weights for the target agent and θ^{adv} is the weights for the adversarial black-box agent. We solve (3.13) by the following alternating minimization procedure. At each iteration, we first update the target agent (θ^{ori}) using the standard policy gradient algorithm, and then use our RL algorithm in Section 3.2.2.1 to update adversarial agent to counter the target model. We iteratively conduct these updates until convergence. The experiments show that the adversarial training procedure can improve the robustness not only under RL attack but also under other white-box attacks.

3.2.5 Experimental Results

We perform extensive experiments on evaluating the robustness of the negotiation agents developed in (LYD17). Furthermore, we show that the robustness of negotiation agents can be significantly improved using the proposed adversarial training procedure. Our codes are publicly available at `https://github.com/cmhcbb/Robustness-of-Dialogue-systems`.

Experimental Setup We use the code released by the authors (LYD17) and follow their instructions to get the target end-to-end negotiation agents. More specifically, we first train the model on 5808 dialogues, based on 2236 unique scenarios in supervised way to imitate the actions of human users. We call this model supervised model (SV agent). Then we use reinforcement learning to conduct goal-oriented training in order to maximize the agent' reward. The second model is called the reinforcement learning model (RL agent). As a result, when doing selfplay between RL agent and SV agent, we could get RL agent with 5.86 perplexity, 89.57% agreement and 7.23 average score, while SV agent achieves 5.47 perplexity and 4.55 average score. These numbers are similar to the numbers reported in (LYD17).

To evaluate the robustness of these agents, we conduct all the proposed attacks on both supervised model (SV agent) and reinforcement learning model (RL agent). The successfulness

of an attack is measured by average score advantage and positive advantage rate (PAR). Average score advantage is defined by averaged adversarial agent's score minus average target agent's score. The value is in the region of $[-10, 10]$ since the total values are controlled to be 10 for both sides, and a larger advantage indicates a more successful attack. Also, we define positive advantage rate (PAR) as the ratio of dialogues that the adversarial agent gets a higher score than the target agent. We will see that most attacks developed in this chapter will improve both average score advantage and PAR. Note that this is the first work on attacking a goal-oriented dialogue agent so there is no previous method that could be included in the comparisons.

Results on Black-box Attacks As introduced in Section 4, we have two black-box attacks: reinforcement learning attack (RL attack) and Transfer attack.

RL Attack. In the reinforcement learning attack, we use a learning rate of 0.1, clip gradients above 1.0, and set the discount factor $\gamma = 0.95$ in (3.8). We train the adversarial agent for 4 epochs on all scenarios. From Table 3.10, we observe that with 100% agreement rate, our adversarial agent could gain 2.32 score advantage against the RL agent and 4.25 advantage against the SV agent. Also, our agent achieves a relatively high positive advantage rate at 84.45% and 69.35% respectively. We show some adversarial dialogues played by adversarial agent and target agent in Table 3.11. It shows that RL agent is able to identify the weak point of target agent by saying "take book you get rest", which could easily let the agent accept the deal and make a bad selection that is inconsistent with the context of dialogue.

Transfer attack. In transfer attack, we first let our adversarial agent speak the sentence generated by the speaking model with target agent for 3 turns. If the end-of-dialog token has never been mentioned, in the 4th turn, the adversarial agent speaks the sentence generated by our RL agent. The detailed results are shown in Table 3.10. We observe that the transfer

attack is not successful—only -0.13 and -1.189 score advantage are achieved. We found that transferring sentences between dialogues is not successful because the item values and conversation histories are quite different between dialogues.

	vs SV agent			vs RL agent		
Model	**PAR%**	**Score(advantage)**	**Agreement%**	**PAR%**	**Score(advantage)**	**Agreement%**
RL model(w/o attack)	75.79	7.23 vs 4.55 (2.68)	89.57	44.70	5.05 vs 5.00 (0.05)	76.36
Transfer attack	44.43	6.41 vs 6.54 (-0.13)	100	36.10	5.65 vs 6.84 (-1.19)	100
RL attack	84.45	8.28 vs 4.03 (4.25)	100	69.35	7.11 vs 4.79 (2.32)	100
Reactive attack	87.00	8.83 vs 3.43 (5.40)	100	90.23	8.72 vs 3.77 (4.95)	100
Preemptive attack	71.86	7.76 vs 4.95 (2.81)	100	69.23	6.78 vs 6.01 (0.77)	100
RA+PA+DA	84.33	8.79 vs 2.96 (5.83)	100	86.93	8.73 vs 2.95 (5.78)	100
RA+PA+TA	83.12	8.67 vs 3.05 (5.62)	100	89.74	8.62 vs 2.92 (5.70)	100

Table 3.10: Negotiation task evaluation with different adversarial agent on 2000 randomly generated scenarios, against the supervised model and reinforcement learning model. The maximum socre is 10. When agents failed to agree, all agents get 0 score. PAR stands for positive advantage rate. RA+PA+DA stands for the combination of reactive attack, preemptive attacka and delayed attack. RA+PA+TA stands for the combination of reactive attack, preemptive attacka and transfer attack.

Input		
Adv agent	1x**book** value 1 4x**hat** value 1 1x**ball** value 5	
RL agent	1x**book** value 2 4x**hat** value 1 1x**ball** value 4	
Adv agent	i want the hats and 2 balls	
RL agent	i need the balls and the hat	
Adv agent	take book you get rest	
RL agent	deal	
Adv agent	⟨*selection*⟩	
Output		**Reward**
Adv agent	4x**hat** 1x**ball**	9/10
RL agent	1x**book**	2/10

Table 3.11: Dialogue example generated by black-box RL attack agent against RL agent.

Results on White-box Attacks We conduct the white-box attacks introduced in Section 3.2.3.

Force target agent to select at a fixed turn. There are two types of algorithms (reactive attack and preemptive attack) introduced in Section 3.2.3.1. The detailed results

are shown in Table 3.10. We observe that the reactive attack could achieve better results than black-box method with 5.40 score advantage against SV agent and 4.98 score advantage against RL agent. On the other hand, preemptive attack is not that successful—it gets 2.81 advantage against SV agent and 0.77 score advantage against RL agent. Furthermore, we have included some adversarial dialogues played by white-box adversarial agent and target agent in Table 3.12. From these examples, we could see that white-box adversarial agent could generate the adversarial sentences, slightly unnatural however still readable, that could fool the target agent to make terrible decisions.

Input		
Adv agent	1x**book** value 0 1x**hat** value 1 3x**ball** value 3	
RL agent	1x**book** value 1 1x**hat** value 0 3x**ball** value 3	
Adv agent	i would like the balls and the hat	
RL agent	i need the balls and the book	
Adv agent	i need the balls and fine book	
RL agent	⟨*selection*⟩	
Output		**Reward**
Adv agent	1x**hat** 1x**book** 3x**ball**	10/10
RL agent		0/10

Table 3.12: Dialogue example generated by reactive attack agent against RL agent.

Input		
Adv agent	1x**book** value 4 2x**hat** value 1 2x**ball** value 2	
RL agent	1x**book** value 8 2x**hat** value 0 2x**ball** value 1	
RL agent	i would like the book and the hat.	
Adv agent	i want reasonable balls and book	
RL agent	⟨*selection*⟩	
Output		**Reward**
Adv agent	1x**book** 2x**ball**	8/10
RL agent	2x**hat**	0/10

Table 3.13: Dialogue example generated by RA+PA+DA attack agent against RL agent.

Force target agent to select at arbitrary turn. To determine when should we begin the attack, we design combinations of reactive attack, preemptive attack and transfer attack or

delayed attack in Section 3.2.3.4. Here, we conduct experiments to validate the effectiveness of these two attack combinations. From Table 3.10, the combinations achieve better results than all the previous attacks. The best result is achieved by the combination of reactive attack, preemptive attack and delayed attack (RA+PA+DA), which gets 5.83 advantage against SV agent and 5.78 score advantage against RL agent, with very high positive advantage rates at 84.33% and 86.93% respectively. We have included some adversarial dialogues played by this adversarial agent and the target agent in Table 3.13. We observe that with the delayed attack, the adversarial agent can decide **when to attack**, thus achieves much better performance than attacking at a fixed turn.

3.2.5.1 Adversarial Training

Using the algorithm proposed in Section 6, we conduct adversarial training using the black-box RL attack model. The results are shown in Table 3.14. First, we observe that the adversarial trained model achieves much better performance against black-box RL attack; the advantage of RL attack drops from 2.32 to -1.8. Moreover, the model achieves consistently better performance against other white-box attacks. For instance, the advantage of the strongest RA+PA+DA attack is reduced from 5.78 to 3.98.

	vs advtrain model		
Model	**PAR%**	**Score(advantage)**	**Agreement%**
RL model(w/o attack)	48.67	6.51 vs 6.64 (-0.13)	91.75
Transfer attack	23.05	4.93 vs 7.59 (-2.66)	100
RL attack	62.61	5.71 vs 7.51 (-1.80)	100
Reactive attack	80.76	8.83 vs 4.31 (4.52)	100
Preemptive attack	34.39	5.64 vs 7.41 (-1.77)	100
RA+PA+DA	73.96	8.05 vs 4.07 (3.98)	100
RA+PA+TA	73.45	8.06 vs 4.13 (3.93)	100

Table 3.14: Negotiation task evaluation with different adversarial agent on 2000 randomly generated scenarios, against adversarial trained model.

3.2.6 Analysis and Discussions

RL agents are more robust than SV agents. From Table 3.10, we could see that all the attack methods perform better when facing SV agents than RL agents. It is because that SV agents only learn to mimic human's action and are trained only on human data. Therefore, it is reasonable that RL agents are more robust than SV agents.

The importance of arbitrary turns. In reactive attack and preemptive attack, we begin our attack at the n-th turn and we set $n = 2$ in the experiments. Here we show the results with different n in Table 3.15. We observe that the performance of white-box attacks are quite consistent with different choices of n. This probably indicates that there the best n varies for different cases. Therefore, if we could change the n from case to case adaptively, which is done by delayed attack, we could see a performance boost.

n	**PAR%**	**Score(advantage)**	**Agreement%**
1	94.02	8.84 vs 3.32 (5.52)	100
2	90.23	8.72 vs 3.77 (4.95)	100
3	92.02	8.81 vs 3.62 (5.19)	100
4	90.35	8.71 vs 3.87 (4.84)	100

Table 3.15: Negotiation task evaluation with different choices of n against RL model.

Adversarial training helps to improve the robustness. We then try to investigate the robustness of the adversarial trained model. We found that in the original model, it is easy for an attacker to find a sentence to quickly end the dialogue. However, after adversarial training, it becomes much harder to find such sentences. Moreover, although we only conduct adversarial training on black-box RL model, the adversarial trained model still achieves better performance against other white-box attacks. This indicates that the adversarial trained model could probably detect the slight unnaturalness of those sentences and thus have a better reading comprehension ability.

Part II

Adversarial defenses

In this part, we introduce the mainstream defense methods to improve the model's adversarial robustness. This part is composed by two chapters: **Attack-dependent defense** and **Attack-independent defense**. In Chapter 4, we start with the widely used attack-specific adversarial training in Section 4.1 and its limitations. To counter the aforementioned limitations, we then introduce CAT in Section 4.2 which adaptively customizes the perturbation level and the corresponding label for each training sample in adversarial training. In Chapter 5, We then briefly cover other attack-independent robust training in Section 5.1. Then we summarize the aforementioned method into a general framework inspired by vicinal risk minimization in Section 5.2. In Section 5.3, we introduce a new defense method called SPROUT and show its effectiveness from different perspectives. Although many algorithms have been proposed to improve the robustness of machine learning models, all of them sacrifice performance on natural data. In Chapter 6, we talk about this issue and introduce a hypothesis in Section 6.2 that Batch Normalization, one of the widely used techniques in convolution neural networks, has a great impact on the robustness trade-off. Following the hypothesis, we introduce RobMask in Section 6.3 to improve the model's generalization to improve the model's performance over both natural data and adversarial examples.

CHAPTER 4

Attack-dependent Robust Training

4.1 Adversarial Training

(SZS13) shows model robustness could improve when augmented with adversarial examples. However, it is limited by its usage of expensive L-BFGS method to generate adversarial examples. (GSS14) proposes a more efficient attack method FGSM that could significantly improve the model robustness. Later, (MMS18) finds L_∞ based PGD could achieve the best performance that generates a 92.76% accuracy with $\epsilon = 0.3$ constraint in MNIST dataset. Specifically, adversarial training can be formulated as a min-max optimization problem. For a K-class classification problem, let $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1,\dots,n}$ denote the set of training samples in the dataset with $\boldsymbol{x}_i \in \mathbb{R}^d, y_i \in \{1, \dots, K\} =: [K]$. Let $f_\theta(\boldsymbol{x}) : \mathbb{R}^d \to [K]$ denote a classification model parameterized by θ. We denote by $h_\theta(\boldsymbol{x}) : \mathbb{R}^d \to [0,1]^K$ as the prediction output for each class, i.e., $f_\theta(\boldsymbol{x}) = \text{argmax}_i[h_\theta(\boldsymbol{x})]_i$. We use standard $O(\cdot)$ notation to hide universal constant factor, and $a \lesssim b$ to indicate $a = O(b)$.

Adversarial training can be formulated as:

$$\min_\theta \frac{1}{n} \sum_{i=1}^{n} \max_{\boldsymbol{x}_i' \in \mathcal{B}(\boldsymbol{x}_i, \epsilon)} \ell(f_\theta(\boldsymbol{x}_i'), y_i), \tag{4.1}$$

where $\mathcal{B}(\boldsymbol{x}_i, \epsilon)$ denotes the ℓ_p-norm ball centered at $\boldsymbol{x}_i$ with radius ϵ. The inner maximization problem aims to find an adversarial version of a given data point $\boldsymbol{x}_i$ that achieves a high loss. In general one can define $\mathcal{B}(\boldsymbol{x}_i, \epsilon)$ based on the threat model, but the ℓ_∞ ball is the most popular choice adopted by recent works (MMS18; ZYJ19; Wan19), which will also be used in this section. For a deep neural network model, the inner maximization does not

have a closed form solution, so adversarial training methods typically use a gradient-based iterative solver to approximately solve the inner problem. The most commonly used choice is the multi-step PGD (MMS18) and C&W attack (CW17). It has since inspired many advanced adversarial training algorithms with improved robustness. For instance, TRADES (ZYJ19) is designed to minimize a theoretically-driven upper bound on prediction error of adversarial examples, which led to the first-ranked defense in the NeurIPS 2018 Adversarial Vision Challenge. Bilateral adversarial training (Wan19) finds robust models by adversarially perturbing the data samples and as well as the associated data labels. A feature-scattering based adversarial training method is proposed in (ZW19). Another line of recent works uses an adversarially trained model along with additional unlabeled data (CRS19; SFK19) or self-supervised learning with adversarial examples (HMK19) to improve robustness.

4.1.1 Limitation

Intuitively, if adversarial training can always find a model with close-to-zero robust error, one should always use a large ϵ for training because it will automatically imply robustness to any smaller ϵ. Unfortunately, in practice a uniformly large ϵ is often harmful. In the following we empirically explain this problem and use it to motivate our proposed algorithm.

We use a simple linear classification case to demonstrate why a uniformly large ϵ is harmful. In Figure 4.1a, we generate a synthetic linearly separable dataset with the margin set to be 1.75 for both classes, where the correct linear boundary can be easily obtained by standard training. In Figure 4.1b, we run adversarial training with $\epsilon = 1$, and since this ϵ is smaller than the margin, the algorithm can still obtain near-optimal results. However, when we use a large $\epsilon = 4$ for adversarial training in Figure 4.1c, the resulting decision boundary becomes significantly worse. It is because adversarial training cannot correctly fit all the samples with a margin up to 4, so it will sacrifice some data samples, leading to distorted and undesirable decision boundary. This motivates the following two problems:

- We shouldn't set the same large ϵ uniformly for all samples. Some samples are intrinsi-

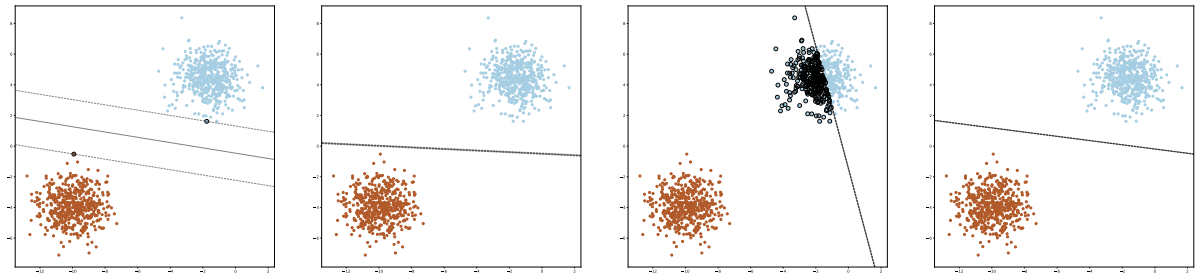
cally closer to the decision boundary and they should use a smaller ϵ. Without doing this, adversarial training will give up on those samples, which leads to worse training and generalization error (see more discussions in Section 4.2.3 on the generalization bounds).

- The adversarial training loss is trying to force the prediction to match the one-hot label (e.g., $[1, 0]$ in the binary classification case) even after large perturbations. However, if a sample is perturbed, the prediction shouldn't remain one-hot. For instance, if a sample is perturbed to the decision boundary of a binary classification problem, the prediction of a perfect model should be $[0.5, 0.5]$ instead of $[1, 0]$, which also makes adversarial training fail to recover a good decision hyperplane.

Furthermore, we observe that even if adversarial training can obtain close-to-zero training error with large ϵ (e.g., (GCL19) proves that this will happen for overparameterized network with large-enough margin), a uniformly large ϵ will lead to larger generalization gap. This could be partially explained by the theoretical results provided by (YRB18), which shows that the adversarial Rademacher complexity has a lower bound with an explicit dependence on the perturbation tolerance. The empirical results in Table 4.1 also illustrate this problem. When conducting adversarial training with $\epsilon = 0.3$ on CIFAR10 VGG-16, we found that the model achieves close-to-zero robust training error on all $\epsilon \leq 0.3$, but it suffers larger generalization gap compared to training with smaller ϵ. This also demonstrates that a uniformly large ϵ is harmful even when it achieves perfect training error.

Table 4.1: The influence of different fixed ϵ values used in adversarial training on the robust accuracy with $\epsilon = 0.01$.

Testing ϵ	Error Type	Training ϵ		
		0.01	0.02	0.03
0.01	Train	99.96%	99.99%	99.16%
	Test	69.79%	69.06%	66.04%



(a) Standard training (b) Adv-train with $\epsilon = 1$ (c) Adv-train with $\epsilon = 4$ (d) CAT (ours) with $\epsilon_{max} = 4$

Figure 4.1: Different training methods on a linearly separable binary classification dataset with 1.75 margin for both classes. Adversarial training with small ϵ works fine, but for a large ϵ beyond the true margin, adversarial training would ruin the classifier's classification performance, while our proposed adaptive customized adversarial training method still keeps a good generalization performance.

