EPFL

Thèse n° 9128

Towards Verifiable, Generalizable and Efficient Robust Deep Neural Networks.

Présentée le 26 août 2022

Faculté informatique et communications
Laboratoire d'images et représentation visuelle
Programme doctoral en informatique et communications

pour l'obtention du grade de Docteur ès Sciences

par

Chen LIU

Acceptée sur proposition du jury

Prof. M. Jaggi, président du jury
Prof. S. Süsstrunk, Dr M. Salzmann, directeurs de thèse
Dr R. Tomioka, rapporteur
Prof. F. Tramer, rapporteur
Prof. N. Flammarion, rapporteur

École polytechnique fédérale de Lausanne

2022

Those who understand others are clever,
but those who know themselves are truly wise.
— Laozi

To my parents...

Acknowledgments

Spending five years in my twenties concentrating on an interesting topic is the treasure of my life. This thesis, as well as my whole Ph.D period, would not be completed smoothly without the help and encouragement from many people. Here, I express my gratitude to all of them.

First, I would like to thank my supervisor Prof. Sabine Süsstrunk. Sabine is the person who saved my research career when I was at loss before joining her laboratory. She provides generous support for my research work and enough flexibility on the project topic. In addition to specific techniques, working with Sabine greatly improves my fundamental research skills, including how to write academic papers, how to do presentations, and how to supervise students. Finally, Sabine consistently encourages me whenever my submitted work is rejected by a conference, which greatly motivates me to polish up my research.

Second, I am grateful to my co-supervisor Dr. Mathieu Salzmann. I am in particular thankful for all the discussions with Mathieu. Mathieu has broad knowledge in different domains, he can usually point out the limitations of unreliable methods proposed by me, which greatly improves the efficiency of my research. Moreover, Mathieu is a perfect paper editor and can always organize the content in a systematic, logical, and professional way. Checking Mathieu's edits always benefits me, including making me realize the limitations of my current work and enlightening the way to improve it.

Third, many thanks to two senior researchers I have worked with: Dr. Ryota Tomioka from Microsoft Research Cambridge and Prof. Tong Zhang from Hong Kong University of Science and Technology. The first year and half of my Ph.D. were supported by Microsoft Research, Ryota can always provide me with interesting research problems and potential solutions when my research was at a loss. Discussion with Ryota also makes me realize the difference between academia and industry. Tong is a famous theoretician in machine learning and I collaborate with his lab in the last two years of my Ph.D. I have only a few direct discussions with Tong, but every time Tong can quickly point out the potential improvement of my works, especially the limitations of the theoretical proofs, even after my double-check.

I would also thank all other jury members of my Ph.D. committee, Prof. Florian Tramèr, Prof. Martin Jaggi, and Prof. Nicolas Flammarion, for their suggestions and questions during the oral exam on improving this thesis.

Some works included in my thesis are not possible without my colleagues' collaboration. I am thankful to all of them: Dr. Ya-Ping Hsieh, Dr. Tao Lin, Zhichao Huang, Ziqi Zhao, and Fabian Latorre. Some of them are theoreticians, and some of them are practitioners, but all provide valuable contributions to the work I participate in. Among them, I would especially thank

Zhichao for our weekly discussion and brainstorm, and for Tao for the career discussion.
I am very fortunate to be a member of the Image and Visual Representation Lab (IVRL) of EPFL, where I meet wonderful persons from all over the world: Dr. Ruofan Zhou, Dr. Edo Collins, Dr. Marjan Shahpaski, Dr. Seungryong Kim, Dr. Hak Gu Kim, Dr. Majed El Helou, Dr. Tong Zhang, Deblina Bhattacharjee, Bahar Aydemir, Baran Ozaydin, Ehsan Pajouheshgar, Martin Nicolas Everaert, Yufan Ren, Peter Grönquist and Dongqing Wang. I would especially thank Tong for our experience of joint supervision on a Master Thesis project, and for Majed for his genius ideas of motivating students when we two are teaching assistants in the computational photography course. Moreover, I also thank two secretaries, Francoise Behn and Nicoletta Isaac, for helping me deal with administrative work.
In addition, I owe my deepest gratitude to my friend both inside and outside EPFL, my Ph.D. life would not be that happy without them. I would thank Ruofan, who is also my undergrad classmate and Ph.D. labmate, for helping me settle down in Lausanne. I spent wonderful times traveling with Tao, Xiaoying, Jing, Xiaoyu, Long, Zijian, Zhongqi, Junxiong, Shengzhao, Lu, and Sailan in many different countries. I enjoyed lunch chat with Kamalaruban, Teresa, Junhong, Zilu, and Sitian, for their wisdom and humor. I tried different kinds of restaurants in Lausanne with Hang, Zhengchao, Jiande, Yao, Yanfei, Fatih, Ali, and Ahmet in the last five years, which is really amazing. During the COVID pandemic, I am really grateful to Maksym for organizing adversarial learning workshops when we are all locked down at home. There are also many other people that lighten my Ph.D. life, the list is just too long to exhaustively put here.
I give my sweetest gratitude to my girlfriend, Qi Dou, for her company throughout my whole Ph.D. Qi spent one and half years in London as a postdoc, before moving to Hong Kong as a faculty. Before my final move to Hong Kong after my graduation, we have never resided in the same country during my Ph.D. However, both of us are devoted and offer encouragement whenever we meet difficulties, and we are always able to solve them. Time and distance cannot separate Qi from me, let alone other difficulties. Having Qi with me, my career plan is much clearer and my life is much more colorful.
Finally, I would like to acknowledge the unconditional support and care from my parents and the whole family. They always double my happiness, and share my stress and frustration during my study, without which I am unable to move forward to get my Ph.D. The long-distance between Lausanne and my hometown, as well as the pandemic, may make the family reunion difficult in the last few years, but I believe things will get much better in the future.

Lausanne, July 19, 2022 Chen Liu

Abstract

In the last decade, deep neural networks have achieved tremendous success in many fields of machine learning. However, they are shown vulnerable against adversarial attacks: well-designed, yet imperceptible, perturbations can make the state-of-the-art deep neural networks output incorrect results. Understanding adversarial attacks and designing algorithms to make deep neural networks robust against these attacks are key steps to building reliable artificial intelligence in real-life applications.

In this thesis, we will first formulate the robust learning problem. Based on the notations of empirical robustness and verified robustness, we design new algorithms to achieve both of these types of robustness. Specifically, we investigate the robust learning problem from the optimization perspectives. Compared with classic empirical risk minimization, we show the slow convergence and large generalization gap in robust learning. Our theoretical and numerical analysis indicates that these challenges arise, respectively, from non-smooth loss landscapes and model's fitting hard adversarial instances. Our insights shed some light on designing algorithms for mitigating these challenges.

Robust learning has other challenges, such as large model capacity requirements and high computational complexity. To solve the model capacity issue, we combine robust learning with model compression. We design an algorithm to obtain sparse and binary neural networks and make it robust. To decrease the computational complexity, we accelerate the existing adversarial training algorithm and preserve its performance stability.

In addition to making models robust, our research provides other benefits. Our methods demonstrate that robust models, compared with non-robust ones, usually utilize input features in a way more similar to the way human beings use them, hence the robust models are more interpretable. To obtain verified robustness, our methods indicate the geometric similarity of the decision boundaries near data points. Our approaches towards reliable artificial intelligence can not only render deep neural networks more robust in safety-critical applications but also make us better aware of how they work.

Keywords: deep neural networks, adversarial robustness, optimization, efficient learning.

Résumé

Les réseaux de neurones profonds ont connu un énorme succès dans de nombreux domaines de l'apprentissage automatique. Cependant, ils se montrent vulnérables aux attaques adverses : des perturbations bien définies, mais imperceptibles, pouvant faire en sorte que les réseaux de neurones profonds produisent des résultats incorrects. Comprendre ces attaques adverses et concevoir des algorithmes pour rendre les réseaux de neurones profonds robustes à ces attaques sont des étapes clés pour construire une intelligence artificielle fiable dans des applications réelles.

Dans cette thèse, nous formulons d'abord le problème d'apprentissage robuste. Sur la base de notions de robustesse empirique et de robustesse vérifiée, nous concevons de nouveaux algorithmes pour atteindre ces deux types de robustesse. Plus précisément, nous étudions le problème d'apprentissage robuste du point de vue de l'optimisation. Nous démontrons la convergence lente et les difficultés de généralisation de l'apprentissage robuste par rapport à la minimisation du risque empirique classique. Notre analyse théorique et numérique indique que ces défis découlent, respectivement, du manque de régularité de la fonction de coût minimisée et de l'ajustement du modèle aux exemples d'entraînement adverse difficiles. Nos observations permette une meilleurs compréhension et facilite la conception d'algorithmes pour atténuer ces défis.

L'apprentissage robuste présente d'autres difficultés, telles que des exigences de capacité de modèle importantes et une complexité de calcul élevée. Pour résoudre le problème de capacité du modèle, nous combinons un apprentissage robuste avec la compression du modèle. Nous concevons un algorithme pour obtenir des réseaux de neurones clairsemés et binaires et les rendre robuste. Pour réduire la complexité de calcul, nous accélérons l'algorithme d'entraînement adverse existant et préservons sa stabilité de performance.

En plus de rendre les modèles robustes, nos recherches offrent d'autres avantages. Nos méthodes démontrent que les modèles robustes, par rapport aux modèles non robustes, utilisent généralement les caractéristiques d'entrée d'une manière plus similaire à la façon dont les êtres humains les utilisent, ce qui rend les modèles robustes plus interprétables. Pour obtenir une robustesse vérifiée, nos méthodes indiquent la similarité géométrique des frontières de décision près des points de données. Nos approches vers une intelligence artificielle fiable peuvent non seulement rendre les réseaux de neurones profonds plus robustes dans les applications critiques pour la sécurité, mais aussi nous rendre plus conscients de leur fonctionnement.

Résumé

Mots clés : réseaux de neurones profonds, robustesse aux attaques adverses, optimisation.

Contents

Notation

Unless explicitly stated, we use light letters, lowercase bold letters, and uppercase bold letters to represent scalars,49 vectors, and higher dimensional tensors, respectively.
The table below list the variables and their corresponding meanings globally. For any variable or expression var, $\overline{var}$ and $\underline{var}$ represent its upper bound and lower bound, respectively. $\widetilde{var}$ represents its estimation.

f	model
p	l_p norm
y	instance label
K	number of categories
L	number of layers
M	dimension of input instances
N	number of training instances
$\mathbf{x}$	input instance
ϵ	strength of the adversarial budget
θ	model parameters
Δ	adversarial perturbation
Φ	activation function
$\mathcal{D}$	data distribution
$\mathcal{L}$	loss function
$\mathcal{S}$	adversarial budget
$\mathbb{1}$	indicator function

1 Introduction

Over the last decade, deep neural networks have achieved great success in the advancement of artificial intelligence, including computer vision [39, 63, 84], natural language processing [10, 147, 170] and reinforcement learning [133, 134]. However, these state-of-the-art models are shown to be vulnerable against adversarial attacks [101, 142]. Some well-designed perturbations of the input can make these models output, with very high confidence, incorrect predictions, but they do not change the semantic meaning of the input hence are imperceptile to human beings. Modern deep neural networks contain millions of trainable parameters and are highly non-convex. Due to this black-box nature, it is not clear what causes these adversarial examples. Therefore, obtaining deep neural networks robust against these attacks becomes a crucial yet challenging task.

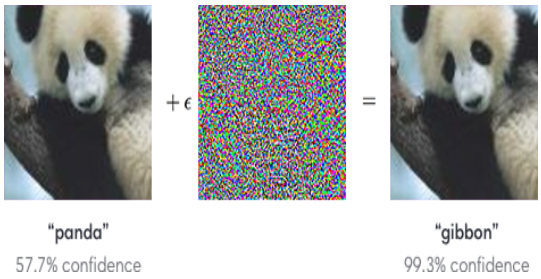
We demonstrate some adversarial examples in different applications in Figure 1.1[1]. These examples indicate that adversarial examples broadly exist in different applications based on deep neural networks. Designing methods that train robust neural networks not only benefits safety-critical applications, such as medical image analysis and auto-driving, but also advances our understandings about how these models work.

1.1 Problem Formulation

In supervised learning, we are provided N K-category training instances, i.e., data-target pairs $\{\mathbf{x}_i, y_i\}_{i=1}^N$, sampled from an unknown distribution $\mathcal{D}$. Here, $\mathbf{x}_i \in \mathbb{R}^M$ and $y_i \in \{0, 1, 2, ..., K-1\}$. We train a model f parameterized by θ to minimize the *empirical risk* as follows:

$$\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(f(\theta, \mathbf{x}_i), y_i) \tag{1.1}$$

[1] Images in Figure 1.1 are from the literature and can be found on the Internet. Image (a) is from [53]. Image (b) is from [4]. Image (c) is from www.bostonherald.com, published on February 25th, 2020. Image (d) is from www.cse.gatech.edu, published on September 21st, 2018.



(a) Image Classification

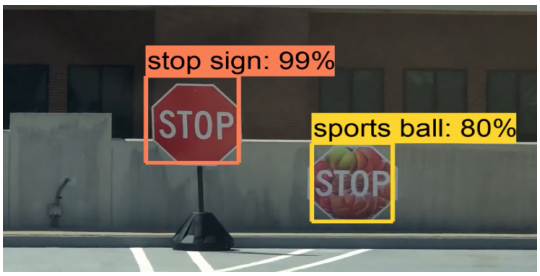
Original Text Prediction = **Negative**. (Confidence = 78.0%)
This movie had terrible acting, terrible plot, and terrible choice of actors. (Leslie Nielsen ...come on!!!) the one part I considered slightly funny was the battling FBI/CIA agents, but because the audience was mainly kids they didn't understand that theme.
Adversarial Text Prediction = **Positive**. (Confidence = 59.8%)
This movie had horrific acting, horrific plot, and horrifying choice of actors. (Leslie Nielsen ...come on!!!) the one part I regarded slightly funny was the battling FBI/CIA agents, but because the audience was mainly youngsters they didn't understand that theme.

Table 1: Example of attack results for the sentiment analysis task. Modified words are highlighted in green and red for the original and adversarial texts, respectively.

Original Text Prediction: **Entailment** (Confidence = 86%)
Premise: *A runner wearing purple strives for the finish line.*
Hypothesis: *A runner wants to head for the finish line.*
Adversarial Text Prediction: **Contradiction** (Confidence = 43%)
Premise: *A runner wearing purple strives for the finish line.*
Hypothesis: *A racer wants to head for the finish line.*

(b) Semantic Analysis

(c) Reinforcement Learning



(d) Object Detection

Figure 1.1 – Some examples of adversarial attacks in different applications. (a) Imperceptible noise makes the image classifiers output the wrong label. (b) Replacing a few words with their synonyms can flip the predictions. (c) Reinforcement learning algorithms in the auto-driving system fail to adapt to new environment. (d) Adding new textures to the "stop sign" can fool the object detection system.

Here, $\mathcal{L}$ is the loss function measuring the disparity between the model's output $f(\theta, \mathbf{x}_i)$ and the target y_i. In a classification problem, y_i is the label of the corresponding data $\mathbf{x}_i$, and $\mathcal{L}$ is usually the softmax cross-entropy function. Let $\widehat{\theta}$ be the result by solving the minimization problem (1.1); it is evaluated based on a test set consisting of another N_{te} instances sampled from the same distribution.

However, the deep neural networks trained by minimizing the empirical risk in (1.1) are not robust against adversarial attacks at all. To obtain robust models, we need to optimize the model's performance on the adversarially perturbed inputs and to minimize the *adversarial empirical risk* as follows:

$$\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \max_{\Delta_i \in \mathcal{S}(\epsilon, \mathbf{x}_i)} \mathcal{L}(f(\theta, \mathbf{x}_i + \Delta_i), y_i) \tag{1.2}$$

Here, the set $\mathcal{S}(\epsilon, \mathbf{x}_i)$ is called the *adversarial budget* that contains all allowable perturbations of the input. Ideally, the adversarial budget should include all perturbations that do not change the semantic meanings of the input. This budget can depend on the input instance $\mathbf{x}_i$ and is parameterized by ϵ which represents the strength of the perturbation. However, it is difficult to mathematically exactly depict the semantic meanings universally applicable for different applications [156]. In most of the existing literature [53, 98], the adversarial

budget is defined based on an l_p norm constraint: $\{\Delta | \|\Delta\|_p \leq \epsilon\}$. Unless explicitly stated, to represent the adversarial budget for notation simplicity, we use $\mathbb{S}_\epsilon^{(p)} := \{\Delta | \|\Delta\|_p \leq \epsilon\}$ and further $\mathbb{S}_\epsilon := \{\Delta | \|\Delta\|_\infty \leq \epsilon\}$.