4.2 CAT: Customized Adversarial Training for Improved Robustness

In this section, we propose the Customized Adversarial Training (CAT) framework that improves adversarial training by addressing the above-mentioned problems. First, our algorithm has an auto-tuning method to customize the ϵ used for each training example. Second, instead of forcing the model to fit the original label, we customize the target label for each example based on its own ϵ. In the following we will describe these two components in more detail.

4.2.1 Auto-tuning Perturbation Strength for Adversarial Training

The first component of our algorithm is an ϵ auto-tuning method which adaptively assigns a suitable ϵ for each example during the adversarial training procedure. Let ϵ_i be the perturbation level assigned to example i. Based on the intuition mentioned in Section 4.1.1, we do not want to further increase ϵ if we find the classifier does not have capacity to robustly

classify the example, which means we should set

$$\epsilon_i = \operatorname*{argmin}_{\epsilon}\{\max_{\boldsymbol{x}_i' \in \mathcal{B}_p(\boldsymbol{x}_i, \epsilon)} f_\theta(\boldsymbol{x}_i') \neq y_i\} \tag{4.2}$$

and the adversarial training objective becomes

$$\min_\theta \frac{1}{n} \sum_{i=1}^{n} \max_{\boldsymbol{x}_i' \in \mathcal{B}_p(\boldsymbol{x}_i, \epsilon_i)} \ell(f_\theta(\boldsymbol{x}_i'), y_i). \tag{4.3}$$

Note that ϵ_i in (4.2) depends on θ, while θ in (4.3) also depends on ϵ_i. We thus propose an alternative update scheme — conducting one SGD update on θ, and then updating the ϵ_i in the current batch. However, finding ϵ_i exactly requires brute-force search for every possible value, which adds significant computational overhead to adversarial training.

Therefore, we only conduct a simplified update rule on ϵ_i as follows. Starting from an initial perturbation level of zero, at each iteration we conduct adversarial attack (e.g., PGD attack) with perturbation tolerance $\epsilon_i + \eta$ where η is a constant. If the attack is successful, then we reset current ϵ_i to 0 to encourage model learning a more robust classifier towards those examples. While if the attack is unsuccessful, which means an attacker still cannot find an adversarial example that satisfies $\max_{\boldsymbol{x}_i' \in \mathcal{B}_p(\boldsymbol{x}_i, \epsilon_i + \eta)} f_\theta(\boldsymbol{x}_i') \neq y_i$, then we increase $\epsilon_i = \epsilon_i + \eta$. The attack results will also be used to update the model parameter θ, so this adaptive scheme does not require any additional cost. In practice, we also have an upper bound on the final perturbation to ensure that ϵ_i remains bounded for each i.

4.2.2 Adaptive Label Uncertainty for Adversarial Training

As mentioned in Section 4.1.1, the standard adversarial training loss is trying to enforce a sample being classified as the original one-hot label after ϵ perturbation. However, this may not be ideal. In the extreme case, if a sample is perturbed to the decision boundary, the prediction must be far away from one-hot. This problem is more severe when using non-uniform ϵ_i, since each different ϵ_i will introduce a different bias to the loss, and that may be one of the reasons that purely adaptive ϵ-scheduling does not work well (see our ablation study in Section 4.2.4 and also the results reported in (BGH19)).

In the following, we propose an adaptive label smoothing approach to reflect different perturbation tolerance on each example. (SVI16) introduced label smoothing that converts one-hot label vectors into one-warm vectors representing low-confidence classification, in order to prevent the model from making over-confident predictions. Specifically, with a one-hot encoded label y, the smoothed version is

$$\tilde{y} = (1-\alpha)y + \alpha u,$$

where $\alpha \in [0,1]$ is the hyperparameter to control the smoothing level. In the adaptive setting, we set $\alpha = c\epsilon_i$ so that a larger perturbation tolerance would receive a higher label uncertainty and c is a hyperparameter. A common choice of u is $u = \frac{1}{K}$. However, this strict requirement tries to enforce every other labels having the same probability, which may not make sense in practice. On the other hand, as shown Section 4.1.1, adversarial training is easy to overfit and generate a large generalization gap. To better address these issues, we sample from a distribution instead. Specifically, we use $u = \text{Dirichlet}(\boldsymbol{\beta})$ where $\text{Dirichlet}(\cdot)$ refers to the Dirichlet distribution and $\boldsymbol{\beta} \in \mathbb{R}^K$ is concentration hyperparamter. With different perturbation tolerance, the adaptive version of label smoothing is

$$\tilde{y_i} = (1 - c\epsilon_i)y_i + c\epsilon_i \text{Dirichlet}(\boldsymbol{\beta}). \tag{4.4}$$

The final objective function: Combining the two aforementioned techniques, our Customized Adversarial Training (CAT) method attempts to minimize the following objective:

$$\begin{aligned} \min_{\theta} \quad & \frac{1}{n}\sum_{i=1}^{n} \max_{\boldsymbol{x}_i' \in \mathcal{B}_p(\boldsymbol{x}_i, \epsilon_i)} \ell(f_\theta(\boldsymbol{x}_i'), \tilde{y_i}) \\ \textbf{s.t.} \ & \epsilon_i = \operatorname*{argmin}_{\epsilon} \{ \max_{\boldsymbol{x}_i' \in \mathcal{B}_p(\boldsymbol{x}_i, \epsilon)} f_\theta(\boldsymbol{x}_i') \neq y_i \}, \end{aligned} \tag{4.5}$$

where $\tilde{y_i}$ is defined in (4.4). As described in Section 4.2.1, we approximately minimize this objective with an alternative update scheme, which incurs almost no additional cost compared to the original adversarial training algorithm. The detailed algorithm is presented in Algorithm 8.

Choice of loss function. In general, our framework can be used with any loss function $\ell(\cdot)$. In the previous works, cross entropy loss is commonly used for ℓ. However, the model

trained by smoothing techniques tends to have a smaller logit gap between true label and other labels. Therefore, in order to encourage model to generate a larger logit gap, we propose a mixed loss to enhance the defense performance towards C&W$_\infty$ attack. That is,

$$\mathrm{CE}(f_\theta(\boldsymbol{x}_i'), \tilde{y}_i) + \max\{\max_{j \neq y_0}\{[Z(\boldsymbol{x}_i')]_j - [Z(\boldsymbol{x}_i')]_{y_0}\}, -\kappa\}, \tag{4.6}$$

where $Z(\boldsymbol{x}) \in \mathbb{R}^K$ is the final (logit) layer output, and $[Z(\boldsymbol{x})]_i$ is the prediction score for the i-th class and y_0 is the original label. The parameter κ encourages the adversary to find higher confident adversarial examples in training.

Algorithm 8 CAT algorithm

Input: Training dataset (X, Y), cross entropy loss or mix loss ℓ, scheduling parameter η, weighting factor c, perturbation upperbound ϵ_{max}
Initial every sample's ϵ_i with 0
for epoch=$1, \ldots, N$ **do**
  for i=$1, \ldots, B$ **do**
    $\tilde{y}_i \leftarrow (1 - c\epsilon_i)y_i + c\epsilon_i \mathrm{Dirichlet}(\boldsymbol{\beta})$
    $\epsilon_i \leftarrow \epsilon_i + \eta$
    $\delta_i \leftarrow 0$
    for $j = 1 \ldots m$ **do**
      $\delta_i \leftarrow \delta_i + \alpha \cdot sign(\nabla_\delta \ell(f_\theta(\boldsymbol{x}_i + \delta_i), \tilde{y}_i)$
      $\delta_i \leftarrow \max(\min(\delta_i, \epsilon_i), -\epsilon_i)$
    if $f_\theta(\boldsymbol{x}_i + \delta_i) \neq y_i$ **then**
      $\epsilon_i \leftarrow 0$
    $\epsilon_i \leftarrow \min(\epsilon_{max}, \epsilon_i)$
    $\tilde{y}_i \leftarrow (1 - c\epsilon_i)y_i + (1 - c\epsilon_i)\mathrm{Dirichlet}(\boldsymbol{\beta})$
    $\theta \leftarrow \theta - \gamma_\theta \nabla_\theta \ell(f_\theta(\boldsymbol{x}_i + \delta_i), \tilde{y}_i)$
return θ

4.2.3 Theoretical Analysis

To better understand how our scheme improves generalization, we now provide some theoretical analysis. Recall we denote by $h_\theta(\boldsymbol{x}) : \mathbb{R}^d \to [0, 1]^K$ as the prediction probability for the K classes. We define the bilateral margin that our paper is essentially maximizing over as follows.

Definition 1 (Bilateral margin)**.** *We define the bilateral perturbed network output by*

$H_\theta(\boldsymbol{x}, \boldsymbol{\delta}^i, \boldsymbol{\delta}^o)$:

$$H_\theta(\boldsymbol{x}, \boldsymbol{\delta}^i, \boldsymbol{\delta}^o) := h_\theta\left(\boldsymbol{x} + \boldsymbol{\delta}^i \|\boldsymbol{x}\|\right) + \left\|\boldsymbol{x} + \boldsymbol{\delta}^i \|\boldsymbol{x}\|\right\| \cdot \boldsymbol{\delta}^o.$$

The bilateral margin is now defined as the minimum norm of $(\boldsymbol{\delta}^i, \boldsymbol{\delta}^o)$ *required to cause the classifier to make false predictions:*

$$\begin{aligned} m_F(\boldsymbol{x}, y) &:= \min_{\boldsymbol{\delta}^i, \boldsymbol{\delta}^o} \sqrt{\|\boldsymbol{\delta}^i\|^2 + \|\boldsymbol{\delta}^o\|^2} \\ s.t. \quad & \max_{y'} H_\theta(\boldsymbol{x}, \boldsymbol{\delta}^i, \boldsymbol{\delta}^o)_{y'} \neq y. \end{aligned} \tag{4.7}$$

This margin captures both the relative perturbation on the input layer $\boldsymbol{\delta}^i$ and the soft-max output $\boldsymbol{\delta}^o$.

Theorem 4. *Suppose the parameter space* Θ *we optimize over has covering number that scales as* $\log \mathcal{N}_{\|\cdot\|_{op}}(\eta, \Theta) \leq \lfloor \mathcal{C}^2/\eta^2 \rfloor$ *for some complexity* $\mathcal{C}$*. Then with probability* $1 - \delta$ *over the draw of the training data, any classifer* $f_\theta, \theta \in \Theta$ *which achieves training error zero satisfies:*

$$\mathbb{E}[f_\theta(\boldsymbol{x}) = y] \lesssim \frac{\mathcal{C} \log^2 n}{\sqrt{n}} \sqrt{\frac{1}{n} \sum_{i=1}^{n} \frac{1}{m_F(\boldsymbol{x}_i, y_i)}} + \zeta,$$

where ζ *is of small order* $O\left(\frac{1}{n} \log(1/\delta)\right)$.

We defer the proof later, which is adapted from Theorem 2.1 of (WM19). We observe the population risk is bounded by two key factors, the average of $\frac{1}{m_F(\boldsymbol{x}_i, y_i)}$ and $\mathcal{C}$, the covering number of the parameter space. On one side, the average of $\frac{1}{m_F(\boldsymbol{x}_i, y_i)}$ is dominated by the samples with the smallest margin. Therefore when we do adversarial training, it is important that we not only achieve higher overall accuracy, but also make sure the samples closer to the decision boundary have large enough margin. This can not be achieved by simply using constant and large ϵ that will maintain a large margin for most samples but sacrifice the accuracy of a small portion of data. On the other hand, the covering number of the network's parameter space can be roughly captured by a bound of product of all layers' weight norms. We hypothesize that with more flexibility in choosing ϵ, our algorithm will converge faster than using larger constant ϵ and will have more implicit regularization effect. To testify this

Table 4.2: The clean and robust accuracy of VGG-16 models trained by various defense methods. All robust accuracy results use $\epsilon = 8/255$ ℓ_∞ ball. $^{(X)}$ denotes using a X-step PGD attack. X random denotes X times random restart.

Methods	No attack	Deepfool	PGD^{100}	C&W^{100}	20 PGD^{1000}	20 C&W^{1000}
Natural train	**93.34%**	16.39%	0.6%	0.0%	0.0%	0.0%
Adv train (MMS18)	80.32%	44.65%	36.36%	37.89%	36.12%	36.8%
TRADES (ZYJ19)	84.85%	48.37%	38.81%	39.49%	37.95%	38.94%
CAT (ours)	85.44%	**70.19%**	**75.54%**	**51.81%**	**75.17%**	**50.08%**

hypothesis, we roughly measure the model complexity $\mathcal{C}$ by the product of the weight norms of different models. In comparison to our model, when training with constant $\epsilon = 0.01, 0.02$ and 0.03, it respectively yields $\mathcal{C}$ as large as $2.54, 3.53$ and 1.39 times of that of our model, which means our model indeed has more implicit regularization effect among others.

4.2.4 Experimental Results

In this section, we conduct extensive experiments to show that CAT achieves a strong result on both clean and robust accuracy. We include the following methods into our comparison:

- Customized Adversarial Training (CAT): Our proposed method.
- Adversarial training: The adversarial training method proposed in (MMS18) where they use a K-step PGD attack as adversary.
- TRADES: TRADES (ZYJ19) improves adversarial training by an additional loss on the clean examples and achieves the state-of-art performance on robust accuracy.
- Natural: the natural training which only minimizes the cross entropy loss.

Furthermore, since many recently proposed adversarial training methods have considered CIFAR-10 with Wide-ResNet structure as the standard setting for reporting their numbers, we also compare our performance with 7 previous methods on this specific setting.

Table 4.3: The clean and robust accuracy of Wide Resnet models trained by various defense methods. All robust accuracy results use $\epsilon = 8/255$ ℓ_∞ ball. We reported the best performance listed in the papers. $^{(*)}$ denotes random-restart is applied in the testing attack. $^{(X)}$ denotes using a X-step PGD attack. $\times$ denotes not reported.

Methods	Clean accuracy	PGD accuracy	C&W accuracy
Natural training	**95.93%**	0%	0%
Adversarial training (MMS18)	87.30%	52.68%	50.73%
Dynamic adversarial training (WMB19)	84.51%	55.03%	51.98%
TRADES (ZYJ19)	84.22%	$56.40\%^{(20)}$	51.98%
Bilateral Adv Training (Wan19)	91.00%	$57.5\%^{(*20)}$	$56.2\%^{(*20)}$
MMA (DSL18)	84.36%	47.18%	$\times$
MART (WZY19)	84.17%	$58.56\%^{(20)}$	54.58%
IAAT (BGH19)	91.34%	$48.53\%^{(*10)}$	56.80%
CAT (ours)	89.61%	$73.16\%^{(*20)}$	$\mathbf{71.67\%}^{(*20)}$

4.2.4.1 Experimental Setup

Dataset and model structure. We use two popular dataset CIFAR-10 (KH09) and Restricted-ImageNet (DDS09) for performance evaluation. For CIFAR-10, we use both standard VGG-16 (SZ15) and Wide ResNet that is used in both vanilla adversarial training (MMS18) and TRADES (ZYJ19). For VGG-16, we implement adversarial training with the standard hyper-parameters and train TRADES with the official implementation. For Wide ResNet, since the model has become standard for testing adversarial training methods, we use exactly the same model structure provided by (MMS18; ZYJ19). We use the models' checkpoint released by TRADES official repository and implement the Madry's adversarial training using the standard hyper-parameters. For Restricted-ImageNet, we use ResNet-50. All our experiments were implemented in Pytorch-1.4.

Implementation details. We set the number of iterations in adversarial attack to be 10 for all methods during training. Adversarial training and TRADES are trained on PGD attacks setting $\epsilon = 8/255$ with cross entropy loss (CE). All the models are trained using SGD with momentum 0.9, weight decay 5×10^{-4}. For VGG-16/Wide ResNet models, we use the

initial learning rate of 0.01/0.1, and we decay the learning rate by 90% at the 80th, 140th, and 180th epoch. For CAT, we set epsilon scheduling parameter $\eta = 0.005$, $\epsilon_{max} = 8/255$ and weighting parameter $c = 10$. We set $\boldsymbol{\beta} = \mathbf{1}$ for the distribution Dirichlet($\boldsymbol{\beta}$), which is equal to a uniform distribution. Also, we set $\kappa = 10$.

4.2.4.2 Robustness Evaluation and Analysis

White-box attacks results. For CIFAR10, we evaluate all the models under white-box $\epsilon = 8/255$ ℓ_∞-norm bounded non-targeted PGD and C&W attack. Specifically, we use both PGD^X (X-step PGD with step size $\epsilon/5$) and C&W$_\infty$. As suggested, we test our model under different steps PGD and multiple random restarts.

The experimental results are shown in Table 4.2, where we can easily see that CAT clearly outperforms other methods. CAT achieves a significant better robust accuracy at the standard 8/255 perturbation threshold considered in the literature, and also have better clean accuracy. We also test the performance of CAT under attacks with 20 restarts and 1,000 iterations to confirm the robustness of the model. Futhermore, we visualize the loss landscape and perform PGD attack with different strength.

Wide-ResNet has become a standard structure for comparing adversarial training methods, and it's standard to train and evaluate with 8/255 ℓ_∞ norm perturbation. For this setting, we collect the reported accuracy from 7 other adversarial training methods, with several of them published very recently, to have a detailed full comparison. As shown in Table 4.3, our method achieves state-of-art robust accuracy while maintaining a high clean accuracy. Due to the page limit, we put the Restricted ImageNet result.

Black-box transfer attacks results. We follow the criterion of evaluating transfer attacks as suggested by (ACW18) to inspect whether the models trained by CAT will cause the issue of obfuscated gradients and give a false sense of model robustness. We generate 10,000 adversarial examples of CIFAR-10 from natural models with $\epsilon = 8/255$ and evaluate

their attack performance on the target model. Table 4.4 shows that CAT achieves the best accuracy compared with adversarial training and TRADES, suggesting the effectiveness of CAT in defending both white-box and transfer attacks.

Table 4.4: Robust accuracy under transfer attack on CIFAR-10

Method	VGG 16	Wide ResNet
Adv train	79.13%	85.84%
TRADES	83.53%	83.90%
CAT	**86.58%**	**88.66 %**

Restricted Imagenet Result In addition to CIFAR, we also test the performance on the Restricted Imagenet dataset, which has been used in previous papers such as (SSK19). This dataset consists of a subset of imagenet classes which have been grouped into 9 different classes. The experimental results are shown in Table 5. All the results except our methods are reported in (SSK19). Similarity, we could see CAT have a clear performance boost over listed methods.

Table 4.5: The clean and robust accuracy of Resnet50 models trained by various defense methods. All robust accuracy are measured under $\epsilon = 8/255$ ℓ_∞ ball. We reported the best performance listed in the papers. $^{(*)}$ denotes random-restart is applied in the testing attack. $^{(X)}$ denotes it use a X-step PGD attack

Methods	Clean accuracy	PGD accuracy	C&W accuracy
Adversarial training (MMS18)	91.83%	17.52%	×
FAT (SSK19)	**91.59 %**	18.81%	×
LAT (SSK19)	89.86 %	22.00%	×
CAT (ours)	88.63%	$58.4\%^{(*20)}$	$58.4\%^{(*20)}$

4.2.4.3 Ablation study

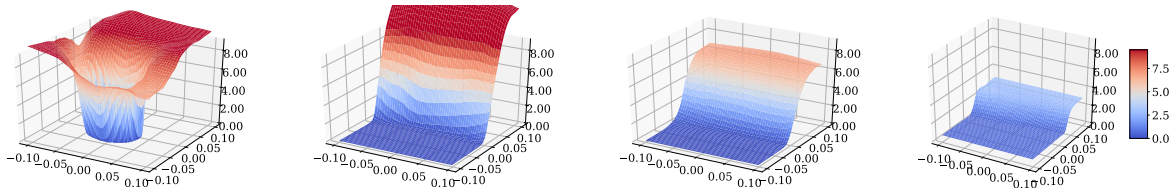
The importance of adaptive label uncertainty. Here we discuss and perform an ablation study using VGG-16 and CIFAR-10 on the importance of adaptive label uncertainty and

Table 4.6: Ablation study on CAT by changing the loss function and removing Label Adaption (LA). All robust accuracy results use $\epsilon = 8/255$ ℓ_∞ ball.

Methods	Clean acc	PGD acc
Adv train	80.32%	36.63%
Adv+LS	80.25%	43.0%
Adp-Adv	87.91%	38.59%
CAT	**84.22%**	**75.54%**

adaptive instance-wise ϵ. In Table 4.6, Adv train denotes the original adversarial training, Adv+LS denotes adversarial training with label smoothing (setting y by Eq (4.4)), Adp-Adv denotes adversarial training with adaptive instance-wise ϵ, and CAT is the proposed method which is a combination of these two tricks. We found that only applying adaptive instance-wise ϵ or label smoothing cannot significantly boost the robust accuracy over standard adversarial training, but the proposed method, by nicely combining these two ideas, can significantly improve the performance. This explains why CAT significantly outperforms some instance adaptive ϵ methods like IAAT and MMA.

Loss Landscape Exploration To further verify the superior robustness using CAT,



(a) Natural (b) Adv train (c) TRADES (d) CAT

Figure 4.2: CAT: Loss landscape comparison of different adversarial training methods

we visualize the loss landscape of different training methods in Figure 4.2. Following the implementation in (EIA18), we divide the data input along a linear space grid defined by the sign of the input gradient and a random Rademacher vector, where the x- and y- axes represent the magnitude of the perturbation added in each direction and the z-axis represents the loss. As shown in Figure 4.2, CAT generates a model with a lower and smoother loss landscape. Also, it could be taken as another strong evidence that we have found a robust

model through CAT training.

4.2.5 Proofs

4.2.5.1 Proof of Theorem 4

In this section we provide the omitted proof for Theorem 4, which is adapted from Theorem 2.1 from (WM19). They defined the all layer margin for a k-layer network $h_\theta(\boldsymbol{x}) = f_k \circ f_{k-1} \circ \cdots f_1(\boldsymbol{x})$ and perturbation $\delta = (\boldsymbol{\delta}_1, \boldsymbol{\delta}_2, \cdots \boldsymbol{\delta}_k)$ as follows:

$$\begin{aligned} h_1(\boldsymbol{x}, \delta) =& f_1(\boldsymbol{x}) + \boldsymbol{\delta}_1 \|\boldsymbol{x}\|_2 \\ h_i(\boldsymbol{x}, \delta) =& f_i(h_{i-1}(\boldsymbol{x}, \boldsymbol{\delta})) + \boldsymbol{\delta}_i \|h_{i-1}(\boldsymbol{x}, \delta)\|_2 \\ H_\theta(\boldsymbol{x}, \delta) =& h_k(\boldsymbol{x}, \delta). \end{aligned}$$

They define the all-layer margin as the minimum norm of $\delta = (\boldsymbol{\delta}_i)_{i=1}^k$ required that causes the classifier to make a false prediction.

$$\begin{aligned} m_F(\boldsymbol{x}, y) :=& \min_{\boldsymbol{\delta}^i, \boldsymbol{\delta}^o} \sqrt{\|\boldsymbol{\delta}^i\|^2 + \|\boldsymbol{\delta}^o\|^2} \\ & \text{subject to } \max_{y'} H_\theta(\boldsymbol{x}, \boldsymbol{\delta}^i, \boldsymbol{\delta}^o)_{y'} \neq y. \end{aligned} \tag{4.8}$$

They consider the function class $\mathcal{F} = \{f_k \circ f_{k-1} \cdots \circ f_1 : f_i \in \mathcal{F}_i\}$ be the class of compositions of functions from function classes $\mathcal{F}_1, \cdots \mathcal{F}_k$. They achieve the generalization bound as follows:

Theorem 5 (Theorem 2.1 from (WM19))**.** *In the above setting, with probability* $1 - \delta$ *over the draw of the data, all classifiers* $F \in \mathcal{F}$ *which achieve training error* 0 *satisfy*

$$\mathbb{E}[f_\theta(\boldsymbol{x}) = y] \lesssim \frac{\sum_i \mathcal{C}_i \log^2 n}{\sqrt{n}} \sqrt{\frac{1}{n} \sum_{i=1}^n \frac{1}{m_F(\boldsymbol{x}_i, y_i)}} + \zeta,$$

where ζ *is of small order* $O\left(\frac{1}{n} \log(1/\delta)\right)$.

For our problem, we define $h_\theta(\boldsymbol{x}) := f_2 \circ f_1(\boldsymbol{x})$, where f_1 is identity mapping, and f_2 is

the original function h_θ. Therefore the all layer margin is reduced to our bilateral margin:

$$\begin{aligned} h_1(\boldsymbol{x}, \boldsymbol{\delta}^i) =& f_1(\boldsymbol{x}) + \boldsymbol{\delta}^i \|\boldsymbol{x}\|_2 = \boldsymbol{x} + \boldsymbol{\delta}^i \|\boldsymbol{x}\| \\ H_\theta(\boldsymbol{x}, \boldsymbol{\delta}) = h_2(\boldsymbol{x}, \boldsymbol{\delta}^i, \boldsymbol{\delta}^o) =& f_2(h_1(\boldsymbol{x}, \boldsymbol{\delta}^i)) + \boldsymbol{\delta}^o \|h_1(\boldsymbol{x}, \boldsymbol{\delta}^i)\| \\ =& h_\theta(\boldsymbol{x} + \boldsymbol{\delta}^i \|\boldsymbol{x}\|) + \boldsymbol{\delta}^o \|\boldsymbol{x} + \boldsymbol{\delta}^i \|\boldsymbol{x}\|\|. \end{aligned}$$

Next, notice since f_1 is identity mapping, and composition with h_θ doesn't affect the overall complexity. We apply Theorem 5 and get our result.

CHAPTER 5

Attack-independent Robust Training

5.1 Introduction

Here we first discuss some attack-independent robust training methods on Gaussian data augmentation, Mixup and label smoothing. Gaussian data augmentation during training is a commonly used baseline method to improve model robustness (ZNR17). It is revisited in (CRK19) as a scalable and certifiable defense method called random smoothing. Mixup (ZCD18) and its variants (VLB18; TCB19) are a recently proposed approach to improve model robustness and generalization by training a model on convex combinations of data sample pairs and their labels. Label smoothing was originally proposed in (SVI16) as a regularizer to stabilize model training. The main idea is to replace one-hot encoded labels by assigning non-zero (e.g., uniform) weights to every label other than the original training label. Although label smoothing is also shown to benefit model robustness (SGH19; GD19), its robustness gain is relatively marginal when compared to adversarial training. In contrast to currently used static (i.e., pre-defined) label smoothing functions, in SPROUT we propose a novel parametrized label smoothing scheme, which enables adaptive sampling of training labels from a parameterized distribution on the label simplex. The parameters of the label distribution are progressively adjusted according to the updates of model weights.

5.2 General Framework for Formulating Robust Training

The task of supervised learning is essentially learning a K-class classification function $f \in \mathcal{F}$ that has a desirable mapping between a data sample $\boldsymbol{x} \in \mathcal{X}$ and the corresponding label $\boldsymbol{y} \in \mathcal{Y}$. Consider a loss function L that penalizes the difference between the prediction $f(\boldsymbol{x})$ and the true label $\boldsymbol{y}$ from an unknown data distribution P, $(\boldsymbol{x}, \boldsymbol{y}) \sim P$. The population risk can be expressed as

$$R(f) = \int L(f(\boldsymbol{x}), \boldsymbol{y}) P(\boldsymbol{x}, \boldsymbol{y}) d\boldsymbol{x} d\boldsymbol{y} \tag{5.1}$$

However, as the distribution P is unknown, in practice machine learning uses empirical risk minimization (ERM) with the empirical data distribution of n training data $\{x_i, y_i\}_{i=1}^n$

$$P_\delta(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{n} \sum_{i=1}^{n} \delta(\boldsymbol{x} = \boldsymbol{x}_i, \boldsymbol{y} = \boldsymbol{y}_i) \tag{5.2}$$

to approximate $P(\boldsymbol{x}, \boldsymbol{y})$, where δ is a Dirac mass. Notably, a more principled approach is to use Vicinity Risk Minimization (VRM) (CWB01), defined as

$$P_\nu(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{n} \sum_{i=1}^{n} \nu(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}} | \boldsymbol{x}_i, \boldsymbol{y}_i) \tag{5.3}$$

where ν is a vicinity distribution that measures the probability of finding the virtual sample-label pair $(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}})$ in the vicinity of the training pair $(\boldsymbol{x}_i, \boldsymbol{y}_i)$. Therefore, ERM can be viewed as a special case of VRM when $\nu = \delta$. VRM has also been used to motivate Mixup training (ZCD18). Based on VRM, we propose a general framework that encompasses the objectives of many robust training methods as the following generalized cross entropy loss:

$$H(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}}, f) = -\sum_{k=1}^{K} [\log g(f(\tilde{\boldsymbol{x}})_k)] h(\tilde{y}_k) \tag{5.4}$$

where $f(\tilde{\boldsymbol{x}})_k$ is the model's k-th class prediction probability on $\tilde{\boldsymbol{x}}$, $g(\cdot) : \mathbb{R} \to \mathbb{R}$ is a mapping adjusting the probability output, and $h(\cdot) : \mathbb{R} \to \mathbb{R}$ is a mapping adjusting the training label distribution. When $\tilde{\boldsymbol{x}} = \boldsymbol{x}$, $\tilde{\boldsymbol{y}} = \boldsymbol{y}$ and $g = h = \mathcal{I}$, where $\mathcal{I}$ denotes the identity mapping function, the loss in (5.4) degenerates to the conventional cross entropy loss.

Based on the general VRM loss formulation in (5.4), in Table 5.1 we summarize a large body of robust training methods in terms of different expressions of $g(\cdot)$, $h(\cdot)$ and $(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}})$.

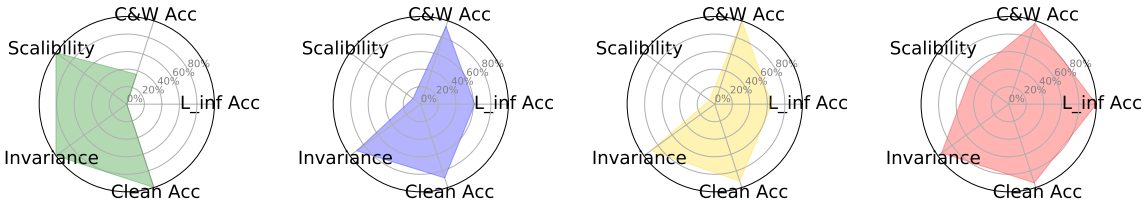
Table 5.1: Summary of robust training methods using VRM formulation in (5.4). $\text{PGD}_\epsilon(\cdot)$ means (multi-step) PGD attack with perturbation budget ϵ. Dirichlet($\mathbf{b}$) is the Dirichlet distribution parameterized by $\mathbf{b}$. GA/LS stands for Gaussian-Augmentation/Label-Smoothing.

Methods	$g(\cdot)$	$h(\cdot)$	$\tilde{\boldsymbol{x}}$	$\tilde{\boldsymbol{y}}$	attack-specific
Natural	$\mathcal{I}$	$\mathcal{I}$	$\boldsymbol{x}$	$\boldsymbol{y}$	$\times$
GA (ZNR17)	$\mathcal{I}$	$\mathcal{I}$	$\mathcal{N}(\boldsymbol{x}, \Delta^2)$	$\boldsymbol{y}$	$\times$
LS (SVI16)	$\mathcal{I}$	$(1-\alpha)\boldsymbol{y} + \alpha\boldsymbol{u}$	$\boldsymbol{x}$	$\boldsymbol{y}$	$\times$
Adversarial training (MMS18)	$\mathcal{I}$	$\mathcal{I}$	$\text{PGD}_\epsilon(\boldsymbol{x})$	$\boldsymbol{y}$	✓
TRADES (ZYJ19)	$\mathcal{I}$	$(1-\alpha)\boldsymbol{y} + \alpha f(\tilde{\boldsymbol{x}})$	$\text{PGD}_\epsilon(\boldsymbol{x})$	$\boldsymbol{y}$	✓
Stable training (ZSL16)	$f(\boldsymbol{x}) \circ f(\tilde{\boldsymbol{x}})$	$\mathcal{I}$	$\mathcal{N}(\boldsymbol{x}, \Delta^2)$	$\boldsymbol{y}$	$\times$
Mixup (ZCD18)	$\mathcal{I}$	$\mathcal{I}$	$(1-\lambda)\boldsymbol{x}_i + \lambda\boldsymbol{x}_j$	$(1-\lambda)\boldsymbol{y}_i + \lambda\boldsymbol{y}_j$	$\times$
LS+GA (SGH19)	$\mathcal{I}$	$(1-\alpha)\boldsymbol{y} + \alpha\boldsymbol{u}$	$\mathcal{N}(\boldsymbol{x}, \Delta^2)$	$\boldsymbol{y}$	$\times$
Bilateral Adv Training (Wan19)	$\mathcal{I}$	$\mathcal{I}$	$\text{PGD}_\epsilon(\boldsymbol{x})$ (one or two step)	$(1-\alpha)\boldsymbol{y}_i + \alpha\text{PGD}_{\epsilon'}(\boldsymbol{y})$	✓
SPROUT (ours)	$\mathcal{I}$	Dirichlet$((1-\alpha)\boldsymbol{y} + \alpha\boldsymbol{\beta})$	$(1-\lambda)\mathcal{N}(\boldsymbol{x}_i, \Delta^2) + \lambda\mathcal{N}(\boldsymbol{x}_j, \Delta^2)$	$(1-\lambda)\boldsymbol{y}_i + \lambda\boldsymbol{y}_j$	$\times$

For example, the vanilla adversarial training in (MMS18) aims to minimize the loss of adversarial examples generated by the (multi-step) PGD attack with perturbation budget ϵ, denoted by $\text{PGD}_\epsilon(\cdot)$. Its training objective can be rewritten as $\tilde{\boldsymbol{x}} = \text{PGD}_\epsilon(\boldsymbol{x})$, $\tilde{\boldsymbol{y}} = \boldsymbol{y}$ and $g = h = \mathcal{I}$. In addition to adversarial training only on perturbed samples of $\boldsymbol{x}$, (Wan19) designs adversarial label perturbation where it uses $\tilde{\boldsymbol{x}} = \text{PGD}_\epsilon(\boldsymbol{x})$, $\tilde{\boldsymbol{y}} = (1-\alpha)\boldsymbol{y} + \alpha\text{PGD}_\epsilon(\boldsymbol{y})$, and $\alpha \in [0, 1]$ is a mixing parameter. TRADES (ZYJ19) improves adversarial training with an additional regularization on the clean examples, which is equivalent to replacing the label mapping function $h(\cdot)$ from identity to $(1-\alpha)\boldsymbol{y} + \alpha f(\tilde{\boldsymbol{x}})$. Label smoothing (LS) alone is equivalent to the setup that $g = \mathcal{I}$, $\tilde{\boldsymbol{x}} = \boldsymbol{x}$, $\tilde{\boldsymbol{y}} = \boldsymbol{y}$ and $h(\cdot) = (1-\alpha)\boldsymbol{y} + \alpha\boldsymbol{u}$, where $\boldsymbol{u}$ is often set as a uniform vector with value $1/K$ for a K-class supervised learning task. Joint training with Gaussian augmentation (GA) and label smoothing (LS) as studied in (SGH19) is equivalent to the case when $\tilde{\boldsymbol{x}} = \mathcal{N}(\boldsymbol{x}, \Delta^2), \tilde{\boldsymbol{y}} = \boldsymbol{y}$, $g = \mathcal{I}$ and $h(\boldsymbol{y}) = (1-\alpha)\boldsymbol{y} + \alpha/K$. We defer the connection between SPROUT and VRM to the next section.

5.3 SPROUT: Scalable Robust and Generalizable Training

In this section, we formally introduce SPROUT, a novel robust training method that automatically finds a better vicinal risk function during training in a self-progressing manner.



(a) Natural (b) Adversarial training (c) TRADES (d) SPROUT (ours)

Figure 5.1: Multi-dimensional performance comparison of four training methods using VGG-16 network and CIFAR-10 dataset. All dimensions are separately normalized by the best-performance method. The average score of each method is 0.6718 for natural (standard training), 0.6900 for PGD-ℓ_∞ based adversarial training (MMS18), 0.7107 for PGD-ℓ_∞ based TRADES (ZYJ19), and 0.8798 for SPROUT (ours). The exact numbers are reported in Section 5.3.4.7.

State-of-the-art robust training algorithms are primarily based on the methodology of adversarial training (GSS14; MMS18), which calls specific attacking algorithms to generate adversarial examples during model training for learning robust models. Albeit effective, these methods have the following limitations: (i) *poor scalability* – the process of generating adversarial examples incurs considerable computation overhead. For instance, our experiments show that, with the same computation resources, standard adversarial training (with 7 attack iterations per sample in every minibatch) of Wide ResNet on CIFAR-10 consumes 10 times more clock time per training epoch when compared with standard training; (ii) *attack specificity* – adversarially trained models are usually most effective against the same attack they trained on, and the robustness may not generalize well to other types of attacks (TB19; KSH19); (iii) *preference toward wider network* – adversarial training is more effective when the networks have sufficient capacity (e.g., having more neurons in network layers) (MMS18).