Solving the inner maximization problem of (1.2) exactly is NP-hard in general, especially in the cases when f is a high-dimensional non-convex function such as a deep neural network. In this regard, we can approximately solve the min-max problem (1.2) by deriving either the upper bound or the lower bound of inner maximal. On one hand, we have by definition $\forall \Delta \in \mathbb{S}_\epsilon^{(p)}, \mathcal{L}(f(\theta, \mathbf{x}_i + \Delta), y_i) \leq \max_{\Delta_i \in \mathbb{S}(\epsilon, \mathbf{x}_i)} \mathcal{L}(f(\theta, \mathbf{x}_i + \Delta_i), y_i)$. Then, we can design powerful adversarial attack algorithms to find a near-optimal adversarial perturbation Δ to derive a lower bound as tight as possible. On the other hand, to derive the upper bound of the inner maximal, we need the upper bound and the lower bound of the model's output $f(\theta, \mathbf{x}_i + \Delta_i)$ for $\Delta_i \in \mathbb{S}_\epsilon^{(p)}$. If the upper bound of the inner maximal still represents a correct prediction, we can then guarantee that the model is provably robust for the input instance $(\mathbf{x}_i, y_i)$ given the adversarial budget $\mathbb{S}_\epsilon^{(p)}$. A tighter bound of the model's output will enable more input instances to be verified.

In summary, the *adversarial attack* problem is to find near-optimal adversarial perturbations for a tight lower bound of the inner maximal in (1.2); *adversarial training* uses these generated adversarial perturbations to solve the outer minimization. We use *empirical robust accuracy* to represent the proportion of input instances that are robust against adversarial attacks. Correspondingly, the *robustness verification* problem is to find a tight upper bound of the inner maximal in (1.2); *provably robust training* minimizes this upper bound to obtain models that are provably robust. We use *verified robust accuracy* to represent the proportion of the input instances that are proven robust under the given adversarial budget. By definition, empirical robust accuracy and verified robust accuracy are the upper bound and the lower bound of the *true robust accuracy*, respectively. Here, *true robust accuracy* is the proportion of input instances that are "actually" robust. Mathematically, it is defined as $\frac{1}{N}\sum_{i=1}^{N} \mathbb{1}(\forall \Delta \in \mathbb{S}_\epsilon^{(p)}, y_i = \text{argmax}_j f(\theta, \mathbf{x}_i + \Delta)_j)$ where $\mathbb{1}$ is the indicator function. Calculating the true robust accuracy is proven NP-hard [154], it is impossible to evaluate the modern deep learning models by this metric. Therefore, empirical robust accuracy and verified robust accuracy become two alternative and feasible metrics for robustness evaluation in practice.

In this thesis, we concentrate on adversarial robustness that is significantly different from robustness against random noises. The latter evaluates the model's average performance in the presence of random perturbations. In contrast, we focus more on the model's worse-case performance within all allowable perturbations. Formally, improving adversarial robustness solves a min-max optimization problem (1.2), whereas improving robustness against random noise does not.

1.2 Extra Benefits of Robustness

Apart from obtaining a model resistant to adversarial attacks, robust learning can also provide other benefits and advance our understandings of how deep neural network works. First, by making models' predictions invariant, we implicitly encode some priors in the robust learning. Specifically, as adversarial budgets (ideally) represent the set of input perturbations that do not change the inputs' semantic meanings, robust learning constrains the models to avoid using features sensitive to such perturbations. In other words, robust learning prevents the models from using features that do not change the inputs' semantic meanings hence they focus on the features strongly correlated with the semantic meanings. In [76], the former are called *non-robust features* and the latter are called *robust features*. By robust learning, we encode priors indicated by the adversarial budgets to make the models predict more like humans do. Existing works [76, 95] show that models by robust learning are more interpretable and explainable, compared with non-robust ones.

In addition, for neural network using ReLU activation function [84] (the most popular activation function) robust learning also increases the sparsity of intermediate activations [29] thus facilitates model compression. This is because the ReLU function always outputs the constant zero when the input is non-positive. Such *inactivated neurons* are predominant in models trained by robust learning, because it encourages the models' output to be invariant to the perturbation of the input. More inactivated neurons in the intermediate layers means more sparse intermediate activations.

Finally, robust models are found to have better performance in transfer learning [121]. Compared with normal pre-trained models, those that are adversarially trained against perturbations have better performance when they are fine-tuned by another relevant dataset. Intuitively, the robust features learned by robust models are more easily generalized hence make the model better adapted to the new dataset. Based on this nice property, robust pre-trained models are better choices for downstream tasks.

1.3 Challenges in Robust Learning

Compared with classic empirical risk minimization in (1.1), robust learning problem (1.2) is more challenging, because the latter optimizes the worst-case loss objective. The challenges of obtaining empirical robustness and verified robustness include but are not limited to the following:

1. **Degradation of Clean Accuracy.** Although both adversarial training and provably robust training can make models robust against adversarial attacks, they sacrifice the performance on clean inputs. It has been theoretically and empirically shown that there is a clear trade-off between the model's clean accuracy and its robust accuracy [178]. In order to make some input instances robust against adversarial attacks, some algorithms

can overly sacrifice the performance on other input instances. We call this phenomenon *over-regularization*.

2. **Slower Convergence.** Adversarial training results in much slower convergence, compared with empirical risk minimization. Such phenomenon arises from both the complexity of single mini-batch updates and the number of updates required for convergence. First, we generate adversarial examples in adversarial training, which dramatically increases the computational complexity for one mini-batch update. Second, adversarial training needs more mini-batch updates to convergence to the neighborhood of a local minimum, thus further prolonging the training process.

3. **Larger Generalization Gap.** Robust learning yields a much larger generalization gap than empirical risk minimization. Sufficiently large models can perfectly fit the training set, both under empirical risk minimization and robust learning. However, the state-of-the-art robust accuracy is still below 70% on CIFAR10 test set under the l_∞ norm based adversarial budget with $\epsilon = 8/255$ [30], compared with near-perfect performance on the clean test set. In addition, the robust accuracy on the test set can degrade significantly in the late phase of adversarial training [118]. We call this phenomenon *adversarial overfitting*, as it does not occur in empirical risk minimization.

4. **Larger Model Capacity Requirements.** By adjusting the width and the depth of the models, we can compare the performance in clean accuracy and robust accuracy for models of different capacities. Comprehensive results [98, 166] have shown that robust learning needs a larger model capacity to achieve competitive performance than empirical risk minimization. By decreasing the capacity of small models, robust learning first fails to converge, whereas empirical risk minimization still obtains non-trivial performance. By increasing the capacity of large models, the performance of empirical risk minimization will first saturate, yet the performance of robust learning still improves.

5. **More Training Data Needs.** The pioneering work in 2018 [126] theoretically concludes that the sample complexity of robust learning is significantly higher than that of empirical risk minimization, irrespective of the model architecture and the training algorithm. Using extra data [18], data augmentation [114] or generated synthetic data [57] can greatly improve the robustness of the model.

Compared with solving one of the points above, it is even more challenging to jointly mitigate multiple points, because solving the challenge above might worsen the other issues. For example, to achieve a better trade-off between the clean accuracy and the robust accuracy, some algorithms use more training data and larger models. To compress the large model while preserving robustness, many algorithms introduce significant computational overhead. In summary, the existing methods are still far away from overcoming the challenges in robust learning comprehensively, there is still much room for improvement.

1.4 Summary of Contributions

In the previous sections, we have demonstrated the motivation, formulation, benefits and the challenges of robust learning. In Chapter 2, we conduct comprehensive literature reviews in several aspects of the problem we study. In the subsequent three sections, we will focus on each aspect of the themes in this thesis, namely *verifiable*, *generalizable* and *efficient*. At the end, we summarize these works and list some of the unsolved problems in this field.

In Chapter 3, we study verified robustness based on the geometric properties of the model's decision boundary. We are, to the best of our knowledge, the first to introduce methods for verifying non-uniform adversary-free regions with larger volumes than the uniform ones. The algorithms for verifying non-uniform bounds are also tools for studying the geometric properties of the model's decision boundaries. In this regard, we can distinguish robust input features from non-robust ones, based on their respective verified adversary-free bounds. We find that the robust and non-robust features of the robust models are much more aligned with human perception, hence robust models are easily interpretable. In addition to verification, we can use the linear approximation of the model's output to bound the adversary-free region by a polyhedral envelope. By introducing a regularization scheme for enlarging the polyhedral envelope, we can train provably robust models. To the best of our knowledge, we are also the first to introduce differentiable approximation of the input's distance to the decision boundary for general neural network models. Extensive experimental results indicate that our methods can mitigate the over-regularization issue: our trained model can achieve much better clean accuracy with competitive robust accuracy compared with baselines.

In Chapter 4, we focus on empirical robustness, especially adversarial training, the most popular and effective method for achieving empirical robustness. We study, in particular, the two challenges of adversarial training: slow convergence and large generalization gap. Our investigations are theoretically grounded, from linear toy models to general nonlinear models, and they are validated by the numerical experiments. For slow convergence, we study the loss landscape of adversarial training. In particular, we prove that adversarial attacks make the loss landscape in the parameter space non-smooth. Non-smooth loss landscapes causes more scattered gradients and slower convergence. We also study the connectivity of the local minima in the adversarial loss landscape. Our results indicate these local minima in the adversarial cases are less connected and more diverse than the non-adversarial cases. Using our findings on the adversarial loss landscape, we propose a warm-up strategy in the adversarial budget to avoid convergence failure and a periodic scheduler to ensemble more diverse minima. For large generalization gaps, we study the adversarial overfitting phenomenon from the aspect of training instances. We conclude that adversarial overfitting occurs when the models fit hard adversarial training instances, from both theoretical and empirical perspectives. We find our theory can explain the success of existing methods that mitigate adversarial overfitting and improve the generalization: they all implicitly avoid fitting hard adversarial training instances.

In Chapter 5, we study how to improve the efficiency of robust learning. We focus on model

compression and training acceleration. For model compression, we extend the *Lottery Ticket Hypothesis* [57] to the adversarial cases. We design novel pruning strategies and an initialization scheme for discovering robust sub-networks inside a randomly-initialized binary network. We show competitive performance in both clean accuracy and robust accuracy, even when comparing with other methods that train full-precision networks. For accelerated adversarial training, we propose an instance-wise adaptive step size in order to solve the *catastrophic overfitting* issue [158]. Theoretically, it has faster convergence, when the magnitudes of the input gradients has large variances. Empirically, we show that it achieves better and more stable performance with almost no computational overhead, compared with existing accelerated adversarial training methods.

In summary, we highlight the contributions of this thesis in the points listed below:

- Verified Robustness
  - First method for verifying non-uniform adversary-free volumes.
  - New method for distinguishing robust input features from non-robust ones.
  - Bounds of the decision boundary by a geometry-inspired polyhedral envelope.
  - First method to obtain a differentiable estimate of the distance between the input and the model's decision boundary for networks of general activation functions.
  - A regularization scheme to improve provably robust training and mitigate over-regularization (i.e., with better accuracy on the clean inputs).
- Empirical Robustness
  - First to point out the non-smooth nature of the adversarial loss landscape.
  - Convergence analysis of adversarial training.
  - Demonstration of weaker connectivities of local minima in the adversarial loss landscape.
  - A periodic training scheme to improve the performance of adversarial training.
  - Theoretical and empirical analyses of adversarial overfitting, showing adversarial overfitting arises from fitting hard adversarial training instances.
  - Case study of existing methods, showing ones that successfully mitigate adversarial overfitting implicitly avoid fitting hard adversarial instances.
- Efficiency of Robust Learning
  - Novel pruning strategy and initialization schemes for obtaining robust sub-networks from randomly-initialized binary networks.
  - Instance-wise adaptive step size for stable and improved fast adversarial training.

2 Related Works

Over the last few years, there have been many works that propose methods for improving the robustness of deep neural networks. Some of these works, however, have been proved invalid in the presence of more powerful and adaptive adversarial attack algorithms. There is an "armed race" between the attack and the defense side of the robust learning problem. In this chapter, we comprehensively review the related works of both sides. Using the challenges found by these works, we also include those that improving the efficiency of robust learning.

2.1 Adversarial Attacks

The robustness properties have been well studied on traditional machine learning models for years, such as support vector machine (SVM) [119, 168]. The robust learning problem for these models can usually be formulated in the form of their trainable parameters, which greatly facilitates our finding solutions that use optimization techniques.

The existence of adversarial examples for deep neural network models was first pointed out in [14]. Due to the high complexity and non-convexity of deep neural networks, to construct adversarial examples, we need to use numerical methods, instead of analytical ones like in the previous works. [53] proposes *Fast Gradient Sign Method (FGSM)* to find adversarial examples in the case of l_∞ adversarial budget $\mathcal{S}_\epsilon$:

$$\Delta = \epsilon \text{sign}\left(\nabla_{\mathbf{x}} \mathcal{L}(f(\theta, \mathbf{x}), y)\right) \tag{2.1}$$

Here, sign is the element-wise sign function that returns -1 for negative numbers and $+1$ otherwise. The value of the perturbation Δ by the FGSM method is the optimality for maximizing the first order Taylor expansion of $\mathcal{L}(f(\theta, \mathbf{x}+\Delta), y) \simeq \mathcal{L}(f(\theta, \mathbf{x}), y) + \langle \Delta, \nabla_{\mathbf{x}} \mathcal{L}(f(\theta, \mathbf{x}), y) \rangle$ under the constrain $\|\Delta\|_\infty \leq \epsilon$. In this regard, we can generalize the FGSM to general l_p adversarial budget $\mathcal{S}_\epsilon^{(p)}$.

As the deep neural networks are usually highly non-convex models, the FGSM can overestimate the linearity of the models. Therefore, [87] introduced *Iterative Fast Gradient Sign*

Method (IFGSM). In IFGSM, we also consider the l_∞ adversarial budget $\mathcal{S}_\epsilon$ and iteratively update the perturbation by the following update rule:

$$\Delta \leftarrow \Pi_{\mathcal{S}_\epsilon}\left[\Delta + \alpha \text{sign}\left(\nabla_\Delta \mathcal{L}(f(\theta, \mathbf{x}+\Delta), y)\right)\right] \tag{2.2}$$

Here, $\alpha \leq \epsilon$ is the step size used in each iteration, and $\Pi_{\mathcal{S}_\epsilon}$ represents projecting into the adversarial budget $\mathcal{S}_\epsilon$. In [98], the *Projected Gradient Descent (PGD)* method further improves IFGSM by random initialization and multiple restarts. PGD randomly initializes Δ within the adversarial budget and runs IFGSM, based on several random starting points of Δ; a robust model should give the correct prediction for all these random starts. PGD is considered one of the strongest attacks that use the first-order information [98].

In addition to the methods directly using the gradient of the loss objective, [17] proposes *CW* attack that converts adversarial budget constraints to a regularization item and achieves impressive performance under l_2 adversarial budget. *DeepFool* in [102] generates adversarial perturbations by iteratively estimating the distance of the perturbed inputs to the model's decision boundary. Similarly, the *Fast Adaptive Boundary (FAB)* attack in [32] estimates optimal adversarial examples, based on the linear approximation of the model's decision boundary. Furthermore, [101] finds it is possible to construct *universal adversarial perturbations*, across different input instances that make the undefended models give incorrect predictions.

All the attacks above belong to *untargeted attacks*. In other words, we make the models output the incorrect predictions; they can be any predictions other than the correct ones. In contrast, *targeted attacks* [174] aim to make the models predict the specific predictions. Formally, untargeted attacks maximize the loss objective on the correct label y: $\max_{\Delta \in \mathcal{S}_\epsilon^{(p)}} \mathcal{L}(f(\theta, \mathbf{x}+\Delta), y)$; whereas, targeted attacks minimize the loss objective on the targeted label y_{target} [86]: $\min_{\Delta \in \mathcal{S}_\epsilon^{(p)}} \mathcal{L}(f(\theta, \mathbf{x}+\Delta), y_{\text{target}})$. For K-category classification problem, the untargeted attack problem can be decomposed into $K-1$ targeted attack sub-problems: each sub-problem uses one of the $K-1$ incorrect labels. Such multi-targeted attacks can boost the performance over single untarget attacks [33].