5.3.1 Self-Progressing Parametrized Label Smoothing

To stabilize training and improve model generalization, (SVI16) introduces label smoothing that converts "one-hot" label vectors into "one-warm" vectors representing low-confidence

classification, in order to prevent a model from making over-confident predictions. Specifically, the one-hot encoded label $\boldsymbol{y}$ is smoothed using

$$\tilde{\boldsymbol{y}} = (1-\alpha)\boldsymbol{y} + \alpha\boldsymbol{u} \tag{5.5}$$

where $\alpha \in [0,1]$ is the smoothing parameter. A common choice is the uniform distribution $\boldsymbol{u} = \frac{1}{K}$, where K is the number of classes. Later works (Wan19; GD19) use an attack-driven label smoothing function $\boldsymbol{u}$ to further improve adversarial robustness. However, both uniform and attack-driven label smoothing disregard the inherent correlation between labels. To address the label correlation, we propose to use the Dirichlet distribution parametrized by $\boldsymbol{\beta} \in \mathbb{R}_+^K$ for label smoothing. Our SPROUT learns to update $\boldsymbol{\beta}$ to find a training label distribution that is most uncertain to a given model θ, by solving

$$\max_{\boldsymbol{\beta}} L(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}}, \boldsymbol{\beta}; \theta) \tag{5.6}$$

where $\tilde{\boldsymbol{y}} = \text{Dirichlet}((1-\alpha)\boldsymbol{y} + \alpha\boldsymbol{\beta})$. Notably, instead of using a pre-defined or attack-driven function for $\boldsymbol{u}$ in label smoothing, our Dirichlet approach automatically finds a label simplex by optimizing $\boldsymbol{\beta}$. Dirichlet distribution indeed takes label correlation into consideration as its generated label $\boldsymbol{z} = [z_1, \ldots, z_K]$ has the statistical properties

$$\mathbb{E}[z_s] = \frac{\beta_s}{\beta_0},\ \text{Cov}[z_s, z_t] = \frac{-\beta_s\beta_t}{\beta_0^2(\beta_0+1)},\ \sum_{s=1}^{K} z_s = 1 \tag{5.7}$$

where $\beta_0 = \sum_{k=1}^{K} \beta_k$ and $s, t \in \{1, \ldots, K\}$, $s \neq t$. Moreover, one-hot label and uniform label smoothing are our special cases when $\boldsymbol{\beta} = \boldsymbol{y}$ and $\boldsymbol{\beta} = \boldsymbol{u}$, respectively. Our Dirichlet label smoothing co-trains with the update in model weights θ during training (see Algorithm 9). The advantage of our proposed self-progressing Dirichlet label smoothing over uniform label smoothing will be justified in our ablation study (see Figure 5.5 in Section 5.3.4.6). In addition, we illustrate the label correlation learned from our Dirichlet label smoothing in Section 5.3.4.8.

5.3.2 Gaussian Data Augmentation and Mixup

Gaussian augmentation. Adding Gaussian noise to data samples during training is a common practice to improve model robustness. Its corresponding vicinal function is the Gaussian vicinity function $\nu(\tilde{\boldsymbol{x}}_i, \tilde{\boldsymbol{y}}_i|\boldsymbol{x}_i, \boldsymbol{y}_i) = \mathcal{N}(\boldsymbol{x}_i, \Delta^2)\delta(\tilde{\boldsymbol{y}}_i = \boldsymbol{y}_i)$, where Δ^2 is the variance of a standard normal random vector. However, the gain of Gaussian augmentation in robustness is marginal when compared with adversarial training (see our ablation study in Section 5.3.4.6). (SGH19) finds that combining uniform or attack-driven label smoothing with Gaussian smoothing can further improve adversarial robustness. Therefore, we propose to incorporate Gaussian augmentaion with Dirichlet label smoothing. The joint vicinity function becomes $\nu(\tilde{\boldsymbol{x}}_i, \tilde{\boldsymbol{y}}_i|\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\beta}) = \mathcal{N}(\boldsymbol{x}_i, \Delta^2)\delta(\tilde{\boldsymbol{y}}_i = \text{Dirichlet}((1-\alpha)\boldsymbol{y}_i + \alpha\boldsymbol{\beta}))$. Training with this vicinity function means drawing labels from the Dirichlet distribution for the original data sample $\boldsymbol{x}_i$ and its neighborhood characterized by Gaussian augmentation.

Mixup. To further improve model generalization, SPROUT also integrates Mixup (ZCD18) that performs convex combination on pairs of training data samples (in a minibatch) and their labels during training. The vicinity function of Mixup is $\nu(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}}|\boldsymbol{x}_i, \boldsymbol{y}_i) = \delta(\tilde{\boldsymbol{x}} = (1-\lambda)\boldsymbol{x}_i + \lambda\boldsymbol{x}_j, \tilde{\boldsymbol{y}} = (1-\lambda)\boldsymbol{y}_i + \lambda\boldsymbol{y}_j)$, where $\lambda \sim \text{Beta}(a, a)$ is the mixing parameter drawn from the Beta distribution and $a > 0$ is the shape parameter. The Mixup vicinity function can be generalized to multiple data sample pairs. Unlike Gaussian augmentation which is irrespective of the label (i.e., only adding noise to $\boldsymbol{x}_i$), Mixup aims to augment data samples on the line segments of training data pairs and assign them convexly combined labels during training.

Vicinity function of SPROUT. With the aforementioned techniques integrated in SPROUT, the overall vicinity function of SPROUT can be summarized as $\nu(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}}|\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\beta}) = \delta(\tilde{\boldsymbol{x}} = \lambda\mathcal{N}(\boldsymbol{x}_i, \Delta^2) + (1-\lambda)\mathcal{N}(\boldsymbol{x}_j, \Delta^2),\ \tilde{\boldsymbol{y}} = \text{Dirichlet}((1-\alpha)((1-\lambda)\boldsymbol{y}_i + \lambda\boldsymbol{y}_j) + \alpha\boldsymbol{\beta})$.

In Section 5.3.4.6, we will show that Dirichlet label smoothing, Gaussian augmentation and Mixup are complimentary to enhancing robustness by showing their diversity in input

Algorithm 9 SPROUT algorithm

Input: Training dataset (X, Y), Mixup parameter λ, Gaussian augmentation variance Δ^2, model learning rate γ_θ, Dirichlet label smoothing learning rate $\gamma_{\boldsymbol{\beta}}$ and parameter α, generalized cross entropy loss L
Initial model θ: random initialization (train from scratch) or pre-trained model checkpoint
Initial $\boldsymbol{\beta}$: random initialization
for epoch=$1, \ldots, N$ **do**
  for minibatch $X_B \subset X, Y_B \subset Y$ **do**
    $X_B \leftarrow \mathcal{N}(X_B, \Delta^2)$
    $X_{mix}, Y_{mix} \leftarrow \text{Mixup}(X_B, Y_B, \lambda)$
    $Y_{mix} \leftarrow \text{Dirichlet}(\alpha Y_{mix} + (1-\alpha)\boldsymbol{\beta})$
    $g_\theta \leftarrow \nabla_\theta L(X_{mix}, Y_{mix}, \theta)$
    $g_{\boldsymbol{\beta}} \leftarrow \nabla_{\boldsymbol{\beta}} L(X_{mix}, Y_{mix}, \theta)$
    $\theta \leftarrow \theta - \gamma_\theta g_\theta$
    $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} + \gamma_{\boldsymbol{\beta}} g_{\boldsymbol{\beta}}$
return θ

gradients.

5.3.3 SPROUT Algorithm

Using the VRM framework, the training objective of SPROUT is

$$\min_{\theta} \max_{\boldsymbol{\beta}} \sum_{i=1}^{n} L(\nu(\tilde{\boldsymbol{x}}_i, \tilde{\boldsymbol{y}}_i | \boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\beta}); \theta), \tag{5.8}$$

where θ denotes the model weights, n is the number of training data, L is the generalized cross entropy loss defined in (5.4) and $\nu(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}} | \boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\beta})$ is the vicinity function of SPROUT. Our SPROUT algorithm co-trains θ and $\boldsymbol{\beta}$ via stochastic gradient descent/ascent to solve the outer minimization problem on θ and the inner maximization problem on $\boldsymbol{\beta}$. In particular, for calculating the gradient $g_{\boldsymbol{\beta}}$ of the parameter $\boldsymbol{\beta}$, we use the Pytorch implementation based on (FMM18). SPROUT can either train a model from scratch with randomly initialized θ or strengthen a pre-trained model. As shown in Section 5.3.4.2, we find that training from either randomly initialized or pre-trained natural models using SPROUT can yield substantially robust models that are resilient to large perturbations. The training steps of SPROUT are summarized in Algorithm 9.

We also note that our min-max training methodology is different from the min-max formulation in adversarial training (MMS18), which is $\min_\theta \sum_{i=1}^{n} \max_{\boldsymbol{\delta}_i:\|\boldsymbol{\delta}_i\|_p \leq \epsilon} L(\boldsymbol{x}_i + \boldsymbol{\delta}_i, \boldsymbol{y}_i; \theta)$, where $\|\boldsymbol{\delta}_i\|_p$ denotes the ℓ_p norm of the adversarial perturbation $\boldsymbol{\delta}_i$. While the outer minimization step for optimizing θ can be identical, the inner maximization of adversarial training requires running multi-step PGD attack to find adversarial perturbations $\{\boldsymbol{\delta}_i\}$ for each data sample in every minibatch (iteration) and epoch, which is attack-specific and time-consuming (see our scalability analysis in Table 4.6). On the other hand, our inner maximization is upon the Dirichlet parameter $\boldsymbol{\beta}$, which is attack-independent and only requires single-step stochastic gradient ascent with a minibatch to update $\boldsymbol{\beta}$. We have explored multi-step stochastic gradient ascent on $\boldsymbol{\beta}$ and found no significant performance enhancement but increased computation time.

Advantages of SPROUT. Comparing to adversarial training, the training of SPROUT is more efficient and scalable, as it only requires one additional back propagation to update $\boldsymbol{\beta}$ in each iteration (see Table 4.6 for a run-time analysis). As highlighted in Figure 5.1, SPROUT is also more comprehensive as it automatically improves robustness in multiple dimensions owing to its self-progressing training methodology. Moreover, we find that SPROUT significantly outperforms adversarial training and attains larger gain in robustness as network width increases (see Figure 5.7), which makes SPROUT a promising approach to support robust training for a much larger set of network architectures.

5.3.4 Experimental Results

5.3.4.1 Experiment Setup

Dataset and network structure. We use CIFAR-10 (KH09) and ImageNet (DDS09) for performance evaluation. For CIFAR-10, we use both standard VGG-16 (SZ15) and Wide ResNet, where the latter is used in both vanilla adversarial training (MMS18) and TRADES (ZYJ19). The Wide ResNet models are pre-trained PGD-ℓ_∞ robust models given

by adversarial training and TRADES. For VGG-16, we implement adversarial training with the standard hyper-parameters and train TRADES using the official implementation. For ImageNet, we use ResNet-152. All our experiments were implemented in Pytorch-1.2 and conducted using dual Intel E5-2640 v4 CPUs (2.40GHz) with 512 GB memory with a GTX 1080 GPU.

Implementation details. As suggested in Mixup (ZCD18), we set the Beta distribution parameter $a = 0.2$ when sampling the mixing parameter λ. For Gaussian augmentation, we set $\Delta = 0.1$, which is within the suggested range in (ZNR17). Also, we set the label smoothing parameter $\alpha = 0.01$. A parameter sensitivity analysis on λ and α is given in Section 5.3.4.9. Unless specified otherwise, for SPROUT we set the model initialization to be a natural model. An ablation study of model initialization is given in Section 5.3.4.6.

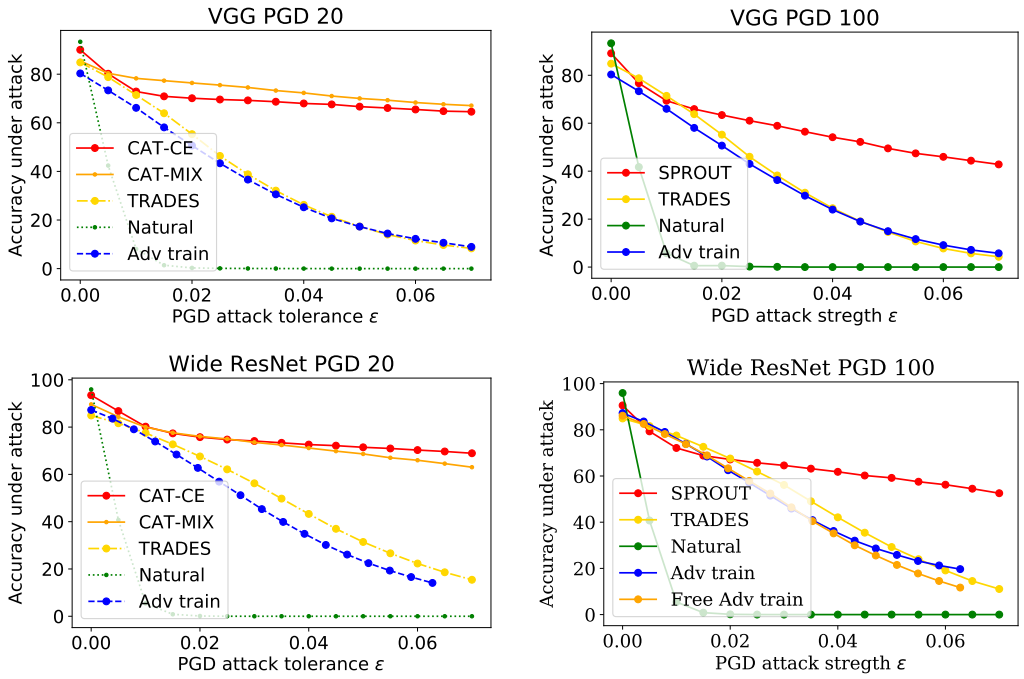


Figure 5.2: Robust accuracy of CIFAR-10 under PGD-ℓ_∞ attack. SPROUT significantly outperforms other methods.

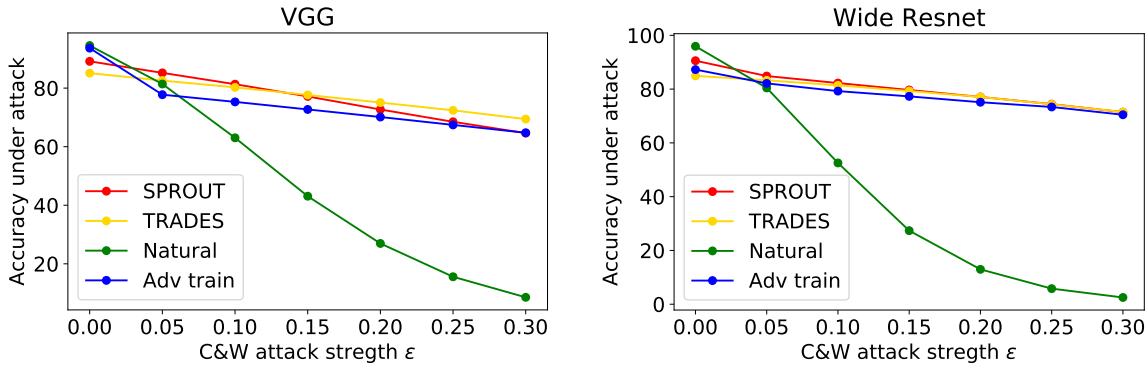


Figure 5.3: Robust accuracy of CIFAR-10 under C&W-ℓ_2 attack

5.3.4.2 Adversarial Robustness under Various Attacks

White-box attacks. On CIFAR-10, we compare the model accuracy under different strength of white-box ℓ_∞-norm bounded non-targeted PGD attack, which is considered as the strongest first-order adversary (MMS18) with an ℓ_∞-norm constraint ϵ (normalized between 0 to 1). All PGD attacks are implemented with random starts and we run PGD attack with 20 and 100 steps in our experiments. The (robust) accuracy under different ϵ values are shown in Figure 5.2. When $\epsilon = 0.03$ and under PGD attack with 20 steps, we find SPROUT achieves 62.24% and 66.23% robust accuracy on VGG16 and Wide ResNet respectively, while TRADES and adversarial training are 10-20% worse than SPROUT. We also find that SPROUT is significantly more robust to PGD-ℓ_∞ attacks with large ϵ values. In addition to the substantially improved robustness, the clean accuracy (i.e., when $\epsilon = 0$) of SPROUT is 5-10 % higher than TRADES and adversarial training, and it is only 2-4% lower than natural model, suggesting SPROUT better balances the robustness-accuracy trade-off. Similar trends are observed in robust accuracy under PGD attack with 100 steps. On Wide ResNet we also report the robust accuracy of the "free adversarial training" (Free Adv Train) method (SNG19), which features similar robust accuracy as adversarial training but can reduce training time.

We also compare the robust accuracy against PGD-ℓ_∞ attacks with multiple random

Table 5.2: Robust accuracy of CIFAR-10 under transfer attack

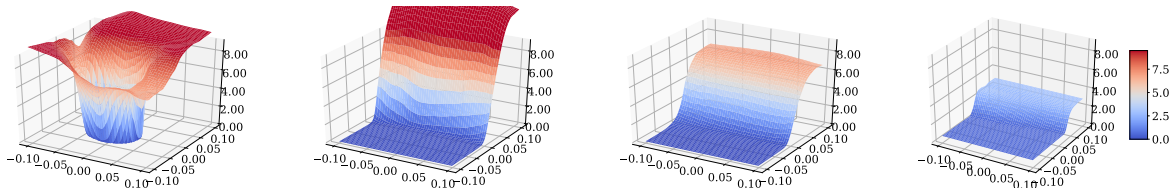
Method	VGG 16	Wide ResNet
Adv Train	79.13%	85.84%
TRADES	83.53%	83.9%
SPROUT	86.28%	89.1%

Table 5.3: Accuracy of ImageNet under PGD-ℓ_∞ attack

Method	Clean Acc	$\epsilon = 0.005$	$\epsilon = 0.01$	$\epsilon = 0.015$	$\epsilon = 0.02$
Natural	78.31%	37.13%	9.14%	2.12%	0.78%
SPROUT	74.23%	65.24%	52.86%	35.04%	12.18%

starts. The results are consistent with the vallina PGD-ℓ_∞ attack, i.e., SPROUT attains the highest accuracy (see Section 5.3.4.10). Moreover, we report the results of C&W-ℓ_∞ attack (CW17) in Section 5.3.4.11, where SPROUT again shows high robust accuracy for large ϵ values. Next, we compare against ℓ_2-norm based C&W attack by using the default attack setting with 10 binary search steps and 1000 iterations per step to find successful perturbations while minimizing their ℓ_2-norm. Figure 5.3 verifies that SPROUT can improve ℓ_∞ robustness by a large margin without degrading ℓ_2 robustness. SPROUT's accuracy under C&W-ℓ_2 attack is similar to TRADES and is better than both natural and adversarial training. The results also suggest that the attack-independent and self-progressing training nature of SPROUT can prevent the drawback of failing to provide comprehensive robustness to multiple and simultaneous ℓ_p-norm attacks in adversarial training (TB19; KSH19).

Transfer attack. We follow the criterion of evaluating transfer attacks in (ACW18) to inspect whether the models trained by SPROUT will cause the issue of obfuscated gradients and give a false sense of robustness. We generate 10,000 PGD-ℓ_∞ adversarial examples from CIFAR-10 natural models with $\epsilon = 0.03$ and evaluate their attack performance on the target model. Table 5.2 shows SPROUT achieves the best accuracy when compared with adversarial training and TRADES, suggesting the effectiveness of SPROUT in defending both white-box and transfer attacks.