So far, all the methods introduced need the gradient of the loss objective. This is the *white box* setting, where the attacker has the access to all the details of the model, and these attack methods are called *white-box attacks*. In many cases, the attackers might not have access to the details of the model hence have to use *black-box attack* methods to generate adversarial perturbations. Using the information that the attackers have access to, black-box attack methods include those based on the model's output probability [2, 5, 12, 60, 74, 100], those based on the model's output label [1, 15, 24, 59] and those without access to the model's behavior [25, 41, 72].

Most of existing attacks focus on l_2 and l_∞ adversarial budgets, because they have clear mathematical formulations and are convex. Unfortunately, there is currently no universal adversarial attack framework for arbitrary adversarial budgets. Different adversarial budgets usually mean different methods, such as those for sparse attacks [34, 99] and Wasserstein

attacks [159].

The adversarial attack algorithm is crucial for evaluating the robustness of the models, because weak adversarial attacks overestimate the model's performance and give a false sense of robustness. To boost the strength of the adversarial attacks, AutoAttack [33] ensembles four attack methods of different categories to comprehensively evaluate the model's robustness. It is broadly used to reliably benchmark the performance of different defense algorithms [30].

2.2 Verified Robustness

Verified robustness enables us to guarantee an adversary-free region in the neighborhood of a given input instance by estimating the upper bound of the inner maximization problem in (1.2). Training deep neural networks to minimize this upper bound generates provably robust models. For robustness verification, we find a tighter upper bound of the inner maximal in (1.2) to verify more input instances. For training provably robust models, we need to find an appropriate loss objective function, not necessarily a tight bound, to facilitate training. Both problems are challenging for deep neural networks, because they are composed of many layers representing different functions, and because the nonlinear functions render their outputs highly non-convex functions of the inputs.

To obtain robustness guarantees, there are two categories of verifiers: *complete verifiers* and *incomplete verifiers*. Complete verifiers can either guarantee the absence of adversarial perturbations or find one, within the adversarial budget, to fool the model. These verifiers include methods based on *Satisfiability Modulo Theories (SMT)* [80] and based on *Mixed Integer Programming (MIP)* [143, 164]. However, complete verification is proven NP-hard [154], and all these methods have super-linear complexity. Therefore, as a complete verifier is used only for small models, we have to use an incomplete verifier for large models. Incomplete verifiers are much faster in deriving the bound of the models' outputs by approximations but, unlike with complete verifiers, once the incomplete verifier fails to verify it, we cannot guarantee the input instances are vulnerable within the adversarial budget. Different incomplete verifiers utilize different techniques to bound the models' outputs, including the linear approximation of the nonlinear activation function [9, 154, 157, 160, 167, 180], layerwise interval bound propagation [55, 179], semi-definite programming [111, 112], symbolic interval analysis [151] and abstract transformers [48, 136, 137]. Randomized smoothing [27, 169] is also broadly used as an incomplete verifier. Different from other methods, this method is probabilistic and can generate, by using Monte Carlo sampling, a robustness guarantee with high probability.

To train deep neural networks that are provably robust, [157] solves the outer minimization problem of (1.2) by its dual problem. *Interval Bound Propagation (IBP)* in [55] and *CROWN-IBP* in [179] directly minimize the upper bound of the inner maximal of (1.2). These methods are later improved by acceleration [132, 152] and more favorable loss landscape [90]. [123] theoretically discusses the inherent limitations of these linear programming (LP) based methods. In addition, [29, 94] use geometric methods to calculate the distance between the input

instance and the decision boundary in the input space. By introducing regularization schemes to encourage larger distances, we can obtain provably robust models. For methods based on randomized smoothing, [27] boost provable robustness by training against Gaussian noises. [122] improve the performance by direct adversarial training on the randomized smoothed model.

Although verified robustness can provide theoretical guarantees and thus suitable for highly safety-critical applications, the algorithm scalability is the major drawback. For deep neural network with millions of parameters, complete verifiers become infeasible, and the output bounds given the incomplete verifiers are increasingly looser with the increase of the network depth. Improving the efficiency of these methods and scaling them to adapt to the industry-sized models would be interesting yet challenging.

2.3 Empirical Robustness

Empirical robustness measures the performance of the models under the state-of-the-art adversarial attacks. Although models with strong empirical robustness performance can have poor verified robustness, empirical robustness, as the upper bound of true robustness, still has meaningful implications of models' performance under adversarial attacks in practice.

Many techniques have been proposed to improve the models' empirical robustness, including input denoising [61, 124, 140], randomized layers [36, 162, 165], special loss objectives [19, 108], manipulating intermediate features [103] and test-time adaptation [23, 131, 173]. Most of these methods, however, are later shown to use *obfuscated gradient* and *gradient masking*, instead of achieving true robustness [7, 31, 33]. In other words, these models are robust only against some specific types of attacks and vulnerable under a strong and adaptive attack. As a result, adversarial training [98] and its variants [3, 18, 65, 85, 161, 177] become the most popular methods that remain unbroken. In adversarial training, we first generate adversarial perturbations, usually by PGD [98], then, by using these perturbations, we optimize model parameters.

Despite effective, adversarial training is more challenging in many aspects. First, adversarially trained models usually have performance on the clean inputs worse than their counterparts trained on clean inputs, i.e., there is a trade-off between clean accuracy and robustness [178]. Furthermore, adversarial training converges more slowly [93] and needs longer training durations. The n step PGD attack needs n forward-backward passes of the network, hence the computational complexity of adversarial training using n step PGD attacks is $n+1$ times the classic empirical risk minimization with the same number of epochs. Moreover, the generalization gap of adversarial training is much larger [118], the performance on the test set degrades significantly in the late phase of training. To address this issue, we can use early stopping or parameter smoothing [21]. Last but not least, compared with performance on clean data, to achieve competitive performance empirical robustness, adversarial training needs more training data and a higher model capacity. Generating more synthetic data [57, 115] will

greatly improve the performance. When the model is small enough, adversarial training can fail to converge, whereas empirical risk minimization can still generate non-trivial models [98]; as the model becomes increasingly larger, the performance on clean inputs first saturates but the performance on adversarially-perturbed inputs still improves. All these challenges in adversarial training prevents it from large-scale deployment on large models and large datasets.

2.4 Efficient Robust Learning

In Chapter 1.3, we identify several challenges in robust learning. There are several efforts to overcome them and to make robust learning more efficient. Most of these works focus on empirical robustness, although verified robustness has similar issues.

The first line of works concentrate on accelerating adversarial training. To accelerate the parameter convergence in training, [130] uses batch replaying to update model parameters in each step of PGD. [158] further accelerates the training by using 1-step PGD and parameters of mixed precision. This method, however, sacrifices the training stability and can suffer from *catastrophic overfitting*. In other words, the models can suddenly overfit to the weak attack, which we use during training, hence it can fail to achieve true robustness. To avoid catastrophic overfitting, [6] proposes gradient alignment. [182] demonstrates that the adversarial perturbations are transferable between two consecutive epochs during training. In this regard, the authors proposes to initialize the perturbation by the one in the last epoch to accelerate and to stabilize adversarial training by using 1-step PGD. They show no catastrophic overfitting using this method.

The other line of works, to construct light-weighted but robust models, combine robust learning with model compression, including network pruning and quantization. [172] proposes alternating direction method of multipliers (ADMM) to alternatively update model parameters and the pruning mask. [58] extends this method to include other model compression techniques, such as quantization. Furthermore, [127] proposes *HYDRA*, a three-phase method for obtaining compressed yet robust models. The three phases include pre-training, score-based pruning, and fine-tuning.

In the next three chapters, we will study verified robustness, empirical robustness, and efficient robust learning. We will introduce some baseline methods in detail before introducing our proposed methods.

3 Verified Robustness

Using the robust learning formulation of (1.2), we can verify whether an input instance is robust against adversarial attacks within the adversarial budget by estimating the upper bound of the inner maximal. Tighter upper bounds lead to better verifiers, whereas smooth bounds can facilitate us to train provably robust neural networks [90]. In this chapter, we first review two categories of methods that estimate the bounds of the model's outputs. We then present two of our works: in the first one, we propose more general verifiers; and in the second, we focus on improving training provably robust neural networks.

The contents of this chapter are mainly from the following two papers. I am the primary contributior of both papers.

- Chen Liu, Ryota Tomioka, Volkan Cevher. "On Certifying Non-uniform Bounds against Adversarial Attacks." International Conference on Machine Learning 2019.
- Chen Liu, Mathieu Salzmann, Sabine, Süsstrunk. "Training Provably Robust Models by Polyhedral Envelope Regularization." IEEE Transactions on Neural Networks and Learning Systems 2021.

3.1 Bounding Network's Output

To obtain the upper bound of the inner maximization in problem (1.2), we first need to estimate the lower and the upper bound of the model's output. For K-category classification problem, we derive the upper bound $\overline{\mathbf{o}}$ and the lower bound $\underline{\mathbf{o}}$ of the model's output $f(\theta, \mathbf{x})$ given the adversarial budget $\mathcal{S}_\epsilon^{(p)}$, then, we can obtain the upper bound of the loss objective. This is formulated by the following inequality for the input instance $(\mathbf{x}, y)$:

$$\begin{aligned} &\text{if } \forall \Delta \in \mathcal{S}_\epsilon^{(p)},\ \underline{\mathbf{o}} \leq f(\theta, \mathbf{x}+\Delta) \leq \overline{\mathbf{o}}; \text{ then } \max_{\Delta \in \mathcal{S}_\epsilon^{(p)}} \mathcal{L}(f(\theta, \mathbf{x}+\Delta)) \leq \mathcal{L}(\mathbf{o}^{(worst)}, y), \\ &\text{where } \mathbf{o}_i^{(worst)} = \begin{cases} \underline{\mathbf{o}}_i & \text{if } i = y \\ \overline{\mathbf{o}}_i & \text{if } i \neq y \end{cases} \end{aligned} \tag{3.1}$$

(3.1) indicates that we can derive the upper bound of the loss objective under attack by plugging in the lower bound of output logits for the correct label and the upper bound of the rest. Therefore, the tightness of the bounds $\underline{\mathbf{o}}$, $\overline{\mathbf{o}}$ determines the tightness of the upper bound of $\max_{\Delta\in\mathcal{S}_\epsilon^{(p)}} \mathcal{L}(f(\theta,\mathbf{x}+\Delta))$.

In the following sections, we will introduce two popular methods we use to bound the output logits: linear approximation and interval bound propagation. For notation simplicity, we consider the feed forward layer here. The methods can be straightforwardly extended to general neural network architectures that can be represented by a *directed acyclic graph (DAG)* [95, 167]. Specifically, we consider an L-layer neural network parameterized by $\{\mathbf{W}^{(i)},\mathbf{b}^{(i)}\}_{i=1}^{L}$:

$$\begin{aligned} \mathbf{z}^{(i+1)} &= \mathbf{W}^{(i)}\widehat{\mathbf{z}}^{(i)}+\mathbf{b}^{(i)} \quad && i=1,2,...,L-1 \\ \widehat{\mathbf{z}}^{(i)} &= \Phi(\mathbf{z}^{(i)}) && i=2,3,...,L-1 \end{aligned} \tag{3.2}$$

where $\Phi(\cdot)$ is the activation function, $\widehat{\mathbf{z}}^{(1)}:=\mathbf{x}$ is the input. In addition, $\{\mathbf{z}^{(i)}\}_{i=2}^{L}$ and $\{\widehat{\mathbf{z}}^{(i)}\}_{i=2}^{L-1}$ represent the pre-activations and post-activations of each layer, respectively.

3.1.1 Linear Approximation

Linear approximation (LA) is proposed in Fast-Lin [154] for ReLU networks, it is then extended to general activation functions in CROWN [180]. Here, we mainly discuss the linear approximations that we will use later.

Since $\Phi(\cdot)$ is an element-wise function, we consider the scalar case: the input $x\in\mathbb{R}$. For an activation function $\Phi(\cdot)$ that is nonlinear and monotonically increasing, and the input x bounded by $\underline{x}\le x\le\overline{x}$, we can use two linear functions with the same slope to bound $\Phi(x)$ by the following inequality.

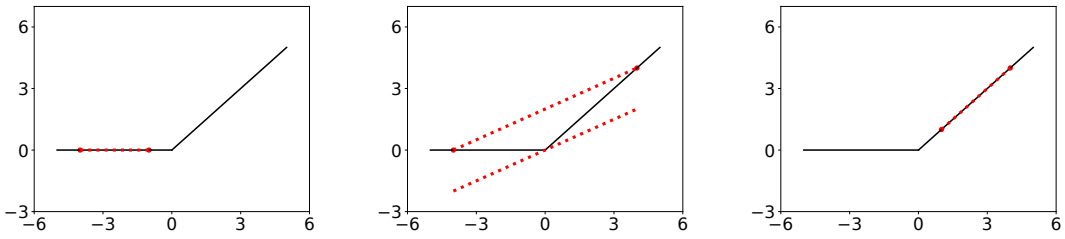
$$\forall\,\underline{x}\le x\le\overline{x},\ dx+\underline{a}\le\Phi(x)\le dx+\overline{a} \tag{3.3}$$

Here, the coefficient d, $\underline{a}$ and $\overline{a}$ all depend on the value of $\underline{x}$, $\overline{x}$ and ultimately the adversarial budget and the model parameters. They are chosen in a way such that the gap $\overline{a}-\underline{a}$ is minimized.

As shown in Figure 3.1, for ReLU function $\Phi(x)=\max(0,x)$, which is convex, we set the value of d, $\underline{a}$ and $\overline{a}$ as follows:

$$d=\begin{cases} 0 & \underline{x}\le\overline{x}\le 0 \\ \frac{\overline{x}}{\overline{x}-\underline{x}} & \underline{x}<0<\overline{x}\,; \\ 1 & 0\le\underline{x}\le\overline{x} \end{cases}\quad \underline{a}=0;\quad \overline{a}=\begin{cases} 0 & \underline{x}\le\overline{x}\le 0 \\ -\frac{\underline{x}\overline{x}}{\overline{x}-\underline{x}} & \underline{x}<0<\overline{x}\,. \\ 0 & 0\le\underline{x}\le\overline{x} \end{cases} \tag{3.4}$$

Unlike the ReLU function, the sigmoid function $\Phi(x)=\frac{1}{1+e^{-x}}$ and tanh function $\Phi(x)=\frac{e^{2x}-1}{e^{2x}+1}$



(a) $\underline{x} \leq \overline{x} \leq 0$ (b) $\underline{x} < 0 < \overline{x}$ (c) $0 \leq \underline{x} \leq \overline{x}$

Figure 3.1 – Linear approximation of the ReLU function in all scenarios. Both the upper and the lower bounds are represented by red dashed lines.

are not convex. However, these two functions are convex when $\underline{x} \leq \overline{x} \leq 0$ and concave when $0 \leq \underline{x} \leq \overline{x}$ (left and right sub-figures of Figure 3.2). Therefore, we can easily obtain a tight linear approximation in these cases. When $\underline{x} \leq 0 \leq \overline{x}$, we do not use the binary search to obtain a tight linear approximation as in [180], because the results will not have an analytical form in this way. We need an analytical form of d, $\underline{a}$ and $\overline{a}$ in order to use them for training model parameters. Instead, we first calculate the slope between the two ends, i.e., $d = \frac{\Phi(\overline{x})-\Phi(\underline{x})}{\overline{x}-\underline{x}}$. Then, we bound the function by two tangent lines of the same slope as d. As shown in Figure 3.2, when Φ is sigmoid or tanh function, we can calculate d, $\underline{a}$ and $\overline{a}$ as follows:

$$d = \frac{\Phi(\overline{x}) - \Phi(\underline{x})}{\overline{x} - \underline{x}}; \quad \underline{a} = \begin{cases} \Phi(t_1) - t_1 d & \underline{x} \leq \overline{x} \leq 0 \\ \frac{\overline{x}\Phi(\underline{x}) - \underline{x}\Phi(\overline{x})}{\overline{x}-\underline{x}} & 0 \leq \underline{x} \leq \overline{x} \end{cases}; \quad \overline{a} = \begin{cases} \frac{\overline{x}\Phi(\underline{x}) - \underline{x}\Phi(\overline{x})}{\overline{x}-\underline{x}} & \underline{x} \leq \overline{x} \leq 0 \\ \Phi(t_2) - t_2 d & 0 < \overline{x} \end{cases} \tag{3.5}$$

The coefficients $t_1 < 0 < t_2$ are the position of tangent points on both sides of the origin. The definitions of t_1 and t_2 for different activation functions are provided in Table 3.1.