(a) Natural (b) Adv Train (c) TRADES (d) SPROUT

Figure 5.4: SPROUT: Loss landscape comparison of different training methods

ImageNet results. As many ImageNet class labels carry similar semantic meanings, to generate meaningful adversarial examples for robustness evaluation, here we follow the same setup as in (ACW18) that adopts PGD-ℓ_∞ attacks with randomly targeted labels. Table 5.3 compares the robust accuracy of natural and SPROUT models. SPROUT greatly improves the robust accuracy across different ϵ values. For example, when $\epsilon = 0.01$, SPROUT boosts the robust accuracy of natural model by over 43%. When $\epsilon = 0.015 \approx 4/255$, a considerably large adversarial perturbation on ImageNet, SPROUT still attains about 35% robust accuracy while the natural model merely has about 2% robust accuracy. Moreover, comparing the clean accuracy, SPROUT is about 4% worse than the natural model but is substantially more robust. We omit the comparison to adversarial training methods as we are unaware of any public pre-trained robust ImageNet models of the same architecture (ResNet-152) prior to the time of our submission, and it is computationally demanding for us to train and fine-tune such large-scale networks with adversarial training. On our machine, training a natural model takes 31,158.7 seconds and training SPROUT takes 59,201.6 seconds. Comparing to the run-time analysis in Section 5.3.4.5, SPROUT has a much better scalability than adversarial training and TRADES. However, instead of ResNet-152, we use SPROUT to train the same ResNet-50 model as the pretrained Free Adv Train network and compare their performance in Section 5.3.4.12.

5.3.4.3 Loss Landscape Exploration

To further verify the superior robustness using SPROUT, we visualize the loss landscape of different training methods in Figure 5.4. Following the implementation in (EIA18), we vary the data input along a linear space defined by the sign of the input gradient and a random Rademacher vector, where the x- and y- axes represent the magnitude of the perturbation added in each direction and the z-axis represents the loss. One can observe that the loss surface of SPROUT is smoother. Furthermore, it attains smaller loss variation compared with other robust training methods. The results provide strong evidence for the capability of finding more robust models via SPROUT.

5.3.4.4 Invariance test

In addition to ℓ_p-norm bounded adversarial attacks, here we also evaluate model robustness against different kinds of input transformations using CIFAR-10 and Wide ResNet. Specifically, we change rotation (with 10 degrees), brightness (increase the brightness factor to 1.5), contrast (increase the contrast factor to 2) and make inputs into grayscale (average all RGB pixel values). The model accuracy under these invariance tests is summarized in Table 5.4. The results show that SPROUT outperforms adversarial training and TRADES. Interestingly, natural model attains the best accuracy despite the fact that it lacks adversarial robustness, suggesting a potential trade-off between accuracy in these invariance tests and ℓ_p-norm based adversarial robustness.

5.3.4.5 Scalability

As illustrated in Section 5.3.3, SPROUT enjoys great scalability over adversarial training based algorithms because its training requires much less number of back-propagations per iteration, which is a dominating factor that contributes to considerable run-time in adversarial training. Table 5.5 benchmarks the run-time of different training methods for 10 epochs.

Table 5.4: Accuracy under invariance tests

Method	Rotation	Brightness	Contrast	Gray
Natural	88.21%	93.4%	91.88 %	91.95%
Adv Train	82.66%	83.64%	84.99%	81.08%
TRADES	80.81%	81.5 %	83.08%	79.27%
SPROUT	85.95%	88.26 %	86.98%	81.64%

Table 5.5: Training-time (seconds) for 10 epochs

Methods	CIFAR-10	
	VGG 16	Wide ResNet
Natural	146.7	1449.6
Adv Train	1327.1	14246.1
TRADES	1932.5	22438.4
SPROUT	271.7	2727.8
Free Adv Train(m=8)	2053.1	20652.5

On CIFAR-10, the run-time of adverarial training and TRADES is about 5× more than SPROUT. We also report the run-time analysis using the default implementation[1] of Free Adv Train (SNG19). Its 10-epoch run-time with the replay parameter $m = 8$ is similar to TRADES. But we also note that Free Adv Train may require less number of epochs when training to convergence.

5.3.4.6 Ablation Study

Dissecting SPROUT. Here we perform an ablation study using VGG-16 and CIFAR-10 to investigate and factorize the robustness gain in SPROUT's three modules: Dirichlet label smoothing (Dirichlet), Gaussian augmentation (GA) and Mixup. We implement all combinations of these techniques and include uniform label smoothing (LS) (SVI16) as another baseline. Their accuracies under PGD-ℓ_∞ attack are shown in Figure 5.5. We highlight some important findings as follows.

• Dirichlet outperforms uniform LS by a significant factor, suggesting the importance of our

[1] `https://github.com/mahyarnajibi/FreeAdversarialTraining`

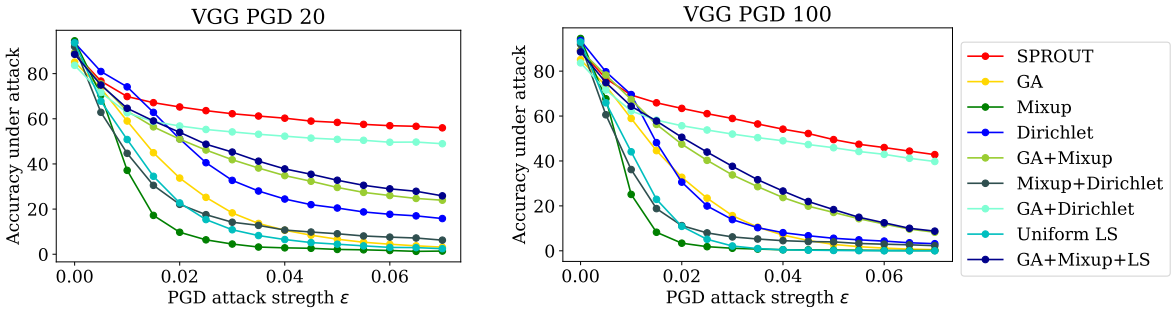


Figure 5.5: Robust accuracy with different combinations of the modules in SPROUT

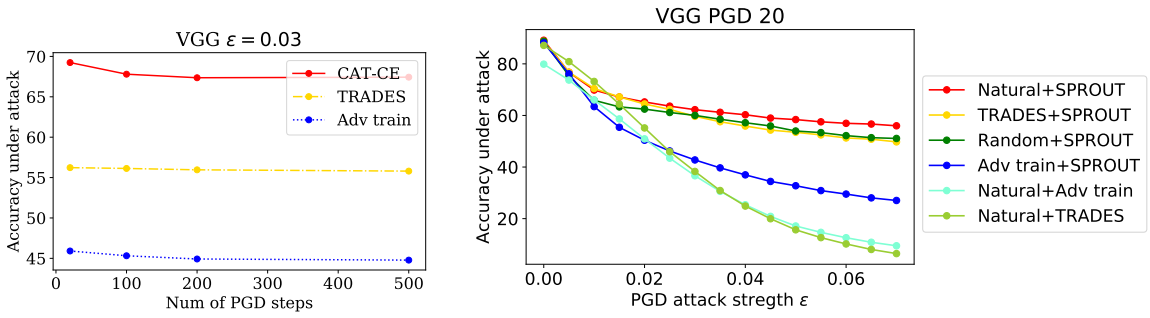
proposed self-progressing label smoothing in improving adversarial robustness.

• Comparing the performance of individual modules alone (GA, Mixup and Dirichlet), our proposed Dirichlet attains the best robust accuracy, suggesting its crucial role in training robust models.

• No other combinations can outperform SPROUT. Moreover, the robust gains from GA, Mixup and Dirichlet appear to be *complimentary*, as SPROUT's accuracy is close to the sum of their individual accuracy. To justify their diversity in robustness, we compute the cosine similarity of their pairwise input gradients and find that they are indeed quite diverse and thus can promote robustness when used together. The details are given in Section 5.3.4.13.

PGD attacks with more iterations. To ensure the robustness of SPROUT is not an artifact of running insufficient iterations in PGD attack (ACW18), Figure 5.6a shows the robust accuracy with varing number of PGD-ℓ_∞ attack steps from 10 to 500 on Wide ResNet and CIFAR-10. The results show stable performance in all training methods once the number of attack steps exceeds 100. It is clear that SPROUT indeed outperforms others by a large margin.

Model weight initialization. Figure 5.6b compares the effect of model initialization using CIFAR-10 and VGG-16 under PGD-ℓ_∞ attack, where the legend $A + B$ means using Model A as the initialization and training with Method B. Interestingly, Natural+SPROUT attains the best robust accuracy when $\epsilon \geq 0.02$. TRADES+SPROUT and Random+SPROUT



(a) (b)

Figure 5.6: Stability in PGD-ℓ_∞ attack and the effect of model initialization. Left: (a) Robust accuracy with different PGD steps. Right: (b) Robust accuracy with different model initialization.

also exhibit strong robustness since their training objective involves the loss on both clean and adversarial examples. In contrast, Adv Train+SPROUT does not have such benefit since adversarial training only aims to minimize adversarial loss. This finding is also unique to SPROUT, as neither Natural+Adv Train nor Natural+TRADES can boost robust accuracy. Our results provide novel perspectives on improving robustness and also indicate that SPROUT is indeed a new robust training method that vastly differs from adversarial training based methods.

Effect on network width. It was shown in (MMS18) that adversarial training (Adv Train) will take effect when a network has sufficient capacity, which can be achieved by increasing network width. Figure 5.7 compares the robust accuracy of SPROUT and Adv Train with varying network width on Wide ResNet and CIFAR-10. When the network has width = 1 (i.e. a standard ResNet-34 network (HZR16a)), the robust accuracy of SPROUT and Adv Train are both relatively low (less than 47%). However, as the width increases, SPROUT soon attains significantly better robust accuracy than Adv Train by a large margin (roughly 15%). Since SPROUT is more effective in boosting robust accuracy as network width varies, the results also suggest that SPROUT can better support robust training for a broader range of network structures.

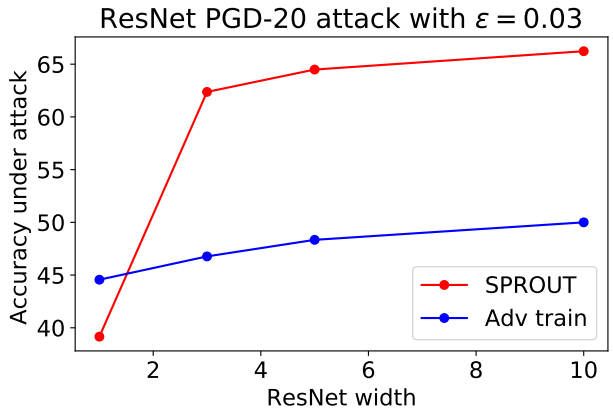


Figure 5.7: Effect of network width against PGD-ℓ_∞ attack on CIFAR-10 and ResNet-34.

Table 5.6: Performance comparison between different training methods on VGG-16 and CIFAR-10

Method	Clean Acc	ℓ_∞ Acc ($\epsilon = 0.03$)	C&W Acc	Invariance (Contrast)	Scalibility (10 epochs)
Natural	95.93%	0%	26.95%	91.88%	146.7 (secs)
Adv Train	84.92%	36.29%	70.13%	84.99%	1327.1 (secs)
TRADES	88.6%	38.29%	75.08%	83.08%	1932.5 (secs)
SPROUT	90.56%	58.93%	72.7%	86.98%	271.7 (secs)

5.3.4.7 Exact Performance Metrics for Figure 5.1

The performance metrics of Figure 5.1 are shown in Table 5.6.

5.3.4.8 Learned Label Correlation from SPROUT

Based on the statistical properties of Dirichlet distribution in (5.7), we use the final $\boldsymbol{\beta}$ parameter learned from Algorithm 9 with CIFAR-10 and VGG-16 to display the matrix of its pair-wise product $\beta_s \cdot \beta_t$ in Figure 5.8. The value in each entry is proportional to the absolute value of the label covariance in (5.7). We observe some clustering effect of class labels in CIFAR-10, such as relatively high values among the group of {airplane, auto, ship, truck} and relatively low values among the group of {bird, cat, deer, dog}. Moreover, since the $\boldsymbol{\beta}$ parameter is progressively adjusted and co-trained during model training, and the final $\boldsymbol{\beta}$ parameter is clearly not uniformly distributed, the results also validate the importance of

using parametrized label smoothing to learn to improve robustness.

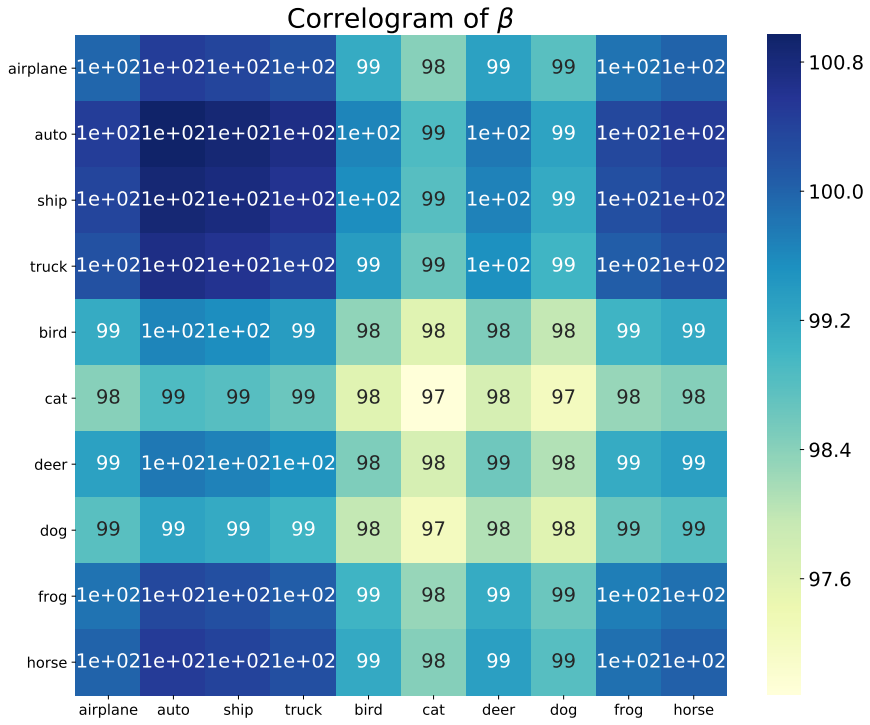


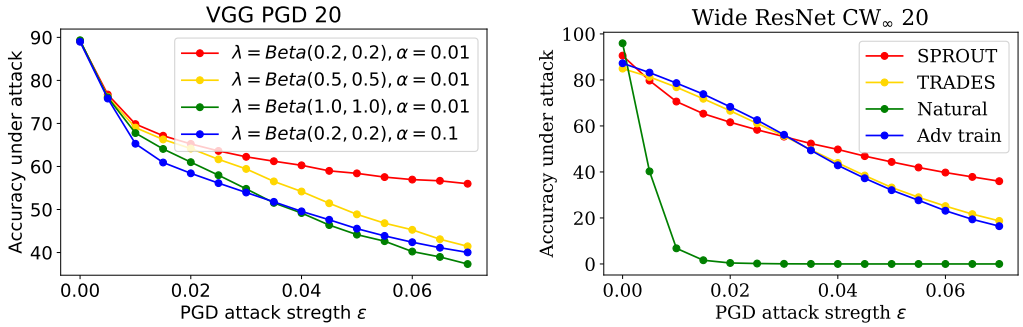
Figure 5.8: Matrix plot of the product $\beta_s \cdot \beta_t$ of the learned $\boldsymbol{\beta}$ parameter on CIFAR-10 and VGG-16.

5.3.4.9 Parameter Sensitivity Analysis

We perform an sensitivity analysis of the mixing parameter $\lambda \sim \text{Beta}(a,a)$ and the smoothing parameter α of SPROUT in Figure 5.9a. When fixing a, we find that setting α too large may affect robust accuracy, as the resulting training label distribution could be too uncertain to train a robust model. Similarly, when fixing α, setting a too large may also affect robust accuracy.

5.3.4.10 Performance with different number of random starts for PGD attack

As suggested by (MMS18), PGD attack with multiple random starts is a stronger attack method to evaluate robustness. Therefore, in Table 5.7, we conduct the following experiment



(a) (b)

Figure 5.9: Left: (a) Sensitivity of hyperparameters λ and α in SPROUT under PGD-ℓ_∞ attack Right: (b) Robust accuracy under C&W-ℓ_∞ attack.

on CIFAR-10 and Wide ResNet and find that the model trained by SPROUT can still attain at least 61% accuracy against PGD-ℓ_∞ attack ($\epsilon = 0.03$) with the number of random starts varying from 1 to 10 and with 20 attack iterations. On the other hand, the accuracy of Adversarial training and TRADES can be as low as 45.21% and 56.7%, respectively. Therefore, The robust accuracy of SPROUT is still clearly higher than other methods. We can conclude that increasing the number of random starts may further reduce the robust accuracy of all methods by a small margin, but the observed robustness rankings and trends among all methods are unchanged. We also perform one additional attack setting: 100-step PGD-ℓ_∞ attack with 10 random restarts and $\epsilon = 0.03$. We find that SPROUT can still achieve 61.18% robust accuracy.

Table 5.7: Robust accuracy of different training methods under PGD-ℓ_∞ attack with $\epsilon = 0.03$ using different number of random starts

# random start	1	3	5	8	10
Adversarial training	45.88%	45.67%	45.52%	45.52%	45.21%
TRADES	57.02%	56.84%	56.77%	56.7%	56.7%
SRPOUT	64.58%	62.53%	61.98%	61.38%	61.00%

5.3.4.11 Performance on C&W-ℓ_∞ attack

To further test the robustness on ℓ_∞ constraint, we replace the cross entropy loss with C&W-ℓ_∞ loss (CW17) in PGD attack. Similar to the PGD-ℓ_∞ attack results in Figure 5.2, Figure 5.9b shows that although SPROUT has slightly worse accuracy under small ϵ values, it attains much higher robust accuracy when $\epsilon \geq 0.03$.

5.3.4.12 Performance comparison with Free Adversarial training on ResNet-50 and ImageNet

Here we compare the performance of SPROUT with a pre-trained robust ResNet-50 model on ImageNet, which is shared by the authors in (SNG19) proposing the free adversarial training method (Free Adv Train). We find that SPROUT obtains similar robust accuracy as Free Adv Train when $\epsilon \leq 0.01$. As ϵ becomes larger, Free Adv Train has better accuracy. However, comparing to the performance of ResNet-152 in Table 5.3, SPROUT's clean accuracy on ResNet-50 actually drops by roughly 13%, indicating a large performance gap that intuitively shound not be present. Therefore, based on the current results, we postulate that the training parameters of SRPOUT for ResNet-50 may not have been fully optimized (we use the default training parameters of ResNet-152 for ResNet-50), and that it is possible that SPROUT has larger gains in robust accuracy as the RestNet models become deeper.