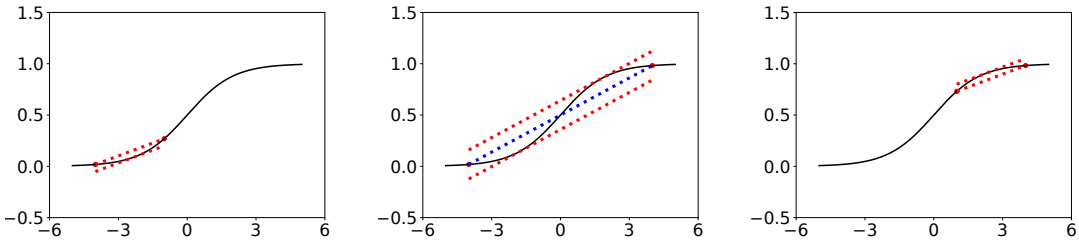
σ	Sigmoid	Tanh
t_1	$-\log \frac{-(2d-1)+\sqrt{1-4d}}{2d}$	$\frac{1}{2}\log \frac{-(d-2)-2\sqrt{1-d}}{d}$
t_2	$-\log \frac{-(2d-1)-\sqrt{1-4d}}{2d}$	$\frac{1}{2}\log \frac{-(d-2)+2\sqrt{1-d}}{d}$

Table 3.1 – Definition of t_1 and t_2 for different activation functions.

Now, we consider the matrix form and the ith layer of the neural network. Based on the formulation of (3.2), we consider the bound of the pre-activation $\underline{\mathbf{z}}^{(i)} \leq \mathbf{z}^{(i)} \leq \overline{\mathbf{z}}^{(i)}$, the post-activation $\widehat{\mathbf{z}}^{(i)}$ can then be bounded by $\mathbf{D}^{(i)}\mathbf{z}^{(i)} + \underline{\mathbf{a}}^{(i)} \leq \Phi(\mathbf{z}^{(i)}) \leq \mathbf{D}^{(i)}\mathbf{z}^{(i)} + \overline{\mathbf{a}}^{(i)}$ based on the linear approximation introduced above. Here, $\mathbf{D}^{(i)}$ is a diagonal matrix, $\underline{\mathbf{a}}^{(i)}$ and $\overline{\mathbf{a}}^{(i)}$ are both bias vectors. Equivalent to the bounds, we have the following claim:

$$\forall \mathbf{z}^{(i)} : \underline{\mathbf{z}}^{(i)} \leq \mathbf{z}^{(i)} \leq \overline{\mathbf{z}}^{(i)}, \exists \mathbf{a}^{(i)} : \underline{\mathbf{a}}^{(i)} \leq \mathbf{a}^{(i)} \leq \overline{\mathbf{a}}^{(i)} s.t.\ \Phi(\mathbf{z}^{(i)}) = \mathbf{D}^{(i)}\mathbf{z}^{(i)} + \mathbf{a}^{(i)} \tag{3.6}$$

(3.6) demonstrates that given the bound of the input, we can replace the nonlinear activation



(a) $\underline{x} \leq \overline{x} \leq 0$ (b) $\underline{x} < 0 < \overline{x}$ (c) $0 \leq \underline{x} \leq \overline{x}$

Figure 3.2 – Linear approximation of the Sigmoid function in all scenarios. Both the upper and the lower bounds are represented by red dashed lines.

function by a linear alternative with a bounded bias term. For an L-layer neural network defined in (3.2), we can iteratively bound the intermediate activation, and finally bound the output logits $\mathbf{z}^{(L)} := f(\theta, \mathbf{x}+\Delta)$ as follows:

$$\begin{aligned}
\mathbf{z}^{(L)} &= \mathbf{W}^{(L-1)}(\Phi(\mathbf{W}^{(L-2)}(...(\mathbf{W}^{(1)}(\mathbf{x}+\Delta)+\mathbf{b}^{(1)})...)+\mathbf{b}^{L-2}))+\mathbf{b}^{(L-1)} \\
&= \mathbf{W}^{(L-1)}(\mathbf{D}^{(i-1)}(\mathbf{W}^{(i-2)}(...(\mathbf{W}^{(1)}(\mathbf{x}+\Delta)+\mathbf{b}^{(1)})...)+\mathbf{b}^{(i-2)})+\mathbf{a}^{(i-1)})+\mathbf{b}^{(i-1)} \\
&= \left(\Pi_{j=2}^{L-1}\mathbf{W}^{(j)}\mathbf{D}^{(j)}\right)\mathbf{W}^{(1)}\Delta + \sum_{i=2}^{L-1}\left(\Pi_{j=i+1}^{L-1}\mathbf{W}^{(j)}\mathbf{D}^{(j)}\right)\mathbf{W}^{(i)}\mathbf{a}^{(i)} \\
&+ \left(\Pi_{j=2}^{L-1}\mathbf{W}^{(j)}\mathbf{D}^{(j)}\right)\mathbf{W}^{(1)}\mathbf{x} + \sum_{i=1}^{L-1}\left(\Pi_{j=i+1}^{:-1}\mathbf{W}^{(j)}\mathbf{D}^{(j)}\right)\mathbf{b}^{(i)}
\end{aligned} \tag{3.7}$$

Algorithm 3.1: Bound the output logits by linear approximation.

1: Input: parameters $\{\mathbf{W}^{(i)}, \mathbf{b}^{(i)}\}_{i=1}^{L-1}$, input instance $\mathbf{x}$, adversarial budget $\mathbb{S}_\epsilon^{(p)}$.
2: $\underline{\mathbf{z}}^{(2)} = \mathbf{W}^{(1)}\mathbf{x}+\mathbf{b}^{(1)} - \epsilon\|\mathbf{W}^{(1)}\|_{:,q}$; $\overline{\mathbf{z}}^{(2)} = \mathbf{W}^{(1)}\mathbf{x}+\mathbf{b}^{(1)} + \epsilon\|\mathbf{W}^{(1)}\|_{:,q}$.
3: **for** $k = 2, ..., L-1$ **do**
4: Calculate $\mathbf{D}^{(k)}$, $\underline{\mathbf{a}}^{(k)}$ and $\overline{\mathbf{a}}^{(k)}$ based on $\underline{\mathbf{z}}^{(k)}$, $\overline{\mathbf{z}}^{(k)}$ and (3.6).
5: $\widetilde{\mathbf{z}}^{(k+1)} = \left(\Pi_{j=2}^{k}\mathbf{W}^{(j)}\mathbf{D}^{(j)}\right)\mathbf{W}^{(1)}\mathbf{x} + \sum_{i=1}^{k}\left(\Pi_{j=i+1}^{:-1}\mathbf{W}^{(j)}\mathbf{D}^{(j)}\right)\mathbf{b}^{(i)}$.
6: $\widetilde{\mathbf{W}}^{(i)} = \left(\Pi_{j=i+1}^{k}\mathbf{W}^{(j)}\mathbf{D}^{(j)}\right)\mathbf{W}^{(i)}$ for $i = 1, 2, ..., k$.
7: $\underline{\mathbf{z}}^{(k+1)} = \widetilde{\mathbf{z}}^{(k+1)} - \epsilon\|\widetilde{\mathbf{W}}^{(1)}\|_{:,q} + \sum_{i=2}^{k}\left(|\widetilde{\mathbf{W}}^{(i)}|_+\underline{\mathbf{a}}^{(i)} + |\widetilde{\mathbf{W}}^{(i)}|_-\overline{\mathbf{a}}^{(i)}\right)$.
8: $\overline{\mathbf{z}}^{(k+1)} = \widetilde{\mathbf{z}}^{(k+1)} + \epsilon\|\widetilde{\mathbf{W}}^{(1)}\|_{:,q} + \sum_{i=2}^{k}\left(|\widetilde{\mathbf{W}}^{(i)}|_-\underline{\mathbf{a}}^{(i)} + |\widetilde{\mathbf{W}}^{(i)}|_+\overline{\mathbf{a}}^{(i)}\right)$.
9: **end for**
10: Output $\underline{\mathbf{z}}^{(L)}$ and $\overline{\mathbf{z}}^{(L)}$.

The right hand side of Equation (3.7) is a linear function of Δ and $\{\mathbf{a}^{(i)}\}_{i=1}^{L-1}$. Based on the constraints $\|\Delta\|_p \leq \epsilon$ and $\underline{\mathbf{a}}^{(i)} \leq \mathbf{a}^{(i)} \leq \overline{\mathbf{a}}^{(i)}$, we can then bound the value of the output logits $\mathbf{z}^{(L)}$, a.k.a, $f(\theta, \mathbf{x}+\Delta)$. We then summarize the linear approximation as Algorithm 3.1. $|\mathbf{W}|$ is the element-wise absolute value of $\mathbf{W}$; $|\mathbf{W}|_-$, $|\mathbf{W}|_+$ are the negative elements and the positive elements of the matrix $\mathbf{W}$, respectively; $\|\mathbf{W}\|_{:,q}$ calculate the l_q norm of each row in the matrix

$\mathbf{W}$ where l_q is the dual norm of l_p, i.e., $\frac{1}{p}+\frac{1}{q}=1$.

The approximation error of Algorithm 3.1 mainly arises from approximating the nonlinear activation functions by linear functions. Since Algorithm 3.1 estimates the bounds of the intermediate activations by contributions of all its preceding layers, it needs $\mathcal{O}(L^2)$ matrix multiplications for one input instances. This makes the linear approximation method much more computationally expensive than the normal feed forward pass, which consumes $\mathcal{O}(L)$ matrix multiplications. The interval bound propagation introduced in the following section can mitigate the computational complexity issue.

3.1.2 Interval Bound Propagation

Interval bound propagation (IBP) [55] is an efficient method to obtain the bounds of neural network's output. Unlike linear approximation, IBP estimate the bounds of intermediate activation only based on its immediate previous layer. Thanks to the monotonicity of the activation function $\Phi(\cdot)$, we can bound $\underline{\mathbf{z}}^{(i+1)}$, $\overline{\mathbf{z}}^{(i+1)}$ by the following equations:

$$\begin{aligned}\underline{\mathbf{z}}^{(i+1)} &= |\mathbf{W}^{(i)}|_+\Phi(\underline{\mathbf{z}}^{(i)}) + |\mathbf{W}^{(i)}|_-\Phi(\overline{\mathbf{z}}^{(i)}) + \mathbf{b}^{(i)}\\ \overline{\mathbf{z}}^{(i+1)} &= |\mathbf{W}^{(i)}|_-\Phi(\underline{\mathbf{z}}^{(i)}) + |\mathbf{W}^{(i)}|_+\Phi(\overline{\mathbf{z}}^{(i)}) + \mathbf{b}^{(i)}\end{aligned} \tag{3.8}$$

For the input layer, we have $\underline{\mathbf{z}}^{(2)} = \mathbf{W}^{(1)}\mathbf{x}+\mathbf{b}^{(1)} - \epsilon\|\mathbf{W}^{(1)}\|_{:,q}$ and $\overline{\mathbf{z}}^{(2)} = \mathbf{W}^{(1)}\mathbf{x}+\mathbf{b}^{(1)} + \epsilon\|\mathbf{W}^{(1)}\|_{:,q}$ under the adversarial budget $\mathcal{S}_\epsilon^{(p)}$. Then, we can iteratively calculate the bounds of intermediate activations by (3.8) until we obtain the bounds $\underline{\mathbf{z}}^{(L)}$, $\overline{\mathbf{z}}^{(L)}$ of the output logits $\mathbf{z}^{(L)}$.

We provide the pseudo-code of IBP as Algorithm 3.2 below. It is clear that IBP needs only $\mathcal{O}(L)$ matrix multiplications for one input instances, comparable to the feed forward pass.

Algorithm 3.2: Bound the output logits by interval bound propagation.

Input: parameters $\{\mathbf{W}^{(i)}, \mathbf{b}^{(i)}\}_{i=1}^{L-1}$, input instance $\mathbf{x}$, adversarial budget $\mathcal{S}_\epsilon^{(p)}$.
$\underline{\mathbf{z}}^{(2)} = \mathbf{W}^{(1)}\mathbf{x}+\mathbf{b}^{(1)} - \epsilon\|\mathbf{W}^{(1)}\|_{:,q}$.
$\overline{\mathbf{z}}^{(2)} = \mathbf{W}^{(1)}\mathbf{x}+\mathbf{b}^{(1)} + \epsilon\|\mathbf{W}^{(1)}\|_{:,q}$.
for $k = 2, ..., L-1$ **do**
    $\underline{\mathbf{z}}^{(i+1)} = |\mathbf{W}^{(i)}|_+\Phi(\underline{\mathbf{z}}^{(i)}) + |\mathbf{W}^{(i)}|_-\Phi(\overline{\mathbf{z}}^{(i)}) + \mathbf{b}^{(i)}$.
    $\overline{\mathbf{z}}^{(i+1)} = |\mathbf{W}^{(i)}|_-\Phi(\underline{\mathbf{z}}^{(i)}) + |\mathbf{W}^{(i)}|_+\Phi(\overline{\mathbf{z}}^{(i)}) + \mathbf{b}^{(i)}$.
end for
Output $\underline{\mathbf{z}}^{(L)}$ and $\overline{\mathbf{z}}^{(L)}$.

Compared with linear approximation, IBP is faster and thus has better scalability. In terms of bound tightness, IBP does not make any approximations of the activation function. However, for linear layers, the IBP bounds are calculated only based on the immediate preceding layer, this "coarse bound" makes the approximation error accumulate much faster than linear approximation with the increase of the network depth L. To conclude, the linear approximation and IBP are complementary to each other. Either one is not guaranteed to give

a better bound than the other.

When the loss function $\mathcal{L}$ is the softmax cross-entropy function, what matters is the margin between the logits corresponding to the correct label and the ones corresponding to the incorrect labels. That is $\mathcal{L}(\mathbf{o}, y) = \mathcal{L}(\mathbf{o} - \mathbf{o}_y, y)$, we can then define $\mathbf{o} - \mathbf{o}_y := \mathbf{W}^{(out)}\mathbf{o}$ where $\mathbf{W}^{(out)} = \mathbf{I} - \mathbf{1}_y$ is a matrix depending on the label y. Here, $\mathbf{I}$ is the identity matrix, and $\mathbf{1}_y$ is the matrix where the elements in y-th column are 1 and otherwise 0. To obtain a tighter bound of the loss objective, we can then merge the matrix $\mathbf{W}^{(out)}$ with the linear operation in the last layer. This is called *elision of the last layer*. It is applicable not only for IBP but also for linear approximation methods introduced in Section 3.1.1.

3.2 Network Verification

We now study the robustness verification problem. In contrast to the methods in Section 2.2 that seek verified regions bounded uniformly along all features of the input instance, we consider non-uniform bounds and use it to study the decision boundaries of the neural networks. In this section, we first introduce the motivation and methods on verifying non-uniform bounds. In addition, we show that the non-uniform bounds not only covers much larger volumes than uniform bounds but also can distinguish non-robust features from robust ones. The latter can facilitate us to analyze the model's decision boundary and interpretability.

3.2.1 Verification on Non-uniform Bounds

As pointed out in [145], input features have different levels of robustness. We consider each dimension of the input instances as one feature and would like identify which features are non-robust features. Intuitively, non-robust features are what the model relies on to make prediction, because changes in these features are more likely to change the model's prediction. That is to say, identifying non-robust features is also identifying key features for the model to make prediction. By comparing these key features with human perception, we can better understand the behavior of different deep neural network models and check if they are interpretable.

We identify non-robust features by verifying a *non-uniform* adversary-free neighborhood of the input instance, because non-robust features should have smaller perturbation tolerance than robust ones. All previous works on robustness verification [154, 160, 180] aim to find the largest *uniform* bounds of the adversary-free region, we are the first to extend this to the *non-uniform* case.