Table 5.8: Robust accuracy under PGD-ℓ_∞ random targeted attack on ImageNet and ResNet-50

Method	Clean Acc	$\epsilon = 0.005$	$\epsilon = 0.01$	$\epsilon = 0.015$	$\epsilon = 0.02$
Natural	76.15%	24.37%	3.54%	0.90%	0.40%
Free Adv Train	60.49%	51.35%	42.29%	32.96%	24.45%
SPROUT	61.23%	51.69%	38.14%	25.98%	18.52%

5.3.4.13 Diversity Analysis

In order to show the three modules (Dirichlet LS, GA and Mixup) in SPROUT lead to robustness gains that are complimentary to each other, we perform a diversity analysis motivated by (KQ19) to measure the similarity of their pair-wise input gradients and report the average cosine similarity in Table 5.9 over 10,000 data samples using CIFAR-10 and VGG-16. We find that the pair-wise similarity between modules is indeed quite small (< 0.103). The Mixup-GA similarity is the smallest among all pairs since the former performs both label and data augmentation based on convex combinations of training data, whereas the latter only considers random data augmentation. The Dirichlet_LS-GA similarity is the second smallest (and it is close to the Mixup-GA similarity) since the former progressively adjusts the training label $\tilde{\boldsymbol{y}}$ while the latter only randomly adjusts the training sample $\tilde{\boldsymbol{x}}$. The Dirichlet_LS-Mixup similarity is relatively high because Mixup depends on the training samples and their labels while Dirichlet LS also depend on them and the model weights. The results show that their input gradients are diverse as they point to vastly different directions. Therefore, SPROUT enjoys complimentary robustness gain and can promote robustness when combining these techniques together.

Table 5.9: Average pair-wise cosine similarity of the three modules in SPROUT

	Dirichilet LS	Mixup	GA
Dirichilet LS	NA	0.1023	0.0163
Mixup	0.1023	NA	0.0111
GA	0.0163	0.0111	NA

CHAPTER 6

Understanding Robustness Trade-off for Generalization

Albeit effective in countering adversarial examples, adversarial training often suffers from inferior performance on clean data (ZYJ19; BGH19). This observation has led prior work to extrapolate that a trade-off between robustness and accuracy may be inevitable, particularly for image classification tasks (ZYJ19; TSE19). However, (YRZ20) recently suggests that it is possible to learn classifiers both robust and highly accurate on real image data. The current state of adversarial training methods falls short of this prediction, and the discrepancy remains poorly understood.

In this chapter, we conduct an in-depth study on understanding the trade-off between robustness and clean accuracy in adversarial training, and introduce *Adversarial Masking*, a new hypothesis stating that a widely used technique, batch normalization (BN), has a significant impact on the trade-off between robustness and natural accuracy. Specifically, we break down BN into normalization and rescaling operations, and find that the rescaling operation has a significant impact on the robustness trade-off while normalization only has marginal influence. Built upon this observation, we hypothesize that adversarial masking (*i.e.*, the combination of the rescaling operation and the follow-up ReLU activation fucntion) acts as a feature masking layer that can magnify or block feature maps to influence the performance of robust or clean generalization. In this hypothesis, different rescaling parameters in BN contribute to different adversarial maskings learned through training. By using a simple linear combination of two adversarial maskings, rather than using robust features learned by adversarial training (MMS18; IST19; ZYJ19), we show that a well-balanced trade-off can be

readily achieved.

Based on the Adversarial Masking hypothesis, we further propose RobMask (**Rob**ust **Mask**ing), a new training scheme that learns an adaptive feature masking for different perturbation strengths. We use the learned adaptive feature masking to incorporate different features so that we could improve model generalization with a better robustness trade-off. Specifically, in each training iteration, we first randomly sample a perturbation strength, and generate a mini-batch of adversarial examples by conducting PGD attacks. This perturbation strength is also encoded as a low-dimensional vector,[1] which is taken as input of a learnable linear projection layer to obtain the rescaling parameter of BN (*i.e.*, adversarial masking when combined with the follow-up ReLU) for processing adversarial examples generated under the corresponding perturbation strength. Both clean and adversarial examples are used for model training. By doing so, rather than hurting the performance on clean test data, we use adversarial examples as powerful regularization to boost model generalization. Experiments on multiple benchmarks demonstrate that RobMask achieves not only better natural accuracy, but also a better trade-off between robustness and generalization.

6.1 Preliminary and Related Work

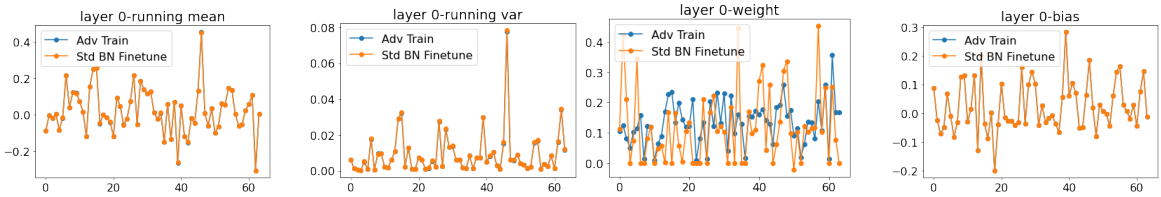
Trade-off between Robustness and Accuracy While effective in improving model robustness, adversarial training bears a performance drop on clean test data. (TSE19) provides a theoretical example of data distribution where any classifier with high test accuracy must also have low adversarial accuracy under ℓ_∞ perturbations. They claim that high performance on both accuracy and robustness may be unattainable due to their inherently opposing goals. (ZYJ19) decomposes the robust error as the sum of natural (classification) error and boundary error, and provides a differentiable upper-bound using

[1] *e.g.*, if $\epsilon_0 = 0$ is encoded as $[1.0, 0.0]$, $\epsilon_{\max} = 8/255$ is encoded as $[0.0, 1.0]$, then $\epsilon = 6/255$ can be encoded as $[0.25, 0.75]$. See Sec. 6.3 for details.

the theory of classification-calibrated loss, based on which they further propose TRADES to achieve different trade-offs by tuning the regularization term. However, (YRZ20) shows that real image datasets are actually separated so that there should exist a robust and perfectly accurate classifier. It suggests that the robustness-accuracy trade-off in deep learning is not inherent but a consequence of current methods for training robust networks. (RXY20) suggests that robust training tends to fit the local structure and lose the generalization on the global structure. By adding more unsupervised data like self-training, it could achieve a better robustness and clean accuracy trade-off.

Adversarial Examples Improve Training Adversarial training could be naturally regarded as a data augmentation technique and brings additional features to training neural networks. (MFU19) shows that adversarial training leads to a significant decrease in the curvature of the loss surface with respect to inputs, resulting in a drastically more "linear" behavior of the network. Therefore, other than treating adversarial examples as an augmentation method, it could serve as a regularizer to make the loss smoother. It has been used on a wide range of applications from reinforcement learning (ZCX20), language understanding (ZCG19; LCH20), vision-and-language understanding (GCL20), to neural architecture search (CH20a). Most recently, (XTG20) proposes AdvProp to assign another batch normalization for generating adversarial examples, and shows improved performance on clean data in image classification tasks.

Batch Normalization Batch Normalization is a widely adopted technique that enables faster and more stable training of deep neural networks. Below, we provide a brief overview of batch normalization, which paves the way to introduce our method. Specifically, batch normalization (IS15) is proposed to reduce the internal covariate shift to ease neural network training. However, several works later (STI18; BGS18) show the performance boost is brought by the regularization effect to improve the smoothness of the loss function and the ability to enable a larger learning rate, instead of reducing the internal covariate shift. Considering



(a) Running mean μ (b) Running variance σ (c) Rescaling γ (d) Rescaling β

Figure 6.1: Batch statistics in the first batch normalization (BN) layer of an adversarial trained ResNet18 model on CIFAR10, with and without further standard fine-tuning of BN (orange and blue lines, respectively). The running mean μ and variance σ, as well as the rescaling shift parameter β are almost the same (overlapped in the figure), while the rescaling weight γ has a significant difference, which has a notable contribution to the clean and robustness trade-off.

a convolutional neural network, we can define the input and output as $I_{b,c,x,y}$ and $O_{b,c,x,y}$, respectively. The dimensions correspond to examples with a batch b, channel c, and two spatial dimensions x, y. A neural network applies the same normalization for all activations in a given channel:

$$O_{b,c,x,y} \leftarrow \gamma \frac{I_{b,c,x,y} - \mu_c}{\sqrt{\sigma_c^2 + \epsilon}} + \beta \quad \forall b, c, x, y, \tag{6.1}$$

where $\mu_c = \frac{1}{|\mathcal{B}|} \sum_{b,x,y} I_{b,c,x,y}$ denotes the mean for channel c, and σ_c denotes the corresponding standard deviation. γ and β are two learnable parameters for the channel-wise affine transformation, *i.e.*, rescaling operations. ϵ is a small number to control numerical stability.

6.2 Adversarial Masking

6.2.1 Batch Normalization Acts as Adversarial Masking

(IST19) disentangles adversarial examples as a natural consequence of non-robust features. Specifically, they construct robust features from an adversarial trained "robust model" directly. Therefore, a common belief is that adversarial robustness comes from feature representations learned through adversarial training (IST19; SIT19). An interesting question we would like to ask is: can we learn robust features from a vanilla standard-trained model, or, can we

Method	Clean Acc.	Robust Acc.
Standard Training	91.97%	0.0%
+ Adv. Finetuning of BN	53.96%	26.51%
Adv. Training	78.47%	48.67%
+ Standard Finetuning of BN	86.96%	5.83%

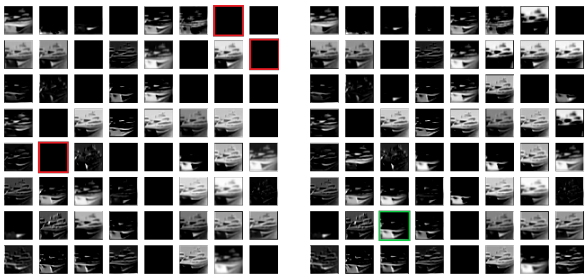
Table 6.1: Clean and robust accuracy of ResNet-18 models trained under different settings on CIFAR-10. All robust accuracy results are obtained using the $\epsilon = 8/255$ ℓ_∞ ball. (BN: Batch Normalization)

Model	p	1.0	0.8	0.6	0.4	0.2	0.0
Adv. Training with and w/o Std. Finetuning of BN	Clean Acc.	78.47%	81.16%	82.8%	84.66%	86.06%	86.96%
	Robust Acc.	48.67%	44.88%	37.2%	24.79%	12.85%	5.83%
Std. Training with and w/o Adv. Finetuning of BN	Clean Acc.	91.97%	88.66%	74.87%	58.76%	53.89%	53.96%
	Robust Acc.	0.0%	0.0%	0.12%	5.73%	19.93%	26.51%

Table 6.2: The clean and robust accuracy using different combination coefficient p on CIFAR-10 with ResNet-18. The 1st block uses adversarial trained models with and without further standard finetuning of batch normalization (BN). The 2nd block uses standard trained models with and without further adversarial finetuning of BN. All robust accuracies are obtained using $\epsilon = 8/255$ ℓ_∞ ball.

obtain non-robust features from an adversarial trained "robust model"?

Deep Analysis of BN To answer this question, we design the following experiments. We first train a ResNet-18 model with standard and adversarial training, then finetune the networks by allowing only the parameters of batch normalization (BN) to be updated while freezing other parameters. Specifically, we finetune BN in a standard trained model using adversarial training, and finetune BN in an adversarial trained model with standard training, respectively, with the same optimizer, learning rate, learning rate scheduler and number of epochs. Results are summarized in Table 6.1. Given a standard trained model, by only performing adversarial finetuning of the BN layers, the resulting model can already achieve a reasonably good robust accuracy of 26.51% (the 1st block in Table 6.1). Similarly, given an adversarial trained model, by only performing standard finetuning of the BN layers, the clean accuracy of the model increases significantly from 78.47% to 86.96% (the 2nd block).



(a) Adv. training with further standard fine-tuning of BN

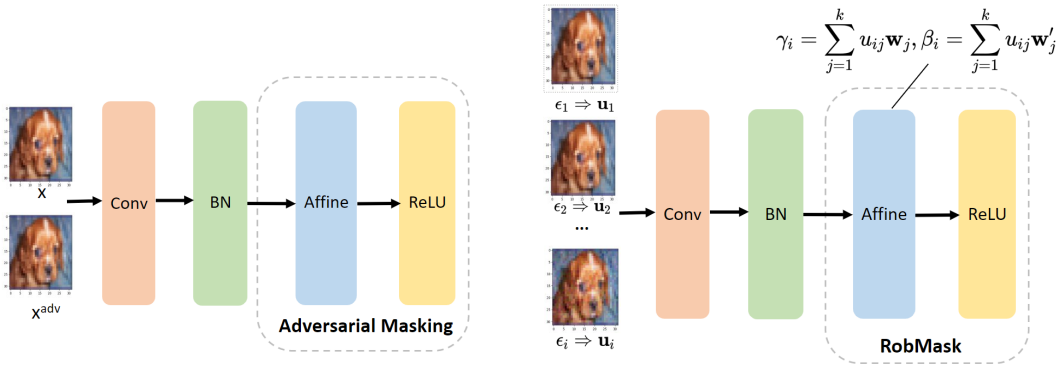
(b) Adversarial training

Figure 6.2: Illustration of the Adversarial Masking effect. We highlight several feature maps (red and green boxes), that are blocked out or magnified when comparing (a) and (b), which can be viewed as a selection mask on "non-robust" and "robust" features.

These experiments demonstrate that we can control the trade-off between clean and robust errors by only tuning the BN layer, so here comes a natural question: which part in BN has contributed to this performance trade-off? To investigate this, we take a step further to check the difference of every parameter used in BN. In particular, we study the first BN layer after the first convolution layer.

As well known, BN uses a running average of the mean and variance during testing. Figure 6.1a and 6.1b illustrate the difference on the running mean μ and running variance σ. The batch statistics with and without further standard finetuning of BN (under the setting in the 2nd block of Table 6.1) are nearly identical across all the dimensions. Figure 6.1c and 6.1d plot the learned rescaling parameters γ and β. We can see that the fine-tuned rescaling parameter γ is completely different from the original one, with β unchanged, indicating that γ has a significant impact on the clean and robust trade-off while still performing similar normalization on both sides. It clearly shows that the rescaling weight contributes more than other parameters in the batch normalization.

Analysis of the Feature Maps After BN On the other hand, Rectified Linear Unit (ReLU) (Aga18) is the most commonly used activation function in deep neural networks. The function returns 0 if it receives any negative input, and for any positive value x it returns



(a) Adversarial Masking. (b) The proposed RobMask method.

Figure 6.3: Illustration of (a) Adversarial Masking hypothesis, and (b) RobMask method for improving the generalization performance. Instead of just using a single masking for both clean and adversarial examples, we use the linear combination of k primary rescaling parameters $\{\boldsymbol{w}_j\}_{j=1}^k$ and $\{\boldsymbol{w}'_j\}_{j=1}^k$ to incorporate different perturbation strength ϵ_i.

that value back (*i.e.*, $f(x) = \max(0, x)$). During the rescaling operation, γ would magnify or shrink the magnitude of feature maps. Together with β, after ReLU activation, features that become negative will be blocked as 0. By combining the rescaling operation with the following ReLU activation function, the resulting layer can be viewed as a masking layer to magnify or block the features maps from the convolution layer (see Figure 6.3(a) for illustration).

To further validate this, we plot the feature maps after ReLU activation functions in Figure 6.2. Specifically, we extract the feature maps after the first BN and ReLU layer. Note that only a few changes are highlighted. 16/64=25% feature maps have changed dramatically; and in fact, all the feature maps have changed to some extent. Some feature maps are blocked after finetuning BN (completely black when all pixels are set to 0) as well as some are magnified significantly. We term the above observation as *Adversarial Masking*, and hypothesize that this leads to the trade-off between robustness and natural accuracy.

6.2.2 Controlling Robustness Trade-off via Adversarial Masking

The above analysis suggests that, rather than feature representations, rescaling in the BN layer together with ReLU activation function serves as a masking layer for selecting different

feature combinations that can achieve different performance trade-offs between clean and perturbed test sets. From this hypothesis, a different combination of BN and ReLU can be regarded as a different adversarial masking. With such a masking, we can readily achieve a series of trade-offs without training the model from scratch like the conventional adversarial training. In the following experiment, we use a simple linear combination of two learned adversarial maskings to achieve this trade-off, and empirically, we observe that this simple design is sufficient. Specifically, denote (γ, β) and (γ', β') as two learned adversarial maskings (*i.e.*, the learned rescaling parameters in the BN layer), and we have $\hat{\gamma} = p\gamma + (1-p)\gamma'$ and $\hat{\beta} = p\beta + (1-p)\beta'$. We then use the new adversarial masking $(\hat{\gamma}, \hat{\beta})$ for evaluation. Table 6.2 shows that different clean and robust accuracies can be readily achieved by selecting different p values.

Previous work (ZYJ19) uses regularization hyperparameter λ to balance between clean error and robust error. By tunning λ, they could achieve different robustness trade-offs. However, it takes enormous time and effort to retrain the model from the scratch. Instead, this finding inspires us that we can just store one model and employ a series of learned adversarial maskings to control the robustness trade-off at real time.

6.3 Improving Model Generalization via RobMask

As mentioned in Section 6.2, different trade-offs can be achieved by linearly combining two pre-trained batch normalization layers. However, this may not be ideal due to several deficiencies. First, the "clean" mask learned by fine-tuning is not distilled and may partially override the mask learned from adversarial examples, leading to a sub-optimal solution. Second, since every perturbation strength tends to have a different masking, if we only utilize one perturbation strength, we lose all the other maskings generated by the perturbation strength in-between. Third, it requires a careful selection of what the maximum perturbation strength is. In the extreme case, if a sample is perturbed to the decision boundary, the learned

adversarial mask might be completely meaningless. If the chosen perturbation strength is too small, there will not be enough regularization for improving generalization. To address these issues, we propose RobMask (**Rob**ust **Mask**ing), a new framework that aims to actively learn the adversarial masking to boost generalization performance.

Specifically, we propose to incorporate different perturbation strengths for model training instead of just one. Note that we could treat $\epsilon = 0$ for the unperturbed data. A straightforward way is to just learn a set of γ_i, β_i independently for every perturbation strength ϵ_i. However, due to the limited number of sampled perturbation strengths, each γ_i could have poor generalization due to over-fitting. At the same time, it weakens the correlation between all the generated maskings.[2] Instead, we need to learn the corresponding maskings jointly.