We need a different mathematical definition of the adversarial budget for non-uniform bounds. Since the adversarial budget $\mathbb{S}_\epsilon^{(p)}$ corresponding to a uniform bound can be formulated as $\mathbb{S}_\epsilon^{(p)} = \{\Delta = \epsilon\mathbf{v} | \|\mathbf{v}\|_p \leq 1\}$, we can parameterize the l_p norm based non-uniform bounds $\mathbb{S}_{\vec{\epsilon}}^{(p)}$ by a vector $\vec{\epsilon} \in \mathbb{R}^M$ with the same dimension as the input instance: $\mathbb{S}_{\vec{\epsilon}}^{(p)} = \{\Delta = \vec{\epsilon} \odot \mathbf{v} | \|\mathbf{v}\|_p \leq 1\}$

where $\odot$ is the element-wise multiplication.

Now, we would like to find the certified region $\mathbb{S}_{\vec{\epsilon}}^{(p)}$ of the largest volume, measured by $\Pi_i \vec{\epsilon}_i$, in which the model outputs consistent label. Formally, for an input instance $\mathbf{x}$ with label y, the problem we focus on is formulated below:

$$\begin{aligned} \min_{\vec{\epsilon}} \quad & \left\{ -\sum_i \log \vec{\epsilon}_i \right\} \\ \underline{\mathbf{z}}_y^{(L)} - \overline{\mathbf{z}}_j^{(L)} \geq \delta \quad & \qquad j = 0, 1, ..., K-1; j \neq c \end{aligned} \tag{3.9}$$

As a common practice, we minimize the negative logarithm of $\Pi_i \vec{\epsilon}_i$ because it is convex and more stable numerically. $\underline{\mathbf{z}}^{(L)}$ and $\overline{\mathbf{z}}^{(L)}$ are the lower bound and the upper bound calculated by Algorithm 3.1 in Section 3.1.1. δ is a small positive constant to make sure the lower bound of the output logits of the true label is strictly higher than the upper bound of others.

We then reformulate the problem as follows:

$$\begin{aligned} \min_{\vec{\epsilon}, \mathbf{c} \geq 0} \quad & \left\{ -\sum_i \log \vec{\epsilon}_i \right\} \\ \underline{\mathbf{z}}_y^{(L)} - \overline{\mathbf{z}}_{j \neq y}^{(L)} - \delta = \mathbf{c} \end{aligned} \tag{3.10}$$

Here, $\overline{\mathbf{z}}_{j \neq y}^{(L)} \in \mathbb{R}^{K-1}$ is the concatenation of the upper bound of all output logits except the one corresponding to the correct label y. $\mathbf{c} (\geq 0) \in \mathbb{R}^{K-1}$ is a slack variable that ensures the non-negativity of the term on the left hand side. For notation simplicity, we define $\mathbf{d} := \underline{\mathbf{z}}_y^{(L)} - \overline{\mathbf{z}}_{j \neq y}^{(L)} - \delta$. $\mathbf{d}$ is a function of $\vec{\epsilon}$, because both $\underline{\mathbf{z}}^{(L)}$ and $\overline{\mathbf{z}}^{(L)}$ depend on $\vec{\epsilon}$.

We can further rewrite the problem (3.10) into a min-max problem using augmented Lagrangian method [66, 110] by introducing the dual variable $\boldsymbol{\lambda} \in \mathbb{R}^{K-1}$ and the coefficient $\rho \in \mathbb{R}^+$. The dual problem to solve is below:

$$\max_{\boldsymbol{\lambda}} \min_{\vec{\epsilon}, \mathbf{c} \geq 0} \left(-\sum_i \log \vec{\epsilon}_i \right) + \langle \boldsymbol{\lambda}, \mathbf{d} - \mathbf{c} \rangle + \frac{\rho}{2} \| \mathbf{d} - \mathbf{c} \|_2^2 \tag{3.11}$$

The inner minimization problem is a quadratic form of $\mathbf{c}$, so the optimal $\mathbf{c}$ has the analytical solution: $\mathbf{c} = \max(0, \mathbf{d} + \frac{1}{\rho}\boldsymbol{\lambda})$. Plug this solution in the problem (3.11), and we can then optimize $\vec{\epsilon}$ by gradient descent. We provide the pseudo-code as Algorithm 3.3 below.

Similar to penalty methods, the coefficient $\{\rho^{(i)}\}_{i=1}^M$ in Algorithm 3.3 is a non-decreasing sequence to enforce the constraint $\mathbf{d} = \mathbf{c}$. However, the Lagrange multiplier term $\langle \boldsymbol{\lambda}, \mathbf{d} - \mathbf{c} \rangle$ makes it unnecessary to increase $\rho^{(i)}$ to $+\infty$. Actually, $\rho^{(i)}$ can stop at a relatively small value here to solve the problem, which avoids numerical instability caused by ill-conditioning.

The minimization in line 5 of Algorithm 3.3 is solved by gradient methods, such as SGD and

Algorithm 3.3: Optimizing $\vec{\epsilon}$

1: Input: Parameters $\{\mathbf{W}^{(i)}, \mathbf{b}^{(i)}\}_{i=1}^{L-1}$, original bounds $\vec{\epsilon}_0$, iterations N_{iter}, augmented coefficient $\{\rho^{(i)}\}_{i=1}^{N_{iter}}$, decaying factor η.
2: Initialization: $\vec{\epsilon} = \vec{\epsilon}_0$, $\boldsymbol{\lambda} = 0$
3: **for** $i = 1, 2, ..., N_{iter}$ **do**
4: $\quad \mathbf{c} = \max(0, \mathbf{d} + \frac{1}{\rho}\boldsymbol{\lambda})$
5: $\quad$ Update $\vec{\epsilon}$ by minimizing (3.11) with optimal $\mathbf{c}$ plugged in.
6: $\quad \boldsymbol{\lambda} = \boldsymbol{\lambda} + \rho^{(i)}(\mathbf{d} - \mathbf{c})$
7: **end for**
8: **while** $\mathbf{d} \geq 0$ is not satisfied **do**
9: $\quad \vec{\epsilon} = \eta\vec{\epsilon}$
10: **end while**
11: Output: $\vec{\epsilon}$

Adam [81]. In practice, gradient explosion might happen when $\vec{\epsilon}$ is small or $\rho^{(i)}$ is big. To avoid overshooting, we apply gradient rescaling to constrain the l_2 norm of the gradient. The term $\log\epsilon_i$ in problem 3.11 implicitly constrains ϵ_i to be positive, so we reparametrize $\epsilon_i = \boldsymbol{\zeta}_i^2$ and optimize vector $\boldsymbol{\zeta}$ instead.

The last while-loop in line 8 of Algorithm 3.3 is to ensure the output $\vec{\epsilon}$ meets the hard constraints $\underline{\mathbf{z}}_y^{(L)} - \overline{\mathbf{z}}_{j\neq y}^{(L)} - \delta \geq 0$. The decaying factor η is close to 1 and is set 0.99 in practice. When $\rho^{(i)}$ is large, the while-loop would break after very few iterations.

3.2.2 Experiments and Analysis

We validate our method of verifying non-uniform bounds in this section. In addition, we analyze and compare the robust models with their non-robust counterparts. The code are publicly available. [1]

Comparison between Uniform and Non-uniform Bounds

Algorithm 3.3 enables us to efficiently find a verified adversary-free non-uniform bound $\mathbb{S}_{\vec{\epsilon}}^{(p)}$. By letting $\vec{\epsilon} = \epsilon\mathbf{1}$ where $\mathbf{1}$ is an all-one vector, we can use Algorithm 3.3 to optimize the scalar ϵ, and we will obtain a uniform bound.

Our first experiment is on a 2D synthetic data. We generate 10 random 2D data points in the space of $[-1,1]^2$ labeled $\{0,1,...,9\}$ as seeds. Another 10000 random points in $[-1,1]^2$ are then generated and assigned the same label as the closest seeds. 90% of these data instances are for training and the rest are for testing.

We use a ReLU fully-connected neural network with two hidden layers of 10 neurons. Since

[1] https://github.com/liuchen11/CertifyNonuniformBounds

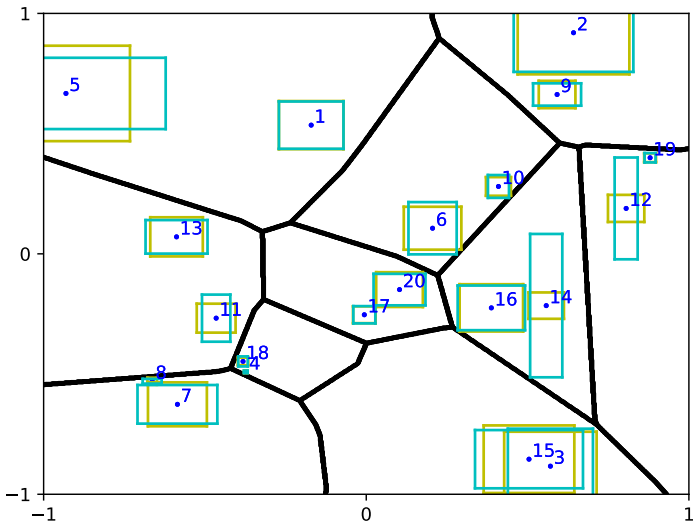


Figure 3.3 – A simple example on the synthetic data. 20 points with their uniform (yellow) and non-uniform (blue) bounds are shown above. Black lines are real decision boundaries.

the decision boundaries of this model are piece-wise linear in this case, the model is shown to have enough capacity and achieve an accuracy of more than 99.9% in the test set.

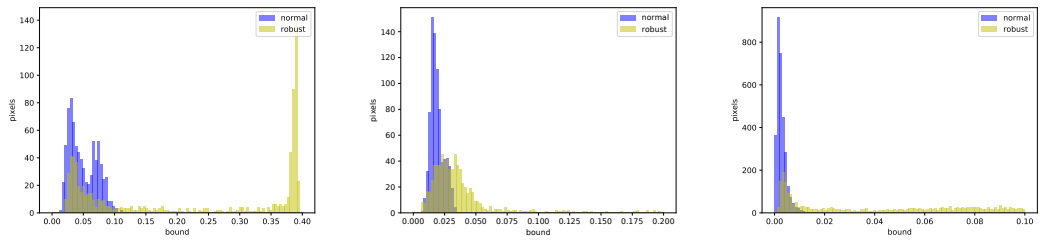
Figure 3.3 demonstrates the results of uniform (yellow) and non-uniform (blue) bounds with 20 random points in each category. We can clearly see the bounds calculated by our algorithm are reasonably tight and areas covered by non-uniform bounds are larger than those of uniform bounds. Although the larger volumes do not necessarily mean the bounds are larger for both features, bounds of some feature are extended in compensation for the other. In some cases, the non-uniform bounds can be significantly larger than uniform bounds (e.g. point 12, 14). This typically means considerable difference in robustness between two input features.

Then, we run our algorithm on real datasets, including MNIST [89], Fashion-MNIST [163] and SVHN [104]. They are benchmarks for image classification and contain tens of thousand images. MNIST and Fashion-MNIST are 28×28 gray-scale images, while SVHN are 32×32 colored images. Unless specified, all pixel values of these images are normalized in the range of $[-1, 1]$.

We study both robust models and non-robust models. Specifically, we obtain non-robust models by empirical risk minimization in (1.1) and robust models by PGD adversarial training [98]. For the latter, we use l_∞ based adversarial budget with $\epsilon = 0.1$. All the models are feed forward networks with three hidden layers. For MNIST, the number of hidden neurons in each layer are 100, 300 and 500; Fashion MNIST and SVHN models have 1024 neurons in each hidden layer. To evaluate the volume of the verified regions, we use the geometric average of bounds among all features, which is $\epsilon_{\text{avg}} = \left(\Pi_{i=1}^{M} \vec{\epsilon}_i\right)^{1/M}$. For uniform bounds $\mathcal{S}_\epsilon^{(p)}$, $\epsilon_{\text{avg}} = \epsilon$.

Dataset	Hidden Neurons	Model Type	Test Accuracy (%)	Uniform ϵ_{avg}	Non-uniform ϵ_{avg}	Ratio
MNIST	100	Vanilla	99.2	0.0295	0.0349	1.183
		Robust	98.1	0.0692	0.1678	2.425
	300	Vanilla	98.0	0.0309	0.0350	1.133
		Robust	98.9	0.0507	0.1404	2.769
	500	Vanilla	98.5	0.0319	0.0360	1.129
		Robust	98.8	0.0436	0.1167	2.677
Fashion MNIST	1024	Vanilla	90.4	0.0134	0.0141	1.052
		Robust	88.4	0.0208	0.0306	1.468
SVHN	1024	Vanilla	84.3	0.0022	0.0072	3.273
		Robust	78.2	0.0054	0.0281	5.204

Table 3.2 – Average uniform and non-uniform bounds in the test sets. Hidden neurons means the number of neurons in each hidden layer. Test accuracy is the accuracy on the clean test set. The ratio is the values of ϵ_{avg} for the non-uniform bounds over the ones for uniform bounds.



(a) MNIST (b) Fashion-MNIST (c) SVHN

Figure 3.4 – The distributions of bounds per feature for normal and robust models among different datasets in a randomly picked image when we are verifying by non-uniform bounds.

We report our results on the test set of each dataset under different settings in Table 3.2. It clearly shows that the non-uniform bounds can verify a larger regions compared with their uniform counterparts. In addition, We notice that the ratio of ϵ_{avg} for non-uniform bounds over the one for uniform bounds is significantly larger in the cases of robust models. Figure 3.4 demonstrates the distributions of bounds per feature for normal and robust models among different datasets in a randomly picked image when we are verifying by non-uniform bounds Compared with the vanilla models, the bounds of some features for the robust models can be much larger than the size of the adversarial budget used during training. This observation means the decision boundary of the robust model is almost aligned in some dimensions corresponding to some features, which makes it possible for our algorithm to extend the bounds of those features without sacrificing the bounds of the others much. It also implies robust models tend to drop irrelevant features and rely on fewer features when making predictions.

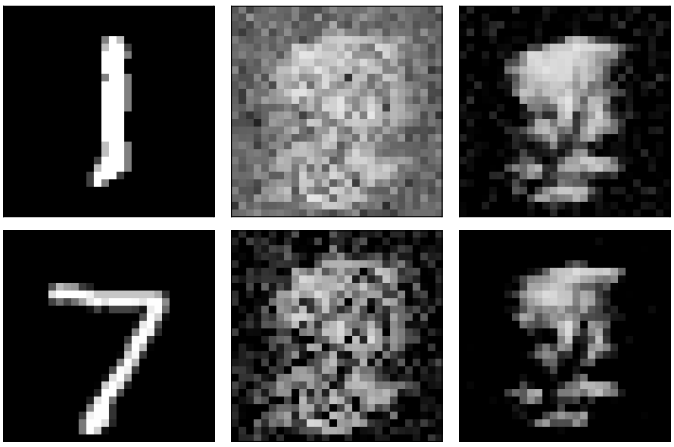


Figure 3.5 – An example of bounding maps of images of digit '1' and '7'. For bounding maps, lighter pixels indicate a smaller bound. (Left) The original images, (Middle) Bounding map of a normal model. (Right) Bounding map of a robust model.

Non-uniform Bounds and Model Interpretability

Given an input instance and a neural network model, we obtain a non-uniform bound parameterized by $\vec{\epsilon} \in \mathbb{R}^M$ by Algorithm 3.3. Unlike synthetic data, we cannot visualize the bounds as in Figure 3.3 for high dimensional input instances. However, we can visualize $\vec{\epsilon}$ just like images and call them *bounding maps*.

To study the properties of bounding maps, we take a simple example of binary classification: to distinguish digit '1' from '7' in MNIST dataset. We use the model with 100 neurons in each hidden layer and visualize the bounding maps of both models for two example images in Figure 3.5.

As shown in Figure 3.5, for the vanilla model, the bounding maps are noisy and can hardly reveal the patterns of the input instances. However, for the robust model, the bounding maps can capture some intrinsic characteristics of the input instances. In the case of digit '1' and '7', people typically distinguish them by the horizontal stroke which digit '7' has and '1' does not. This corresponds to the relatively smaller bounds of features in the middle above of the images. It indicates the decision boundaries are closer to the data points in the directions of these features and the model puts more weight on these features to make predictions. On the contrary, both digit '1' and '7' have a vertical stroke, we can correspondingly see a dark clear vertical stroke in the bounding maps of the robust model.