To this end, we assume that every rescaling parameter γ_i can be well-approximated by a linear combination of k basic rescaling parameters $\{\boldsymbol{w}_j\}_{j=1}^k$, where k is a small number. By encoding perturbation strength ϵ_i into a k-dimensional vector $\boldsymbol{u}_i$, we can linearly combine $\boldsymbol{w}_j$ using $\boldsymbol{u}_i$ to obtain a rescaling parameter for strength ϵ_i as:

$$\begin{aligned} \gamma_i &= \sum_{j=1}^{k} u_{ij} \boldsymbol{w}_j \,, \\ \boldsymbol{u}_i &= (1 - p_i)\boldsymbol{u}_0 + p_i \boldsymbol{u}_{\epsilon_{max}} \,, \\ \epsilon_i &= p_i \cdot \epsilon_{max} \,. \end{aligned} \tag{6.2}$$

Take $k = 2$ as an example: we can encode $\boldsymbol{u}_0 = [1.0\ 0.0]^T$ for $\epsilon = 0$, and $\boldsymbol{u}_{\epsilon_{max}} = [0.0\ 1.0]^T$ for $\epsilon_i = \epsilon_{max}$, respectively. Naturally, the intermediate perturbation strength $\epsilon_i = p_i \cdot \epsilon_{max}$ can be encoded as $\boldsymbol{u}_i = (1 - p_i)\boldsymbol{u}_0 + p_i \boldsymbol{u}_{\epsilon_{max}}$. Therefore, instead of learning γ_i separately for every ϵ_i, we learn a low-rank matrix $\mathbf{W} = [\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_k]$ to incorporate different perturbation strengths and learn a series of maskings. Similarly, we learn another matrix **W'** for β_i. During training, in every iteration, with a randomly selected perturbation strength $\epsilon_i = p_i \cdot \epsilon_{max}$, we first generate a mini-batch of adversarial examples by conducting PGD attacks. Then, we learn the rescaling parameter of BN by using a low-rank linear layer (**W** and **W'**) and encoded

[2] A certain degree of correlation still exists since all the other parameters besides BN are still shared.

Algorithm 10 The proposed RobMask method for improving model generalization.

Input: Training dataset $\{\boldsymbol{x}_i, y_i\}_{i=1}^n$, perturbation upper bound ϵ_{max}.
for epoch= $1, \ldots, N$ **do**
  for $i = 1, \ldots, B$ **do**
    Sample a random number p from 0 to 1
    # obtain the perturbation strength in this mini-batch
    $\epsilon_i \leftarrow p \cdot \epsilon_{max}$
    # encode the perturbation strength as a vector
    $\boldsymbol{u}_i \leftarrow \text{Encode}(\epsilon_i)$
    $\boldsymbol{\delta}_i \leftarrow 0$
    for $j = 1, \ldots, m$ **do**
      # PGD adversarial attack
      $\boldsymbol{\delta}_i \leftarrow \boldsymbol{\delta}_i + \alpha \cdot \text{sign}(\nabla_{\boldsymbol{\delta}} \ell(f_\theta(\boldsymbol{x}_i + \boldsymbol{\delta}_i), y_i)$
      $\boldsymbol{\delta}_i \leftarrow \max(\min(\boldsymbol{\delta}_i, \epsilon_i), -\epsilon_i)$
    # the rescaling parameters in BN
    $\gamma_i \leftarrow \sum_{j=1}^k u_{ij} \boldsymbol{w}_j$, $\beta_i \leftarrow \sum_{j=1}^k u_{ij} \boldsymbol{w}'_j$
    # update **W** and **W'**
    $\mathbf{W} \leftarrow \mathbf{W} - \eta_1 \cdot \nabla_{\mathbf{W}} \ell(f_\theta(\boldsymbol{x}_i + \boldsymbol{\delta}_i, \gamma_i, \beta_i), y_i)$
    $\mathbf{W'} \leftarrow \mathbf{W'} - \eta_2 \cdot \nabla_{\mathbf{W}'} \ell(f_\theta(\boldsymbol{x}_i + \boldsymbol{\delta}_i, \gamma_i, \beta_i), y_i)$
    # update neural network parameters θ
    $\theta \leftarrow \theta - \eta_3 \cdot \nabla_\theta \ell(f_\theta(\boldsymbol{x}_i + \boldsymbol{\delta}_i, \gamma_i, \beta_i), y_i)$
return $\theta, \mathbf{W}, \mathbf{W'}$

attack strength $\boldsymbol{u}_i$. Finally, we minimize the total loss using stochastic gradient descent (SGD) to update model parameters. We summarize the detailed algorithm in Algorithm 10.

Connection with AdvProp (XTG20) hypothesizes that the performance degradation on unperturbed test dataset is mainly caused by the distribution mismatch between adversarial examples and clean images. They propose AdvProp to assign an auxiliary batch normalization for adversarial examples, and show that adversarial examples can be useful to achieve better performance on clean test data. However, as shown in Figure 6.1, the running mean and variance are kept the same after fine-tuning. We argue that the improved model generalization is realized by a different adversarial mask learned by auxiliary batch normalization in the AdvProp procedure.

Although we utilize adversarial training to boost generalization as well, our approach

Model	#Epochs	CIFAR-10			CIFAR-100		
		BN	AdvProp	RobMask	BN	AdvProp	RobMask
ResNet-18	20	92.59%	93.45%	**94.64%**	75.88%	76.15%	**77.61%**
	100	94.87%	95.3%	**96.10%**	77.7%	76.82%	**78.77%**
DenseNet-121	20	93.26%	94.61%	**95.30%**	74.79%	73.61%	**75.11%**
	100	94.71%	94.61%	**96.47%**	77.63%	76.88%	**80.18%**
Preact-18	20	91.93%	92.55%	**94.04%**	70.79%	72.71%	**73.59%**
	100	94.37%	95.33%	**95.97%**	76.14%	76.87%	**78.19%**
ResNeXt-29	20	93.12%	93.37%	**95.03%**	74.26%	72.18%	**74.65%**
	100	95.15%	95.29%	**96.06%**	78.60%	76.35%	**79.54%**

Table 6.3: Comparison on CIFAR-10/100 over ResNet-18, DenseNet-121, Preact-18, and ResNeXt-29. Models are trained for 20 and 100 epochs using normal Batch Normalization (BN), AdvProp and our RobMask. RobMask shows a significant performance improvement on all model architectures. Also, RobMask trained with 20 epochs achieves a comparable performance to 100-epoch training using BN and AdvProp.

has clear differences. First, in AdvProp, there is no connection between the traditional and auxiliary batch normalization (BN). The traditional BN only obtains inputs from clean data, and the auxiliary BN only obtains inputs from adversarial examples. This type of disentanglement would completely separate different masks, which violates the reality that some masks can be useful for both clean and robust performance. Second, AdvProp has to designate a perturbation strength that should not be too large or too small, which is difficult to tune in practice. To be noted, while AdvProp fixed one perturbation strength to one batch-norm, each $\boldsymbol{w}_k$ in RobMask denotes a "basis" and all the BN parameters for each different epsilon are based on a linear combination of these bases. Also, the proposed RobMask method is more general than Advprop, and AdvProp can be considered as one special case of RobMask when we set the linear layer rank k to 2 and freeze $p = 1$ in the whole training process.

Model	ImageNet		
	BN	AdvProp	RobMask
ResNet-18	69.76%	69.79%	**70.14%**

Table 6.4: Comparison on ImageNet over ResNet-18. Models are trained for 105 epochs using normal Batch Normalization (BN), AdvProp and our RobMask.

6.4 Experimental Results

In this section, we conduct experiments to show that RobMask can successfully improve generalization performance. We also provide additional robustness evaluation for completeness.

Experimental Setup **Datasets and Model Architectures** We use two popular datasets CIFAR-10 and CIFAR-100 (KH09) and one large-scale dataset ImageNet (DDS09) for experiments. For model architectures, we use the popular ResNet (HZR16a) family including Preact ResNet (HZR16b), ResNeXt (XGD16) and the recent well-performed DenseNet (HLW16).

Baselines We compare RobMask with two baselines: (*i*) AdvProp: Dual batch normalization (XTG20), where different batch normalizations are used for clean and adversarial examples during training; and (*ii*) BN: Standard training with normal batch normalization enabled. Note that as our main goal is to improve the generalization performance instead of robust test accuracy, we do not compare against standard adversarial training methods, as they are reported to largely decrease generalization performance (MMS18; ZYJ19; BGH19).

Implementation Details For CIFAR-10 and CIFAR-100, we set the number of iterations in adversarial attack to 7 for all the methods during training. All PGD attacks are non-targeted attacks with random initialization. We set the PGD attack strength $\epsilon = 8/255$ with cross-entropy (CE) loss and the step-size to $\epsilon/5$. All models are trained using SGD with momentum 0.9, weight decay 5×10^{-4}. We use cosine learning rate scheduling with initial learning rate $\gamma_1 = \gamma_2 = \gamma_3 = 0.1$. For ImageNet, we follow the PyTorch implementation at a

ϵ	Attack	0	2/255	4/255	6/255	8/255
Adv. Training	PGD	78.86%	72.61%	65.27%	57.15%	**47.97%**
	AutoAttack	78.86%	70.99%	62.94%	53.83%	44.66%
AdvProp	PGD	85.98%	78.34%	69.24%	57.61%	46.04%
	AutoAttack	85.98%	77.73%	67.57%	55.36%	43.13%
RobMask	PGD	**89.99%**	**82.91%**	**72.23%**	**58.63%**	44.50%
	AutoAttack	**89.99%**	**81.87%**	**70.99%**	55.90%	41.70%
RobMask ($\epsilon = 10/255$)	AutoAttack	86.50%	78.47%	68.08%	**56.89%**	**44.69%**

Table 6.5: Robust accuracy under different levels of PGD ℓ_∞ attacks and AutoAttack on CIFAR-10 with ResNet-18 architecture. RobMask clearly outperforms AdvProp and standard adversarial training in all the test perturbation strengths except $\epsilon = 8/255$ on which AdvProp and standard adversarial training are trained.

Github repo.[3] To have a fair comparison, for RobMask, we set $k = 2$ and $\epsilon_{max} = 8/255$ in all our experiments, which has the same number of model parameters and same regularization strength as AdvProp. All our experiments are implemented in PyTorch.

Experimental Results **Generalization** Table 6.3 summarize the results of all the evaluated methods on CIFAR-10 and CIFAR-100. Across all the tested model architectures, RobMask shows a significant improvement over both normal batch normalization (BN) and AdvProp. Specifically, as shown in Table 6.3, for 100-epochs training on CIFAR-10, RobMask achieves around 1.5% test accuracy improvement over BN and 0.8% over AdvProp, respectively. Similar improvements can also be observed on the CIFAR-100 dataset. Further, when comparing results between Table 6.3, we observe that RobMask also leads to faster convergence: 20-epochs training using RobMask leads to results that are comparable to 100-epochs training using BN. Additionally, we add the large-scale dataset ImageNet into the comparison. Table 6.4 summarize the results with ResNet-18 on ImageNet dataset. Clearly, while AdvProp has a very limited improvement on ResNet-18, RobMask has around 0.4 percent improvement, which further shows RobMask's effectiveness on generalization.

[3] `https://github.com/tingxueronghua/pytorch-classification-advprop`

Robustness Evaluation In addition to improved generalization performance, our method can also achieve a better robust and natural accuracy trade-off over adversarial training. For CIFAR-10 and CIFAR-100, we evaluate all the methods under the white-box $\epsilon = 8/255$ ℓ_∞-norm bounded non-targeted PGD attack. Specifically, we use 100-step PGD with step size $\epsilon/5$) that is equipped with random start. Moreover, to further verify the robust accuracy achieved, we use AutoAttack (CH20b) to evaluate the performance. Note that, when $\epsilon = 0$, robust accuracy is reduced to the test accuracy of unperturbed (natural) test samples, *i.e*, clean accuracy. Results are summarized in Table 6.5. RobMask clearly outperforms other methods among ϵ from 0 to 6/255. However, RobMask performs slightly worse on $\epsilon = 8/255$. It is because both AdvProp and adversarial training models are trained with adversarial examples generated with $\epsilon = 8/255$, while our methods use a random perturbation where $\epsilon_{max} = 8/255$. That is, we use a weaker perturbation strength compared to both AdvProp and adversarial training.

To achieve a better result on $\epsilon = 8/255$, we relax the max epsilon constraint from 8/255 to 10/255. From Table 6.5, we can see that the clean performance drops slightly with an increasing robust accuracy on $\epsilon >= 6/255$ so that we now can achieve a better robust accuracy on $\epsilon = 8/255$. Even with the slight degrading performance on clean accuracy, RobMask achieves a better adversarial robustness trade-off over other methods.

Importance of Low-rank Matrix We conduct an ablation study on DenseNet-121 over CIFAR-10 to investigate the importance of using a low-rank matrix and to incorporate multiple perturbation strengths. Also, we show the impact of choosing different rank k in AdvProp. Specifically, we add another dimension and set $k = 3$. Note that, even if AdvProp could use many BNs simultaneously, they assume every BN is independent. Here, we extend AdvProp to use a randomly selected strength $\epsilon_i = p_i \cdot \epsilon_{max}$ to generate adversarial examples, and then feed into auxiliary batch normalization. AdvProp can also be generalized using multiple auxiliary BNs when given multiple perturbation strengths. In experiments, instead of an auxiliary batch normalization for adversarial examples generated by $\epsilon = 8/255$, we also give

Method	Clean Acc.
AdvProp	94.61%
with random perturbation strength	94.50%
with 2 auxiliary BNs	95.10%
RobMask ($k = 2$)	**96.47%**
RobMask ($k = 3$)	96.40%

Table 6.6: Comparison on CIFAR-10 over DenseNet-121 on AdvProp, its extensions, and RobMask with $k = 3$. Models are trained for 100 epochs.

another batch normalization for adversarial examples generated under $\epsilon = 4/255$. Table 6.6 shows AdvProp has degenerated performance when using random perturbation strength. When adding more auxiliary batch normalization, the performance improves slightly, also observed in (XTG20). However, RobMask still significantly outperforms AdvProp variants. When $k = 3$, RobMask could achieve a comparable accuracy 96.40% and still have a better performance than AdvProp with a large margin.

CHAPTER 7

Conclusion

Neural networks provide state-of-the-art results for most machine learning tasks. Unfortunately, neural networks are vulnerable to adversarial examples. That is, a slightly modified example could be easily generated and fool a well-trained image classifier based on deep neural networks (DNNs) with high confidence. This makes it difficult to apply neural networks in security-critical areas like autonomous driving. This area is known as adversarial robustness in machine learning. The central focus of this thesis is to understand when and why this phenomenon happens and make a way to improve its ability to counter such examples.

7.1 Adversarial Attacks

To understand when the adversarial examples appear, first, we start with adversarial attacks on image classification tasks. Specifically, we first formulate the attack into an optimization problem and introduce how to find to solve the problem into different settings including white-box, soft-label black-box, and hard-label black-box. Then we take a deep dive into the most practical setup for evaluating the adversarial robustness of a machine learning system with limited access: the hard-label black-box attack setting for generating adversarial examples, where limited model queries are allowed and only the decision is provided to a queried data input. While all previous works either use a transfer-based attack (which has a very low success rate) or random walk (which is extremely slow and lack theoretical guarantees), we are the first to formalize this problem into an optimization problem and developed OPT-attack and Sign-OPT, two state-of-art methods which could be widely used

to test the robustness of many real-world models in the industry. Second, other than image classification, we also extend this discussion to the more challenging discrete domains such as natural language processing (NLP) tasks. Since the inherent discreteness nature of text strings, we propose Seq2Sick to evaluate the robustness of the seq2seq model, which has inspired many follow-up works and has been cited since its debut. Moreover, we take a step further to study how to evaluate the robustness of a goal-oriented dialogue system where there can be many turns of interactions between adversarial and target agents, while all of the previous work consider attacking a static model, where except input image/sentence there is no interaction between the attacker and the target model. These two works first point out the current deep learning-based seq2seq model like machine translation and dialog agent has serious concerns on facing noisy inputs and malicious agents.

7.2 Adversarial Defenses

While crafting adversarial examples is an important technique to evaluate the robustness of DNNs, there is a huge need for improving the model's robustness as well. Enhancing model robustness under new and even adversarial environments is a crucial milestone toward building trustworthy machine learning systems. In general, it could be divided into two categories: attack-dependent defense. and attack-independent defenses. As one of the most effective ways to improve the model's robustness in attack-dependent defense methods, although many variants have been proposed, adversarial training sacrifices huge performance on natural data. We think the loss of clean accuracy in adversarial training is mainly due to the inductive biases introduced in the training methods. For instance, enforcing all the samples to have a uniform-robust region may be a wrong assumption. So we develop CAT which adaptively customizes the perturbation level and the corresponding label for each training sample in adversarial training. Also, as one of the most effective methods to improve robustness, adversarial training suffers from poor scalability and attack specificity. Therefore, on the

other hand, we take a different perspective on attack-independent defenses, which attempts to add the out-of-distribution data into the training to gain the property of adversarial examples. We, therefore, propose a new framework SPROUT that adjusts training label distribution via our proposed parametrized label smoothing technique, making training free of attack generation and more scalable. We also motivate SPROUT using a general formulation based on vicinity risk minimization, which includes many robust training methods as special cases.

7.3 Future Directions

Building better tools to evaluate robustness: Despite many adversarial attacks have been proposed in different settings, they are insufficient to measure the model's adversarial robustness because nearly all of them are gradient-based. Without a good gradient, where following the gradient does not successfully optimize the loss, those attacks cannot succeed. Therefore, some of the current defense methods would cause gradient masking or obfuscated gradient so that it could achieve good performance to counter those attacks, while the adversarial robustness is not truly improved. On the other hand, machine learning verification has been a reliable measurement of robustness by bounding the prediction behavior of the model within a certain input region. However, all existing verification approaches only focus on ReLU networks, and the bounds need to be re-derived and re-implemented for every different network architecture and activation, which is very time-consuming and scale-limited. Therefore, it is a great idea to develop a reliable, fast way to measure the model's adversarial robustness.

Robust models without sacrificing clean accuracy: Although many algorithms have been proposed to improve the robustness of machine learning models, all of them sacrifice performance on natural data. Despite some previous work reveals the difficulty of adversarial generalization, recent work has proved that a robust neural network actually exists but it's

hard to find it. I think the loss of clean accuracy in adversarial training is mainly due to the inductive biases introduced in the training methods. For instance, enforcing all the samples to have a uniform robust region may be a wrong assumption. Furthermore, in certified defense, since the bounds are usually tight in some sub-regions, training procedure will bias the model towards those regions, leading to degenerated models (many neurons are never activated). We will thus develop ways to improve existing methods by mitigating the inductive biases and try to obtain robust models without sacrificing the performance on natural data.

Harnessing robustness to improve model performance: Adversarial training was originally proposed as a means to enhance the security of machine learning systems. Despite it has been shown it would cause the accuracy decreasing in the image classification domain, however, recently, it has been shown that adversarial training significantly improves the performance of state-of-the-art image classification and language understanding tasks when being correctly applied. Theoretically, adversarial training makes the loss smoothness so that it helps to avoid model converging to bad local optimal. Intuitively, adversarial examples make network representations align better with salient data characteristics and the learned representations are less sensitive to texture distortions and focus more on shape information. Moreover, adversarial examples could be thought as special data augmentation and strong regularization. With the aforementioned benefits, it is promising for us to use adversarial robustness as a tool to understand neural networks' behavior and enable us to build a better performed and trustworthy artificial intelligent model.

Bibliography

[ACW18] Anish Athalye, Nicholas Carlini, and David Wagner. "Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples." *International Coference on International Conference on Machine Learning*, 2018.

[Aga18] Abien Fred Agarap. "Deep learning using rectified linear units (relu)." *arXiv preprint arXiv:1803.08375*, 2018.