We need to mention similar property of robust models is found in [145] but from a more microscopic perspective. [145] visualizes the gradients of the model's loss function w.r.t the input data and finds that the gradients for robust models are *significantly more interpretable*, while the gradients for normal models are generally noise. Our investigations are more on a macroscopic level, our non-uniform bounds explore the shape of model's decision boundary

Dataset	Hidden Neurons	Model Type	Mean Cosine	Minimum Cosine
MNIST	100	Vanilla	0.9548	0.2304
		Robust	0.9957	0.9155
	300	Vanilla	0.9774	0.5038
		Robust	0.9964	0.9104
	500	Vanilla	0.9874	0.6367
		Robust	0.9941	0.8920
Fashion MNIST	1024	Vanilla	0.9805	0.5652
		Robust	0.9752	0.7166
SVHN	1024	Vanilla	0.9836	0.7129
		Robust	0.9952	0.9339

Table 3.3 – Mean and minimum cosine similarity of $\vec{\epsilon}$ among the dataset for different models.

but we have the same claim: *robust models are more interpretable*.

In addition, in Figure 3.5, we notice similar patterns of bounding maps found for the same model but different input instances. Therefore, we calculate the cosine similarity of $\vec{\epsilon}$ for two images, since the direction of $\vec{\epsilon}$ indicates the shape of non-uniform bounds. For all models presented in Table 3.2, we report the average and minimum values of cosine similarity among all input instance pairs in Table 3.3. It is clear that the values of $\vec{\epsilon}$ for different images but the same model are highly correlated, which indicates the geometric similarity of non-uniform bounds. [2] What's more, such correlation is even stronger in the cases of robust models.

The high correlation means some features are consistently more robust than the other features. For a given model, the values of $\vec{\epsilon}$ for most input instances are almost collinear, so the direction of $\vec{\epsilon}$ can be regarded as a quantitative and data-agnostic metric measuring the robustness of input features. It is also beneficial to use this direction for initialization, i.e., $\vec{\epsilon}_0$ in Algorithm 3.3, when we estimate the non-uniform bound for a new data point.

Since the shape of the non-uniform bound reveals the decision boundary, high correlation of $\vec{\epsilon}$ also indicates the uniformity of the direction of the decision boundary. That is to say, in an M dimensional input space, there exists a subspace $\mathcal{X}$ of dimensionality $M' \ll M$ that contains most directions of decision boundary around the data manifold. This is consistent with what [101] points out.

An extreme example is the classifier whose decision boundary is linear, the non-uniform bound of the largest volume for any input data has exactly the same shape. That is to say, the values of $\vec{\epsilon}$ for any input data point are exactly collinear and $M = 1$ in this case. Our experimental results show the stronger correlation of $\vec{\epsilon}$ in the cases of robust models. This implies the most directions of a robust model's decision boundary can be obtained in a subspace of even lower dimensionality than a non-robust model. That is to say, the decision boundary of a robust model should be simpler in some sense.

[2] For two random vectors uniformly distributed in $[0,1]^{784}$, the expectation of cosine similarity between them is around 0.75. The expectation decreases for random vectors in higher dimensions.

3.3 Training Provably Robust Networks

Section 3.2 explores the model's decision boundaries by verifying a non-uniform bounds. In this section, we will utilize geometric properties of the decision boundaries to train provably robust networks [94]. We first introduce a polyhedral envelope to bound the decision boundaries, which enables us to calculate a lower bound of the input instances' distance to the decision boundaries. This lower bound, which is differentiable w.r.t. the model parameters, can not only accelerate the verification but also construct a regularization scheme to train provably robust networks. In addition, our methods are generally applicable and shown to achieve provable robustness without sacrificing the clean accuracy too much.

3.3.1 Geometric Bounds of Decision Boundaries

Based on the linear approximation introduced in Section 3.1.1, we can derive the bounds of the output of the neural network given an adversarial budget. If we consider the constraints of Δ and $\{\mathbf{a}_{i=2}^{L-1}\}$ in Equation (3.7), we can derive two linear functions of the input instance $\mathbf{x}$ representing the upper and the lower bounds of the model's outputs $\mathbf{z}^{(L)}$.

In Section 3.1.1, we point out the issue of high computational complexity of the linear approximation. Alternatively, we can derive the linear bounds of the model's output based on interval bound propagation (IBP) introduced in Section 3.1.2. We first use IBP (Algorithm 3.2) to estimate the lower and the upper bound of each intermediate layers. For nonlinear activation functions, we can then linearize them by (3.3). By replacing the these nonlinear functions by their linear alternatives, we can finally obtain two linear functions of the input instances representing the upper and the lower bound of the model's outputs. It is straightforward that this method have the same computational complexity as IBP in terms of matrix multiplication. We call the linear bounds by these two methods *LA-inspired bounds* and *IBP-inspired bounds*, respectively.

Recall that correctly classifying the input instance $(\mathbf{x}, y)$ means $\mathbf{z}_y^{(L)} - \mathbf{z}_i^{(L)} > 0$ for $\forall i \neq y$, we use *the elision of the last layer* introduced in Section 3.1.2 to derive the linear lower bound of $\mathbf{z}_y^{(L)} - \mathbf{z}_i^{(L)}$ as follows:

$$\mathbf{z}_y^{(L)} - \mathbf{z}_i^{(L)} \geq \underline{\mathbf{z}_y^{(L)} - \mathbf{z}_i^{(L)}} := \mathbf{U}_i \mathbf{x} + \mathbf{p}_i. \tag{3.12}$$

Here, $\mathbf{U}$ and $\mathbf{p}$ are calculated either by LA-inspired bounds or IBP-inspired bounds. Based on this linear bound, if we have $\forall i \neq y$, $\mathbf{U}_i \mathbf{x} + \mathbf{p}_i > 0$, then we have $\forall i \neq y$, $\mathbf{z}_y^{(L)} - \mathbf{z}_i^{(L)} > 0$, which is exactly the conditions for correct classification. That is to say $\forall i \neq y$, $\mathbf{U}_i \mathbf{x} + \mathbf{p}_i > 0$ is the sufficient condition for robustness verification. This sufficient condition represents of $K-1$ hyperplanes in the input space. Within the adversarial budget, these hyperplanes provide a polyhedral envelope of the true decision boundaries.

To facilitate analysis, we use d_{iy} to represent the distance between the input instance $\mathbf{x}$ and the hyperplane $\mathbf{U}_i \mathbf{x} + \mathbf{p}_i > 0$. Furthermore, we use $d_y := \min_{i \neq y} d_{iy}$ to denote the distance between

the input instance $\mathbf{x}$ and the polyhedral envelope's boundary. Note that these distances can be defined on arbitrary l_p norm and $d_y = 0$ if the sufficient condition is not satisfied. Consider that the linear bounds in (3.12) are based on the adversarial budget $\mathbb{S}_\epsilon^{(p)}$, we can conclude that there is no adversarial examples that change the model's prediction in the intersection of the polyhedral envelope and the set of allowable perturbed inputs defined by the adversarial budget.

The lemma below formalizes the vanilla case of our robustness certification, when there are no additional constraints on the input.

Lemma 3.1 *Given a model f and an input instance $(\mathbf{x}, y)$, let $\mathbf{U}$ and $\mathbf{p}$ in (3.12) be calculated using a predefined adversarial budget $\mathbb{S}_\epsilon^{(p)}$. Then, there is no adversarial example inside an l_p norm ball of radius d centered around $\mathbf{x}$, with $d = \min\{\epsilon, d_y\}$, where $d_{iy} = \max\left\{0, \frac{\mathbf{U}_i\mathbf{x}+\mathbf{p}_i}{\|\mathbf{U}_i\|_q}\right\}$, $d_y = \min_{i\neq y} d_{iy}$. l_q is the dual norm of the l_p norm, i.e., $\frac{1}{p} + \frac{1}{q} = 1$.*

Proof: Let $\mathbf{x}' = \mathbf{x} + \Delta$ be a point that breaks condition (3.12). Then,

$$\begin{aligned} & \mathbf{U}_i(\mathbf{x}+\Delta) + \mathbf{p}_i && < 0 \\ \iff\ & \mathbf{U}_i\Delta && < -\mathbf{U}_i\mathbf{x} - \mathbf{p}_i \\ \implies\ & -\|\mathbf{U}_i\|_q\|\Delta\|_p && < -\mathbf{U}_i\mathbf{x} - \mathbf{p}_i \\ \iff\ & \|\Delta\|_p && > \frac{\mathbf{U}_i\mathbf{x}+\mathbf{p}_i}{\|\mathbf{U}_i\|_q} \end{aligned} \tag{3.13}$$

The $\implies$ comes from Hölder's inequality. (3.13) indicates that a perturbation of l_p norm over $d_{ic} = \max\left\{0, \frac{\mathbf{U}_i\mathbf{x}+\mathbf{p}_i}{\|\mathbf{U}_i\|_q}\right\}$ is needed to break the sufficient condition of $\mathbf{z}_y^{(L)} - \mathbf{z}_i^{(L)} > 0$. Based on the assumption of adversarial budget $\mathbb{S}_\epsilon^{(p)}$ when linearizing the model, the l_p norm of a perturbation to produce an adversarial example is at least $\min\{\epsilon, d_y\}$. □

In many applications, the input is constrained in a hypercube $\left[r^{(min)}, r^{(max)}\right]^M$. For example, for images with normalized pixel intensities, an attacker will not perturb the image outside the hypercube $[0,1]^M$. Such constraint on the attacker allows us to ignore the regions outside the allowable input space, even if they are inside the adversarial budget $\mathbb{S}_\epsilon^{(p)}$. That is to say, when the input instance has additional constraints, we need to adjust the algorithm to verify a larger adversary-free region.

To obtain robustness guarantees in this scenario, we need to recalculate d_{ic}, which is now the distance between the input and the hyperplanes in (3.12) within the hypercube $[r^{(min)}, r^{(max)}]^M$. The value of d_{ic} is then the solution of the following optimization problem:

$$\begin{aligned} & \min_{\Delta} \|\Delta\|_p \\ & s.t.\ \ \mathbf{a}\Delta + b \leq 0,\ \ \Delta^{(min)} \leq \Delta \leq \Delta^{(max)} \end{aligned} \tag{3.14}$$

where, to simplify the notation, we define $\mathbf{a} = \mathbf{U}_i$, $b = \mathbf{U}_i\mathbf{x} + \mathbf{p}_i$, $\Delta^{(min)} = r^{(min)} - \mathbf{x}$ and $\Delta^{(max)} = r^{(max)} - \mathbf{x}$. When $b \leq 0$, the minimum is obviously 0 as the optimal Δ is an all-zero vector. In this case, either we cannot certify the input at all, or even the clean input is misclassified. When $b > 0$, by Hölder's inequality, $\mathbf{a}\Delta + b \geq -\|\Delta\|_p \|\mathbf{a}\|_q + b$, with equality reached when Δ^p and $\mathbf{a}^q$ are collinear. Based on this, the optimal Δ of minimum l_p norm to satisfy $\mathbf{a}\Delta + b \leq 0$ is

$$\widehat{\Delta}^{(i)} = -\frac{b}{\|\mathbf{a}\|_q^q} \text{sign}(\mathbf{a}_i) \, |\mathbf{a}_i|^{\frac{q}{p}} \ , \tag{3.15}$$

where sign$(\cdot)$ is the sign function which returns $+1$ for positive numbers and -1 for negative numbers.

$\widehat{\Delta}^{(i)}$ in Equation 3.15 provides the optimal Δ satisfying $\mathbf{a}\Delta + b \leq 0$. To satisfy the constaint $\Delta^{(min)} \leq \Delta \leq \Delta^{(max)}$, we can use the greedy algorithm that approaches this goal progressively. That is, we first calculate the optimal $\widehat{\Delta}^{(i)}$ based on Equation (3.15) and check if the constraint $\Delta^{(min)} \leq \Delta \leq \Delta^{(max)}$ is satisfied. For the elements in $\widehat{\Delta}^{(i)}$ where it is not, we clip their values within $\left[\Delta^{(min)}, \Delta^{(max)}\right]$ and keep them fixed. We then optimize the remaining elements in the next iteration and repeat this process until the constraint is satisfied for all elements. The pseudo-code is provided as Algorithm 3.4 below.

Algorithm 3.4: Greedy algorithm to solve Problem (3.14).

Input: $\mathbf{x}$, $\mathbf{a}$, b, $\Delta^{(min)}$, $\Delta^{(max)}$ in (3.14) and maximum number of iterations $I^{(max)}$
Set of fixed elements $\mathcal{S}^{(f)} = \varnothing$
Iteration number $k = 0$
Calculate $\widehat{\Delta}^{(i)}$ according to Equation (3.15)
while $\Delta^{(min)} \leq \widehat{\Delta} \leq \Delta^{(max)}$ not satisfied and $k < I^{(max)}$ **do**
  Violated entries $\mathcal{S}^{(v)} = \left\{ j \middle| \widehat{\Delta}_j^{(i)} < \Delta_j^{(min)} \text{ or } \widehat{\Delta}_j^{(i)} > \Delta_j^{(max)} \right\}$
  $\widehat{\Delta}_j^{(i)} = \text{clip}\left(\widehat{\Delta}_j, \text{min} = \Delta_j^{(min)}, \text{max} = \Delta_j^{(max)}\right), j \in \mathcal{S}^{(v)}$
  $\mathcal{S}^{(f)} = \mathcal{S}^{(f)} \cup \mathcal{S}^{(v)}$
  Update $\widehat{\Delta}^{(i)}$ according to (3.15) with elements in $\mathcal{S}^{(f)}$ fixed
  Update $k = k + 1$
end while
Output: $d_{iy} = \|\widehat{\Delta}^{(i)}\|_p$

The following theorem guarantees the optimality of Algorithm 3.4's output given a sufficient large $I^{(max)}$.

Theorem 3.1 *If the maximum number of iterations $I^{(max)}$ in Algorithm 3.4 is large enough to satisfy $\Delta^{(min)} \leq \widehat{\Delta}^{(i)} \leq \Delta^{(max)}$ in Problem equation 3.14, then the output $\|\widehat{\Delta}^{(i)}\|_p$ is the optimum of Problem (3.14), i.e., d_{ic}.*

Proof: We use the primal-dual method to solve the optimization problem (3.14), which is a convex optimization problem with linear constraints.

It is clear that there exists an image inside the allowable pixel space for which the model predicts the wrong label. That is, the constrained problem (3.14) is strictly feasible:

$$\exists \Delta \; s.t. \; \mathbf{a}\Delta + b < 0, \Delta^{(min)} < \Delta < \Delta^{(max)} \, . \tag{3.16}$$

Thus, this convex optimization problem satisfies *Slater's Condition*, i.e., strong duality holds. We then rewrite the primal problem as

$$\begin{aligned} &\min_{\Delta^{(min)} \le \Delta \le \Delta^{(max)}} \|\Delta\|_p^p \\ &s.t. \;\; \mathbf{a}\Delta + b \le 0 \end{aligned} \tag{3.17}$$

We minimize $\|\Delta\|_p^p$ instead of directly $\|\Delta\|_p$ in order to decouple all elements in vector Δ. In addition, we consider $\Delta^{(min)} \le \Delta \le \Delta^{(max)}$ as the domain of Δ instead of constraints for simplicity. We write the dual problem of (3.17) by introducing a coefficient of relaxation $\lambda \in \mathbb{R}_+$:

$$\max_{\lambda \ge 0} \min_{\Delta^{(min)} \le \Delta \le \Delta^{(max)}} g(\Delta, \lambda) := \|\Delta\|_p^p + \lambda(\mathbf{a}\Delta + b) \tag{3.18}$$

To solve the inner minimization problem, we set the gradient $\frac{\partial g(\Delta,\lambda)}{\partial \Delta_j} = \text{sign}(\Delta_j) p |\Delta_j|^{p-1} + \lambda \mathbf{a}_j$ to be zero and obtain $\Delta_j = -\text{sign}(\mathbf{a}_j) \left| \frac{\lambda \mathbf{a}_j}{p} \right|^{\frac{1}{p-1}}$. Based on the convexity of function $g(\Delta, \lambda)$ w.r.t. Δ, we can obtain the optimal $\widetilde{\Delta}_j^{(i)}$ in the domain:

$$\widetilde{\Delta}_j^{(i)} = \text{clip}\left(-\text{sign}(\mathbf{a}_j) \left| \frac{\lambda \mathbf{a}_j}{p} \right|^{\frac{1}{p-1}}, \text{min} = \Delta_j^{(min)}, \text{max} = \Delta_j^{(max)} \right) . \tag{3.19}$$

Based on strong duality, we can say that the optimal $\widetilde{\Delta}^{(i)}$ is chosen by setting a proper value of λ. Fortunately, $\|\widetilde{\Delta}^{(i)}\|_p$ increases monotonically with λ, so the smallest λ corresponds to the optimum.

As we can see, the expression of $\widehat{\Delta}^{(i)}$ in (3.15) is consistent with $\widetilde{\Delta}^{(i)}$ in (3.19) if λ is set properly.[3] The greedy algorithm in Algorithm 3.4 describes the process of gradually increasing λ to find the smallest value satisfying the constraint $\mathbf{a}\Delta + b \le 0$. With the increase of λ, the elements in vector Δ remain unchanged when they reach either $\Delta^{(min)}$ or $\Delta^{(max)}$, so we keep such elements fixed and optimize the others.

□

In Algorithm 3.4, if $I^{(max)}$ is set so small that the while-loop breaks with $\Delta^{(min)} \le \widehat{\Delta} \le \Delta^{(max)}$

[3] The power term $\frac{q}{p} = \frac{1}{p-1}$ when $\frac{1}{p} + \frac{1}{q} = 1$

unsatisfied, then the output of Algorithm 3.4 is the upper bound of Problem (3.14), and thus we eventually get a suboptimal but still valid robustness guarantee. [32] solves the same problem when designing an attack and points out Algorithm 3.4 will converge in $\mathcal{O}(M \log M)$ time. We observed $I^{(max)} = 20$ to be sufficient to satisfy the condition in Theorem 3.1. In practice, the while-loop breaks within 5 iterations in most cases, which means Algorithm 3.4 introduces very little overhead.

3.3.2 Finer-grained and Faster Verification

Based on Lemma 3.1, when $\epsilon < d_y$, our proposed method has the same robustness guarantees as KW [157], Fast-Lin [154] and CROWN [180] if the underlying linear bounds are calculated in the same way. When $0 < d_y < \epsilon$, KW / Fast-Lin / CROWN cannot certify the data point at all, while our method still gives non-trivial robustness guarantees thanks to the geometric interpretability of the polyhedral envelope. Figure 3.6 compares the certified bounds of KW / Fast-Lin[4] and our method on a randomly picked input for different values of ϵ in the predefined adversarial budget. We name our method, demonstrated as Algorithm 3.4, *Polyhedral Envelope Certification* (PEC). We can clearly see the two-phase behavior of both methods in Figure 3.6. In the second phase, unlike KW / Fast-Lin, PEC still provides a non-trivial certification bound.

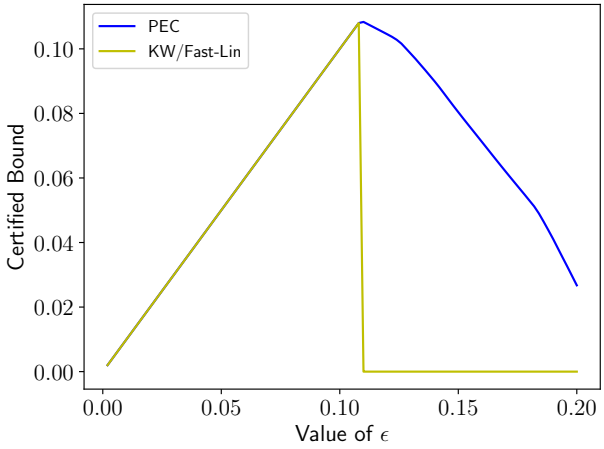


Figure 3.6 – Certified l_∞-based bound of a randomly picked input instance by our method (PEC) and KW / Fast-Lin for different values of ϵ. The model is the 'FC1' model on MNIST trained by 'MMR+at' in [29].

Figure 3.7 shows a 2D sketch of the two phases mentioned above. When ϵ is smaller than a threshold, as in the left half of the figure, the linear bounds in (3.12) are tight but only valid in a small region $\mathcal{S}_\epsilon^{(p)}$. Therefore, the certified robustness is ϵ at most. When ϵ is bigger than this threshold, the linear bounds are valid in a larger region but becomes inevitably loose. This is

[4] In the case of ReLU networks, Fast-Lin and KW are algorithmically the same and yield the same robustness certification.

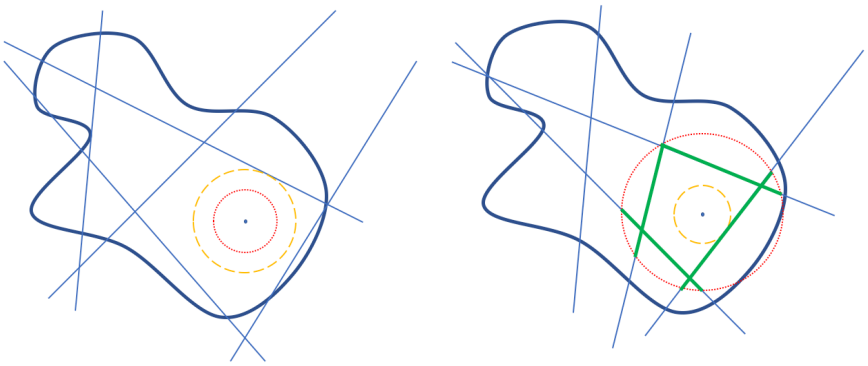


Figure 3.7 – 2D sketch of decision boundary (dark blue bold lines), hyperplane defined by (3.12) (light blue lines), adversarial budget (red dotted circle), polyhedral envelope (green bold lines) inside the adversarial budget. The distance between the input data and the hyperplanes is depicted by a yellow dashed circle. The left and right half correspond to the cases when d_y is bigger and smaller than ϵ, respectively.

because the value of d_{iy} monotonically decreases with the increase of ϵ for all linear bounds of the model's output. This is depicted in the right half of the figure, where the distances between the input and the hyperplanes are smaller. The certified robustness is then d_y. The hyperplane segments inside the adversarial budget (green bold lines) never exceed the decision boundary (dark blue bold lines), by definition of the polyhedral envelope. As a result, to obtain the largest verified adversary-free regions, we need to find the optimal value of ϵ given the input. In Figure 3.6, this is the x-axis value of the 'peak' in the curves. In the sketches of Figure 3.7, this is when the polyhedral envelope is tangent to the adversarial budget. Finding the optimal value of ϵ given to the input is tricky, but we show that our method (PEC) will accelerate the search for it.

Algorithm 3.5: Search for optimal value of ϵ

Input: $\mathbf{x}$, $\underline{\epsilon}$, $\overline{\epsilon}$, ϵ_Δ
Set the bounds of ϵ: $\epsilon_{up} = \overline{\epsilon}$, $\epsilon_{low} = \underline{\epsilon}$
while $\epsilon_{up} - \epsilon_{low} > \epsilon_\Delta$ **do**
  $\epsilon_{try} = \frac{1}{2}(\epsilon_{low} + \epsilon_{up})$
  Calculate the radius ϵ_{cert} of the verified adversary-free region by Algorithm 3.4.
  Update lower bound: $\epsilon_{low} = \max\{\epsilon_{low}, \epsilon_{cert}\}$
  if $\epsilon_{try} > \epsilon_{cert}$ **then**
    Update upper bound: $\epsilon_{up} = \epsilon_{try}$
  end if
end while
Output: the optimal value of ϵ: $\frac{1}{2}(\epsilon_{low} + \epsilon_{up})$

To search for the optimal value of ϵ, [157] uses Newton's method, which is an expensive second-order method. [154, 180] use binary search to improve efficiency. Thanks to the non-trivial certified bounds in the second phase, our proposed PEC can further accelerate their strategy.

During the search, when the algorithm fails to fully verify the value guess $\widetilde{\epsilon}$, the binary search strategy used in Fast-Lin / CROWN can only conclude that the optimal value is smaller than $\widetilde{\epsilon}$. By contrast, in addition to this upper bound of the optimal value, PEC may output a non-trivial verified bound d_y, in which case we can also conclude that the optimal value is larger than d_y. The tighter lower bound on the optimal value makes PEC need fewer steps to reach the required optimal value precision and thus accelerates the search.

The pseudo code for finding the optimal ϵ is provided as Algorithm 3.5. ϵ_Δ, $\underline{\epsilon}$, $\overline{\epsilon}$ represent the precision requirement, the original estimate of the lower bound and of the upper bound, respectively. Typically, $\underline{\epsilon}$ is set to 0 and $\overline{\epsilon}$ is set to a large value corresponding to a perceptible the image perturbation.

3.3.3 Polyhedral Envelope Regularization

Now, we have calculated the verified bounds $d := \min\{\epsilon, d_y\}$ of adversary-free region based on Algorithm 3.4. Since d is differentiable w.r.t. the model parameters θ, we can incorporate our verification method during training so as to obtain provably robust models as in [29]. To this end, we design a regularization term that encourages larger values of d.

We first introduce the *signed distance* $\widehat{d}_{iy}$: when $d_{iy} > 0$, $\widehat{d}_{iy} = d_{iy}$; otherwise, $\widehat{d}_{iy}$ is a negative number whose absolute value is the distance between the input instance and the polyhedral envelope. In the latter case, there is no verified adversary-free region or even the clean input instance itself is misclassified. $\widehat{d}_{iy}$ can be calculated by a greedy algorithm similar to the one in Algorithm 3.4.

Now, we sort $\{\widehat{d}_{iy}\}_{i=1,i\neq y}^{K}$ as $\widehat{d}_{j_1 y} \leq \widehat{d}_{j_2 y} \leq ... \leq \widehat{d}_{j_{K-2} y} \leq \widehat{d}_{j_{K-1} y}$ and then define the *Polyhedral Envelope Regularization* (PER) term, based on the smallest C distances, as:

$$\text{PER}(\mathbf{x}, \alpha, \beta, C) = \beta \sum_{i=1}^{C} \max\left(0, 1 - \frac{\widehat{d}_{j_i y}}{\alpha}\right). \tag{3.20}$$

Note that, following [29], to accelerate training, we take into account the smallest $C > 1$ distances, and in practice, we set $C = 4$ as in [29]. When $\widehat{d}_{j_i y} \geq \alpha$, the distance is considered large enough, so the corresponding term is zero and will not contribute to the gradient of the model parameters. This avoids over-regularization and allows us to maintain accuracy on clean input instances. In practice, we do not activate PER in the early training stages, when the model is not well trained and the corresponding polyhedral envelope is meaningless. Such a 'warm up' trick is commonly used in deep learning practice [54].

We can further incorporate PER with adversarial training in a similar way to [29]. Here, the distance $\widehat{d}_{j_i y}$ in Equation (3.20) is calculated between the polyhedral envelope and the adversarial example generated by PGD [98] instead of the clean input instance. Note that,

the polyhedral envelope is the same in both cases because it only depends on the adversarial budget $\mathbb{S}_\epsilon^{(p)}$. We call this method *PER+at*.

Calculating the polyhedral envelope is expensive in terms of both computation and memory, because of the need to obtain linear bounds of the output logits $\mathbf{z}^{(L)}$. To prevent such a prohibitive computational and memory overhead, we use the *stochastic robust approximation* in [150]. For a mini-batch of size B, we only calculate the PER or PER+at regularization term for $B' < B$ instances randomly sub-sampled from this mini-batch. Each instances in the mini-batch has the same probability to be sampled. [101] empirically observed the geometric correlation of high-dimensional decision boundaries near the data manifold. Although this finding is based on regularly trained models, we find it also holds for models trained by PER / PER+at: in practice, a B' much smaller than B provides a good approximation of the full-batch regularization.

We summarize the full pipeline of PER+at as Algorithm 3.6 as follows:

Algorithm 3.6: Full pipeline of PER+at method

1: **Input:** $\mathcal{D}$, α, β, ϵ, C, B, B'
2: Sample $(\mathbf{X}, \mathbf{y})$ from the dataset $\mathcal{D}$.
3: Subsample $(\mathbf{X}_s, \mathbf{y}_s)$ from the minibatch.
4: Calculate $\mathbf{U}$ and $\mathbf{p}$ in Equation (3.12) for $(\mathbf{X}_s, \mathbf{y}_s)$ given ϵ.
5: Generate adversarial examples $(\mathbf{X}', \mathbf{y}')$ of the whole mini-batch, including the subsamples.
6: Calculate PER regularization term based on Equation (3.20).
7: The final loss is $\frac{1}{2}(\mathcal{L}(\mathbf{X}, \mathbf{y}) + \mathcal{L}(\mathbf{X}', \mathbf{y}')) + \text{PER}(\mathbf{X}'_s, \alpha, \beta, C)$.
8: Back-propagation and update model parameters.

Consider $\mathbf{U}$ and $\mathbf{p}$ in Equation (3.12) can be calculated by either LA-inspired method or IBP-inspired method introduced in Section 3.1, we call the corresponding PER methods *L-PER*, *I-PER*, and PER+at methods *L-PER+at*, *I-PER+at*, respectively.

We now compare the computational complexity of our proposed methods with existing works on an L-layer neural network model with K-dimensional output and M-dimensional input. For simplicity, let each hidden layer have n neurons and usually $n \gg \max\{K, M\}$ is satisfied. In this context, the FLOP complexity of PGD [98] with h iterations is $\mathcal{O}(Ln^2h) \sim \mathcal{O}(Ln^2)$, because typically $h \ll n$. Among the methods that train provably robust networks, the LA-inspired algorithm such as Fast-Lin [154] / CROWN [180] needs $\mathcal{O}(L^2n^3)$ FLOPS to obtain the linear bounds of the output logits. However, the complexity can be reduced to $\mathcal{O}(LMn^2)$ at the cost of bound tightness when we use the IBP-inspired algorithm. Note that the IBP-inspired algorithm also calculates the linear bound of the output logits and is thus different from IBP [55], whose complexity is $\mathcal{O}(Ln^2)$, i.e., the same as a forward propagation. To update the model parameters, KW needs a back-propagation which costs $\mathcal{O}(Ln^2)$ FLOPs. Therefore the complexity of KW [157] is also $\mathcal{O}(L^2n^3)$, with the linear approximation dominating the complexity. In CROWN-IBP [179], the bounds of all intermediate layers are estimated by IBP, which costs $\mathcal{O}(Ln^2)$ FLOPs. The last layer's bound is then estimated in the same way as CROWN, which costs $\mathcal{O}(Ln^3)$ FLOPs, dominating the complexity of CROWN-IBP. For MMR [29],

Methods	Complexity
PGD [98]	$\mathcal{O}(Ln^2)$
Fast-Lin [154] / CROWN [180]	$\mathcal{O}(L^2n^3)$
KW [157]	$\mathcal{O}(L^2n^3)$
MMR / MMR+at [29]	$\mathcal{O}(LMn^2)$
IBP [55]	$\mathcal{O}(Ln^2)$
CROWN-IBP [179]	$\mathcal{O}(Ln^3)$
I-PER / I-PER+at (Ours)	$\mathcal{O}(LMn^2)$
L-PER / L-PER+at (Ours)	$\mathcal{O}(L^2n^3)$

Table 3.4 – Complexity of different methods on an L-layer neural network model with K-dimensional output and M-dimensional input. Each hidden layer has n neurons.

the complexity to calculate the expression of the input's linear region is $\mathcal{O}(LMn^2)$. MMR then calculates the distances between the input and $\mathcal{O}(Ln)$ hyper-planes, costing $\mathcal{O}(LMn)$. Altogether, the complexity of MMR is $\mathcal{O}(LMn^2)$. MMR+at has the same complexity as MMR, because the overhead of adversarial training can be ignored.

Among our methods, the complexity of LA-inspired L-PER is $\mathcal{O}(L^2n^3)$. Like MMR and KW, the overhead of distance calculation and back-propagation can be ignored. Similarly, the complexity of I-PER is dominated by the IBP-inspired bound of the output logits, which is $\mathcal{O}(LMn^2)$. Note that L-PER has the same complexity as Fast-Lin, CROWN and KW, and the complexity of I-PER is smaller than that of CROWN-IBP because $M \ll n$. L-PER+at and I-PER+at have the same complexity as L-PER and I-PER, respectively, since the overhead of adversarial training is negligible. Table 3.4 summarizes the complexity of all methods.