[ASC19] Moustafa Alzantot, Yash Sharma, Supriyo Chakraborty, Huan Zhang, Cho-Jui Hsieh, and Mani B Srivastava. "Genattack: Practical black-box attacks with gradient-free optimization." In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1111–1119, 2019.

[BCB14] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate." *arXiv preprint arXiv:1409.0473*, 2014.

[BDL18] Thomas Brunner, Frederik Diehl, Michael Truong Le, and Alois Knoll. "Guessing smart: Biased sampling for efficient black-box adversarial attacks." *arXiv preprint arXiv:1812.09803*, 2018.

[BGH19] Yogesh Balaji, Tom Goldstein, and Judy Hoffman. "Instance adaptive adversarial training: Improved accuracy tradeoffs in neural nets." *arXiv preprint arXiv:1910.08051*, 2019.

[BGS18] Johan Bjorck, Carla Gomes, Bart Selman, and Kilian Q Weinberger. "Understanding batch normalization." *arXiv preprint arXiv:1806.02375*, 2018.

[BHL17] Arjun Nitin Bhagoji, Warren He, Bo Li, and Dawn Song. "Exploring the Space of Black-box Attacks on Deep Neural Networks." *arXiv preprint arXiv:1712.09491*, 2017.

[BRB17] Wieland Brendel, Jonas Rauber, and Matthias Bethge. "Decision-Based Adversarial Attacks: Reliable Attacks Against Black-Box Machine Learning Models." *arXiv preprint arXiv:1712.04248*, 2017.

[BSW14] Pierre Baldi, Peter Sadowski, and Daniel Whiteson. "Searching for exotic particles in high-energy physics with deep learning." *Nature communications*, **5**:4308, 2014.

[BWA18] Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Animashree Anandkumar. "signSGD: Compressed Optimisation for Non-Convex Problems." In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 560–569, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[CH20a] Xiangning Chen and Cho-Jui Hsieh. "Stabilizing differentiable architecture search via perturbation-based regularization." In *International Conference on Machine Learning*, pp. 1554–1565. PMLR, 2020.

[CH20b] Francesco Croce and Matthias Hein. "Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks." *arXiv preprint arXiv:2003.01690*, 2020.

[CJL16] William Chan, Navdeep Jaitly, Quoc Le, and Oriol Vinyals. "Listen, attend and spell: A neural network for large vocabulary conversational speech recognition." In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pp. 4960–4964. IEEE, 2016.

[CLC19] Minhao Cheng, Thong Le, Pin-Yu Chen, Huan Zhang, JinFeng Yi, and Cho-Jui Hsieh. "Query-Efficient Hard-label Black-box Attack: An Optimization-based Approach." In *International Conference on Learning Representations*, 2019.

[CRK19] Jeremy M Cohen, Elan Rosenfeld, and J Zico Kolter. "Certified adversarial robustness via randomized smoothing." *International Conference on Machine Learning*, 2019.

[CRS19] Yair Carmon, Aditi Raghunathan, Ludwig Schmidt, Percy Liang, and John C Duchi. "Unlabeled data improves adversarial robustness." *Neural Information Processing Systems*, 2019.

[CSC19] Minhao Cheng, Simranjit Singh, Pin-Yu Chen, Sijia Liu, and Cho-Jui Hsieh. "Sign-OPT: A Query-Efficient Hard-label Adversarial Attack." *arXiv preprint arXiv:1909.10773*, 2019.

[CSV09] Andrew R Conn, Katya Scheinberg, and Luis N Vicente. *Introduction to derivative-free optimization*, volume 8. Siam, 2009.

[CSZ18] Pin-Yu Chen, Yash Sharma, Huan Zhang, Jinfeng Yi, and Cho-Jui Hsieh. "EAD: elastic-net attacks to deep neural networks via adversarial examples." In *AAAI*, 2018.

[CW17] Nicholas Carlini and David Wagner. "Towards evaluating the robustness of neural networks." In *Security and Privacy (SP), 2017 IEEE Symposium on*, pp. 39–57. IEEE, 2017.

[CWB01] Olivier Chapelle, Jason Weston, Léon Bottou, and Vladimir Vapnik. "Vicinal risk minimization." In *Advances in neural information processing systems*, pp. 416–422, 2001.

[CZS17] Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. "Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models." In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pp. 15–26. ACM, 2017.

[DBW12] John C Duchi, Peter L Bartlett, and Martin J Wainwright. "Randomized smoothing for stochastic optimization." *SIAM Journal on Optimization*, **22**(2):674–701, 2012.

[DDS09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. "Imagenet: A large-scale hierarchical image database." In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248–255. IEEE, 2009.

[DSL18] Gavin Weiguang Ding, Yash Sharma, Kry Yik Chau Lui, and Ruitong Huang. "Max-margin adversarial (mma) training: Direct input space margin maximization through adversarial training." *arXiv preprint arXiv:1812.02637*, 2018.

[EIA18] Logan Engstrom, Andrew Ilyas, and Anish Athalye. "Evaluating and understanding the robustness of adversarial logit pairing." *arXiv preprint arXiv:1807.10272*, 2018.

[ERL17] Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. "HotFlip: White-Box Adversarial Examples for NLP." *arXiv preprint arXiv:1712.06751*, 2017.

[FMM18] Mikhail Figurnov, Shakir Mohamed, and Andriy Mnih. "Implicit reparameterization gradients." In *Advances in Neural Information Processing Systems*, pp. 441–452, 2018.

[Fri97] Jerome H Friedman. "On bias, variance, 0/1—loss, and the curse-of-dimensionality." *Data mining and knowledge discovery*, **1**(1):55–77, 1997.

[GCL19] Ruiqi Gao, Tianle Cai, Haochuan Li, Cho-Jui Hsieh, Liwei Wang, and Jason D Lee. "Convergence of Adversarial Training in Overparametrized Neural Networks." In *Advances in Neural Information Processing Systems*, pp. 13009–13020, 2019.

[GCL20] Zhe Gan, Yen-Chun Chen, Linjie Li, Chen Zhu, Yu Cheng, and Jingjing Liu. "Large-Scale Adversarial Training for Vision-and-Language Representation Learning." *arXiv preprint arXiv:2006.06195*, 2020.

[GD19] Morgane Goibert and Elvis Dohmatob. "Adversarial Robustness via Adversarial Label-Smoothing." *arXiv preprint arXiv:1906.11567*, 2019.

[GL13] Saeed Ghadimi and Guanghui Lan. "Stochastic first-and zeroth-order methods for nonconvex stochastic programming." *SIAM Journal on Optimization*, **23**(4):2341–2368, 2013.

[GSS14] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. "Explaining and harnessing adversarial examples." *arXiv preprint arXiv:1412.6572*, 2014.

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

[GWL18] Zhitao Gong, Wenlu Wang, Bo Li, Dawn Song, and Wei-Shinn Ku. "Adversarial Texts with Gradient Methods." *arXiv preprint arXiv:1801.07175*, 2018.

[HLW16] Gao Huang, Zhuang Liu, and Kilian Q Weinberger. "Densely connected convolutional networks. CoRR abs/1608.06993 (2016)." *arXiv preprint arXiv:1608.06993*, 2016.

[HMK19] Dan Hendrycks, Mantas Mazeika, Saurav Kadavath, and Dawn Song. "Using Self-Supervised Learning Can Improve Model Robustness and Uncertainty." *Neural Information Processing Systems*, 2019.

[HNW16] Thanh-Le Ha, Jan Niehues, and Alexander Waibel. "Toward multilingual neural machine translation with universal encoder and decoder." *arXiv preprint arXiv:1611.04798*, 2016.

[HZR16a] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[HZR16b] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Identity mappings in deep residual networks." In *European conference on computer vision*, pp. 630–645. Springer, 2016.

[IEA18] Andrew Ilyas, Logan Engstrom, Anish Athalye, and Jessy Lin. "Black-box Adversarial Attacks with Limited Queries and Information." In *International Conference on Machine Learning*, pp. 2142–2151, 2018.

[IS15] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." *arXiv preprint arXiv:1502.03167*, 2015.

[IST19] Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Madry. "Adversarial examples are not bugs, they are features." In *Advances in Neural Information Processing Systems*, pp. 125–136, 2019.

[JL17] Robin Jia and Percy Liang. "Adversarial examples for evaluating reading comprehension systems." *arXiv preprint arXiv:1707.07328*, 2017.

[JLM18] Kwang-Sung Jun, Lihong Li, Yuzhe Ma, and Jerry Zhu. "Adversarial attacks on stochastic bandits." In *Advances in Neural Information Processing Systems*, pp. 3640–3649, 2018.

[KGB16] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. "Adversarial machine learning at scale." *arXiv preprint arXiv:1611.01236*, 2016.

[KH09] Alex Krizhevsky, Geoffrey Hinton, et al. "Learning multiple layers of features from tiny images." 2009.

[KQ19] Sanjay Kariyappa and Moinuddin K Qureshi. "Improving Adversarial Robustness of Ensembles with Diversity Training." *arXiv preprint arXiv:1901.09981*, 2019.

[KSH19] Daniel Kang, Yi Sun, Dan Hendrycks, Tom Brown, and Jacob Steinhardt. "Testing Robustness Against Unforeseen Adversaries." *arXiv preprint arXiv:1908.08016*, 2019.

[LBB98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE*, **86**(11):2278–2324, 1998.

[LCC19] Sijia Liu, Pin-Yu Chen, Xiangyi Chen, and Mingyi Hong. "signSGD via Zeroth-Order Oracle." In *International Conference on Learning Representations*, 2019.

[LCH20] Xiaodong Liu, Hao Cheng, Pengcheng He, Weizhu Chen, Yu Wang, Hoifung Poon, and Jianfeng Gao. "Adversarial training for large neural language models." *arXiv preprint arXiv:2004.08994*, 2020.

[LCL16] Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song. "Delving into transferable adversarial examples and black-box attacks." *arXiv preprint arXiv:1611.02770*, 2016.

[LKC18] Sijia Liu, Bhavya Kailkhura, Pin-Yu Chen, Paishun Ting, Shiyu Chang, and Lisa Amini. "Zeroth-Order Stochastic Variance Reduction for Nonconvex Optimization." In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pp. 3727–3737. Curran Associates, Inc., 2018.

[LYD17] Mike Lewis, Denis Yarats, Yann N Dauphin, Devi Parikh, and Dhruv Batra. "Deal or no deal? end-to-end learning for negotiation dialogues." *arXiv preprint arXiv:1706.05125*, 2017.

[MDG16] Takeru Miyato, Andrew M Dai, and Ian Goodfellow. "Adversarial training methods for semi-supervised text classification." *arXiv preprint arXiv:1605.07725*, 2016.

[MFF16] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. "Deepfool: a simple and accurate method to fool deep neural networks." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2574–2582, 2016.

[MFU19] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Jonathan Uesato, and Pascal Frossard. "Robustness via curvature regularization, and vice versa." In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9078–9086, 2019.

[MMS18] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. "Towards deep learning models resistant to adversarial attacks." In *ICLR*, 2018.

[MR10] Sébastien Marcel and Yann Rodriguez. "Torchvision the Machine-vision Package of Torch." In *Proceedings of the 18th ACM International Conference on Multimedia*, MM '10, pp. 1485–1488, New York, NY, USA, 2010. ACM.

[Nes11] Yurii Nesterov. "Random gradient-free minimization of convex functions." Technical report, 2011.

[NS17] Yurii Nesterov and Vladimir Spokoiny. "Random gradient-free minimization of convex functions." *Foundations of Computational Mathematics*, **17**(2):527–566, 2017.

[PMG17] 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*, pp. 506–519. ACM, 2017.

[PRW02] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. "BLEU: a method for automatic evaluation of machine translation." In *Proceedings of the*

40th annual meeting on association for computational linguistics, pp. 311–318. Association for Computational Linguistics, 2002.

[RCW15] Alexander M Rush, Sumit Chopra, and Jason Weston. "A neural attention model for abstractive sentence summarization." *arXiv preprint arXiv:1509.00685*, 2015.

[RXY20] Aditi Raghunathan, Sang Michael Xie, Fanny Yang, John Duchi, and Percy Liang. "Understanding and mitigating the tradeoff between robustness and accuracy." *arXiv preprint arXiv:2002.10716*, 2020.

[SFK19] Robert Stanforth, Alhussein Fawzi, Pushmeet Kohli, et al. "Are Labels Required for Improving Adversarial Robustness?" *Neural Information Processing Systems*, 2019.

[SGH19] Ali Shafahi, Amin Ghiasi, Furong Huang, and Tom Goldstein. "Label Smoothing and Logit Squeezing: A Replacement for Adversarial Training?" *arXiv preprint arXiv:1910.11585*, 2019.

[SIT19] Shibani Santurkar, Andrew Ilyas, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Madry. "Image synthesis with a single (robust) classifier." In *Advances in Neural Information Processing Systems*, pp. 1262–1273, 2019.

[SNG19] Ali Shafahi, Mahyar Najibi, Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S Davis, Gavin Taylor, and Tom Goldstein. "Adversarial Training for Free!" *Neural Information Processing Systems*, 2019.

[SSK19] Mayank Singh, Abhishek Sinha, Nupur Kumari, Harshitha Machiraju, Balaji Krishnamurthy, and Vineeth N Balasubramanian. "Harnessing the vulnerability of latent layers in adversarially trained models." *arXiv preprint arXiv:1905.05186*, 2019.

[STI18] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. "How does batch normalization help optimization?" *arXiv preprint arXiv:1805.11604*, 2018.

[SVI16] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. "Rethinking the inception architecture for computer vision." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

[SVL14] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. "Sequence to Sequence Learning with Neural Networks." In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pp. 3104–3112, 2014.

[SZ15] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." *International Conference on Learning Representations*, 2015.

[SZS13] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. "Intriguing properties of neural networks." *arXiv preprint arXiv:1312.6199*, 2013.

[TB19] Florian Tramèr and Dan Boneh. "Adversarial Training and Robustness for Multiple Perturbations." *Neural Information Processing Systems*, 2019.

[TCB19] Sunil Thulasidasan, Gopinath Chennupati, Jeff Bilmes, Tanmoy Bhattacharya, and Sarah Michalak. "On Mixup Training: Improved Calibration and Predictive Uncertainty for Deep Neural Networks." *Neural Information Processing Systems*, 2019.

[TSE19] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and

Aleksander Madry. "Robustness may be at odds with accuracy." In *International Conference on Learning Representations*, 2019.

[TTC18] Chun-Chen Tu, Pai-Shun Ting, Pin-Yu Chen, Sijia Liu, Huan Zhang, Jinfeng Yi, Cho-Jui Hsieh, and Shin-Ming Cheng. "AutoZOOM: Autoencoder-based Zeroth Order Optimization Method for Attacking Black-box Neural Networks." *CoRR*, **abs/1805.11770**, 2018.

[TTC19] Chun-Chen Tu, Paishun Ting, Pin-Yu Chen, Sijia Liu, Huan Zhang, Jinfeng Yi, Cho-Jui Hsieh, and Shin-Ming Cheng. "AutoZOOM: Autoencoder-based Zeroth Order Optimization Method for Attacking Black-box Neural Networks." *AAAI*, 2019.

[VLB18] Vikas Verma, Alex Lamb, Christopher Beckham, Amir Najafi, Ioannis Mitliagkas, Aaron Courville, David Lopez-Paz, and Yoshua Bengio. "Manifold mixup: Better representations by interpolating hidden states." *International Conference on Machine Learning*, 2018.

[VSP17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. "Attention is all you need." In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

[Wan19] Jianyu Wang. "Bilateral Adversarial Training: Towards Fast Training of More Robust Models Against Adversarial Attacks." *International Conference on Computer Vision*, 2019.

[WM19] Colin Wei and Tengyu Ma. "Improved sample complexities for deep networks and robust classification via an all-layer margin." *arXiv preprint arXiv:1910.04284*, 2019.

[WMB19] Yisen Wang, Xingjun Ma, James Bailey, Jinfeng Yi, Bowen Zhou, and Quan-

quan Gu. "On the Convergence and Robustness of Adversarial Training." In *International Conference on Machine Learning*, pp. 6586–6595, 2019.

[WZY19] Yisen Wang, Difan Zou, Jinfeng Yi, James Bailey, Xingjun Ma, and Quanquan Gu. "Improving adversarial robustness requires revisiting misclassified examples." In *International Conference on Learning Representations*, 2019.

[XGD16] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. "Aggregated residual transformations for deep neural networks. 2016." *arXiv preprint arXiv:1611.05431*, 2016.

[XTG20] Cihang Xie, Mingxing Tan, Boqing Gong, Jiang Wang, Alan L Yuille, and Quoc V Le. "Adversarial examples improve image recognition." In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 819–828, 2020.

[YCH18] Puyudi Yang, Jianbo Chen, Cho-Jui Hsieh, Jane-Ling Wang, and Michael I Jordan. "Greedy Attack and Gumbel Attack: Generating Adversarial Examples for Discrete Data." *arXiv preprint arXiv:1805.12316*, 2018.

[YRB18] Dong Yin, Kannan Ramchandran, and Peter Bartlett. "Rademacher complexity for adversarially robust generalization." *arXiv preprint arXiv:1810.11914*, 2018.

[YRZ20] Yao-Yuan Yang, Cyrus Rashtchian, Hongyang Zhang, Ruslan Salakhutdinov, and Kamalika Chaudhuri. "A Closer Look at Accuracy vs. Robustness." *arXiv preprint arXiv:2003.02460*, 2020.

[ZCD18] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. "mixup: Beyond empirical risk minimization." *International Conference on Learning Representations*, 2018.

[ZCG19] Chen Zhu, Yu Cheng, Zhe Gan, Siqi Sun, Tom Goldstein, and Jingjing Liu. "Freelb: Enhanced adversarial training for natural language understanding." In *ICLR*, 2019.

[ZCX20] Huan Zhang, Hongge Chen, Chaowei Xiao, Bo Li, Duane Boning, and Cho-Jui Hsieh. "Robust deep reinforcement learning against adversarial perturbations on observations." *arXiv preprint arXiv:2003.08938*, 2020.

[ZDS17] Zhengli Zhao, Dheeru Dua, and Sameer Singh. "Generating natural adversarial examples." *arXiv preprint arXiv:1710.11342*, 2017.

[ZNR17] Valentina Zantedeschi, Maria-Irina Nicolae, and Ambrish Rawat. "Efficient defenses against adversarial attacks." In *ACM Workshop on Artificial Intelligence and Security*, pp. 39–49, 2017.

[ZSL16] Stephan Zheng, Yang Song, Thomas Leung, and Ian Goodfellow. "Improving the robustness of deep neural networks via stability training." In *Proceedings of the ieee conference on computer vision and pattern recognition*, pp. 4480–4488, 2016.

[ZW19] Haichao Zhang and Jianyu Wang. "Defense Against Adversarial Attacks Using Feature Scattering-based Adversarial Training." *Neural Information Processing Systems*, 2019.

[ZYJ19] Hongyang Zhang, Yaodong Yu, Jiantao Jiao, Eric P Xing, Laurent El Ghaoui, and Michael I Jordan. "Theoretically principled trade-off between robustness and accuracy." *International Conference on Machine Learning*, 2019.