3.3.4 Experiments and Analysis

We validate our proposed methods by experiments in this section, including L-PER, L-PER+at, I-PER and I-PER+at. We focus on popular benchmarks: MNIST [89] and CIFAR10 [83]. All of our experiments can be completed on a single NVDIA TITAN XP GPU machine with 12GB memory within several hours. Codes and checkpoints are publicly available. [5]

Training and Verifying ReLU Networks

We first demonstrate the benefits of our approach over existing training and certification methods under the same computational complexity. To this end, we use the same model architectures as in [29, 157]: **FC1**, which is a fully-connected network with one hidden layer of 1024 neurons; and **CNN**, which has two convolutional layers followed by two fully-connected layers. For this set of experiments, all activation functions are ReLU.

When it comes to training, we consider 7 baselines, including plain training (plain), adversarial

[5] https://github.com/liuchen11/PolyEnvelope

training (AT) [98], KW [157], IBP [55], CROWN-IBP [179], MMR and MMR plus adversarial training (MMR + at) [29]. We compare them with our proposed methods: L-PER, L-PER+at when using linear approximation to derive the output bounds for PER and PER+at, respectively; I-PER, I-PER+at when using IBP-inspired linear bounds of the output. We do not compare randomized smoothing [27, 122] or layerwise training [9]. This is because the verified bounds of randomized smoothing are not exact but probabilistic, and layerwise training has significant computational overhead.[6] For fair comparison, we use the same adversarial budget in both the training and the test phases.

To evaluate the models' performance on the test set, we first report the clean test error (CTE) and the empirical robust error against PGD attack (PGD). Based on the discussions in Section 3.3.2, KW, Fast-Lin and our proposed PEC (Algorithm 3.4) have the same certified robust error, which is the proportion of the input data whose certified regions are smaller than the adversarial budget. Therefore, for these three methods, we report the certified robust error as CRE Lin. We also report the certified robust error by IBP [55] as CRE IBP. For l_∞ robustness, we use a complete certifier called MIPVerify [143] to calculate the exact robust error, denoted by CRE MIP. [7] In addition, we calculate the average certified bound obtained by Fast-Lin / KW (ACB Lin)[8], IBP (ACB IBP) and PEC (ACB PEC). Note that the average certified bound here is from the *one-shot* certifier, i.e., without searching for the optimal adversarial budget. We do not report the certified bound obtained by MMR [29], because, in practice, it only gives trivial results. As a matter of fact, [29] emphasize their training method and report certification results using only KW and MIP.

We use the same adversarial budgets and model architectures as [29] and thus directly download the KW, MMR and MMR+at models from the checkpoints provided online.[9] For IBP and CROWN-IBP, we use the same hyper-parameter settings as [179] except that we align the training duration to other methods and the use stochastic robustness approximation of Section 3.3.3 to reduce the computational and memory consumption. For CNN models, we use the *warm up* trick consisting of performing adversarial training before adding our PER or PER+at regularization term. The running time overhead of pre-training is negligible compared with computing the regularization term.

In all experiments, we use the Adam optimizer [81] with an initial learning rate of 10^{-3} and train all models for 100 epochs with a mini-batch of 100 instances. For CNN models, we decrease the learning rate to 10^{-4} for the last 10 epochs. When we train CNN models on MNIST, we only calculate the polyhedral envelope of 20 instances subsampled from each mini-batch. When we train CNN models on CIFAR10, this subsampling number is 10. These settings make our algorithm possible to be trained on a GPU with 12 GB memory. For PER

[6] For CNN models, [9] trains 200 epochs for each layer and 800 epochs in total, while the other baselines use only 100 epochs. If we reduce the training epochs of each layer to 25 epochs, the model does not converge well. For FC1 models, [9] is the same as KW, because there is only one hidden layer.

[7] MIPVerify is available on https://github.com/vtjeng/MIPVerify.jl

[8] Fast-Lin and KW is algorithmically the same in ReLU networks

[9] https://github.com/max-andr/provable-robustness-max-linear-regions.

and PER+at, the value of C in Equation 3.20 is always 4. We search in the logarithmic scale for the value of β and in the linear scale for the value of α. For ϵ, we ensure that its values in the end of training are close to the ones used in the adversarial budget $\mathcal{S}_\epsilon^{(p)}$. We compare constant values with an exponential growth scheme for ϵ but always use constant values for α and β. The optimal values we found for different settings are provided in Table 3.5.

Task	α	β	ϵ
MNIST FC1, l_∞	0.15	0.1	initial value 0.0064 ×2 every 20 epochs
MNIST CNN, l_∞	0.15	PER: 0.3 PER+at: 0.03	0.1
CIFAR10 CNN, l_∞	0.1	PER: 0.0003 PER+at: 0.001	0.008
MNIST FC1, l_2	0.45	1.0	initial value 0.02 ×2 every 20 epochs
MNIST CNN, l_2	0.45	1.0	0.3
CIFAR10 CNN, l_2	0.15	PER: 0.3 PER+at: 1.0	0.1

Table 3.5 – Values of α, β and ϵ for different experiments.

We constrain the attacker to perturb the images within $[0,1]^M$. The full results for l_∞ attacks and l_2 attacks are summarized in Table 3.6 and Table 3.7, respectively. For l_∞ attacks, our proposed methods achieve the best verified accuracy, calculated by the complete certifier (CRE MIP), in all cases. For l_2 attacks, they also achieve the best estimated verified accuracy, calculated by the Fast-Lin / KW / PEC certifier (CRE Lin), in all cases. In addition, the performance of I-PER and I-PER+at is on par with that of L-PER and L-PER+at, which illustrates that our framework is not sensitive to the tightness of the underlying method for outputs' linear bounds and thus generally applicable.

As observed in previous work [111], different incomplete certifiers are complementary. For example, IBP is only able to certify IBP-trained models and has worse certification results on other models. For the training methods other than IBP and CROWN-IBP, we notice big gaps between the true robustness (CRE MIP) and the IBP certified robustness (CRE IBP). This is because IBP and CROWN-IBP solve a different optimization problem from the other methods. Specifically, IBP and CROWN-IBP do not make any approximation of the activation function, they only utilize the monotonicity of the activation function to propagate the bounds. However, all the other methods use linear approximations to bound the outputs of the activation functions. We also note that the stochastic robustness approximation greatly hurts the performance of IBP and CROWN-IBP on CIFAR10. However, the result reported in [179] without stochastic robustness approximation on the same architecture is still worse

than our method.[10] Consistently with Section 3.3.2, our geometry-inspired PEC has better average certified bounds than Fast-Lin / KW given the same adversarial budget. For example, on the CIFAR10 model against l_∞ attack, 10% – 20% of the test points are not certified by Fast-Lin / KW but have non-trivial bounds with PEC.

Methods	CTE (%)	PGD (%)	CRE Lin (%)	CRE IBP (%)	CRE MIP (%)	ACB Lin	ACB IBP	ACB PEC
MNIST - FC1, ReLU, l_∞, $\epsilon = 0.1$								
plain	1.99	98.37	100.00	100.00	100.00	0.0000	0.0000	0.0000
AT	1.42	9.00	97.94	100.00	100.00	0.0021	0.0000	0.0099
KW	2.26	8.59	12.91	69.20	10.90	0.0871	0.0308	0.0928
IBP	1.65	9.67	87.27	**15.20**	12.36	0.0127	**0.0848**	0.0705
CROWN-IBP	1.98	9.50	67.39	**14.45**	11.39	0.0326	**0.0855**	0.0800
MMR	2.11	17.82	33.75	99.88	24.90	0.0663	0.0001	0.0832
MMR+at	2.04	10.39	17.64	95.09	14.10	0.0824	0.0049	0.0905
L-PER	**1.60**	**7.45**	**11.71**	92.89	**7.69**	**0.0883**	0.0071	**0.0935**
L-PER+at	1.81	7.73	12.90	99.90	8.22	0.0871	0.0001	0.0925
I-PER	**1.60**	**6.28**	**11.96**	93.33	**8.10**	**0.0880**	0.0067	**0.0934**
I-PER+at	**1.54**	7.15	13.96	98.55	8.48	0.0868	0.0014	0.0927
MNIST - CNN, ReLU, l_∞, $\epsilon = 0.1$								
plain	1.28	85.75	100.00	100.00	100.00	0.0000	0.0000	0.0000
AT	1.02	4.75	91.91	100.00	100.00	0.0081	0.0000	0.0189
KW	1.21	3.03	**4.44**	100.00	4.40	**0.0956**	0.0000	**0.0971**
IBP	1.51	4.43	23.89	**8.13**	5.23	0.0761	**0.0919**	0.0872
CROWN-IBP	1.85	4.28	10.72	**6.91**	4.83	0.0893	**0.0931**	0.0928
MMR	1.65	6.07	11.56	100.00	6.10	0.0884	0.0000	0.0928
MMR+at	1.19	3.35	9.49	100.00	3.60	0.0905	0.0000	0.0939
L-PER	1.44	3.44	5.13	100.00	3.62	0.0949	0.0000	0.0965
L-PER+at	**0.50**	**2.02**	4.85	100.00	**2.21**	0.0952	0.0000	0.0969
I-PER	1.03	2.40	4.64	99.55	2.52	**0.0954**	0.0004	0.0967
I-PER+at	**0.48**	**1.29**	**4.61**	99.94	**1.47**	**0.0954**	0.0001	**0.0971**
CIFAR10 - CNN, ReLU, l_∞, $\epsilon = 2/255$								
plain	24.62	86.29	100.00	100.00	100.00	0.0000	0.0000	0.0000
AT	27.04	48.53	85.36	100.00	88.50	0.0011	0.0000	0.0015
KW	39.27	46.60	**53.81**	99.98	**48.00**	**0.0036**	0.0000	**0.0040**
IBP	46.74	56.38	61.81	**67.58**	58.80	0.0030	**0.0025**	0.0034
CROWN-IBP	58.32	63.56	66.28	**69.10**	65.44	0.0026	**0.0024**	0.0029
MMR	34.59	57.17	69.28	100.00	61.00	0.0024	0.0000	0.0032
MMR+at	35.36	49.27	59.91	100.00	54.20	0.0031	0.0000	0.0037
L-PER	39.21	50.98	57.45	99.98	52.70	0.0033	0.0000	0.0038
L-PER+at	**28.87**	**43.55**	**56.59**	100.00	48.43	**0.0034**	0.0000	**0.0040**
I-PER	29.34	51.54	64.34	99.98	54.87	0.0028	0.0000	0.0036
I-PER+at	**26.66**	**43.35**	57.72	100.00	**47.87**	0.0033	0.0000	**0.0040**

Table 3.6 – Full results of 11 training schemes and 8 evaluation schemes for ReLU networks under l_∞ attacks. The best and the second best results among provably robust training methods (plain and AT excluded) are bold. In addition, the best results are underlined.

[10] The DM-small model in [179] yields a certified robust error of 52.57% on CIFAR10 when $\epsilon = 2/255$.

Method	CTE (%)	PGD (%)	CRE Lin (%)	CRE IBP (%)	ACB Lin	ACB IBP	ACB PEC
MNIST - FC1, ReLU, l_2, $\epsilon = 0.3$							
plain	1.99	9.81	40.97	99.30	0.1771	0.0021	0.2300
AT	1.35	2.99	14.85	99.23	0.2555	0.0023	0.2684
KW	1.23	2.70	**4.91**	41.55	**0.2853**	0.1754	**0.2892**
IBP	1.36	2.90	6.87	**9.01**	0.2794	**0.2730**	0.2876
CROWN-IBP	1.26	2.80	6.36	**8.73**	0.2809	**0.2738**	0.2884
MMR	2.40	5.88	7.76	99.55	0.2767	0.0013	0.2845
MMR+at	1.77	3.76	5.68	99.86	0.2830	0.0004	0.2880
L-PER	1.26	2.44	5.35	59.17	0.2840	0.1225	0.2888
L-PER+at	**0.67**	**1.40**	**4.84**	64.79	**0.2855**	0.1056	**0.2910**
I-PER	1.21	2.59	5.34	54.13	0.2840	0.1376	0.2888
I-PER+at	**0.74**	**1.46**	7.81	72.85	0.2766	0.0814	0.2860
MNIST - CNN, ReLU, l_2, $\epsilon = 0.3$							
plain	1.28	4.93	100.00	100.00	0.0000	0.0000	0.0000
AT	1.12	2.50	100.00	100.00	0.0000	0.0000	0.0000
KW	1.11	2.05	5.84	100.00	0.2825	0.0000	0.2861
IBP	2.37	3.85	51.12	**11.73**	0.1534	**0.2648**	0.1669
CROWN-IBP	2.89	4.44	31.62	**12.29**	0.2051	**0.2631**	0.2178
MMR	2.57	5.49	10.03	100.00	0.2699	0.0000	0.2788
MMR+at	1.73	3.22	9.46	100.00	0.2716	0.0000	0.2780
L-PER	1.02	1.87	**5.04**	100.00	**0.2849**	0.0000	**0.2882**
L-PER+at	**0.43**	**0.91**	**5.43**	100.00	**0.2837**	0.0000	**0.2878**
I-PER	1.11	2.16	6.37	100.00	0.2809	0.0000	0.2851
I-PER+at	**0.52**	**1.12**	7.89	100.00	0.2763	0.0000	0.2812
CIFAR10 - CNN, ReLU, l_2, $\epsilon = 0.1$							
plain	23.29	47.39	100.00	100.00	0.0000	0.0000	0.0000
AT	25.84	35.81	99.96	100.00	0.0000	0.0000	0.0000
KW	40.24	43.87	48.98	100.00	0.0510	0.0000	0.0533
IBP	57.90	60.03	64.78	**78.13**	0.0352	**0.0219**	0.0366
CROWN-IBP	71.21	72.51	76.23	**80.97**	0.0238	**0.0190**	0.0256
MMR	40.93	50.57	57.07	100.00	0.0429	0.0000	0.0480
MMR+at	37.78	43.98	53.33	100.00	0.0467	0.0000	0.0502
L-PER	34.10	52.54	63.42	100.00	0.0369	0.0000	0.0465
L-PER+at	**25.76**	**33.47**	**46.74**	100.00	**0.0533**	0.0000	**0.0580**
I-PER	33.94	43.06	56.80	100.00	0.0432	0.0000	0.0484
I-PER+at	**24.85**	**31.32**	**47.28**	100.00	**0.0528**	0.0000	**0.0572**

Table 3.7 – Full results of 11 training schemes and 7 evaluation schemes for ReLU networks under l_2 attacks. The best and the second best results among provably robust training methods (plain and at excluded) are bold. In addition, the best results are underlined.

When compared with KW, our methods, especially PER+at, have much better clean test accuracy. In other words, a model trained by L-PER+at or I-PER+at is not as over-regularized as other training methods for provable robustness. Figure 3.8 shows the distribution of parameter values of KW, MMR+at, L-PER+at models on CIFAR10 against l_∞ attacks and l_2 attacks. As we can see, the parameters of L-PER+at models have much larger norms than KW and MMR+at,

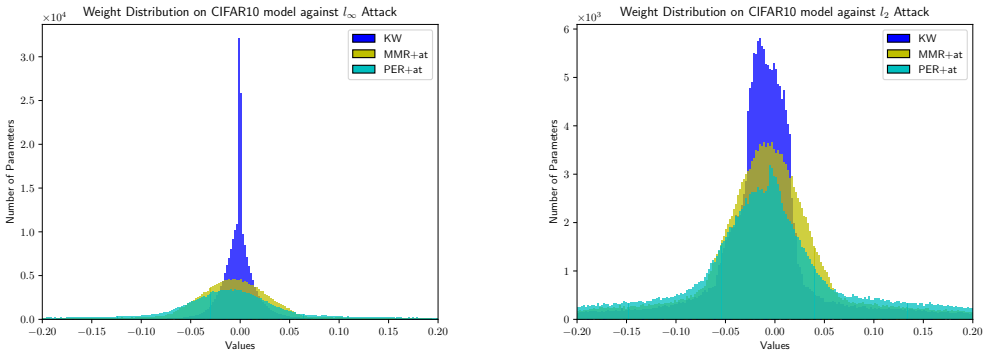


Figure 3.8 – Parameter value distributions of CIFAR10 models trained against l_∞ attacks (left) and l_2 attacks. The parameters of models trained by PER+at have larger magnitude.

whose parameters are more sparse. The norms of the model parameters indicate the model capacity [106, 105], so L-PER+at models better preserve the model capacity.

The better performance of (L/I)-PER+at over (L/I)-PER, and of MMR+at over MMR, evidences the benefits of augmenting the training data with the adversarial examples. However, this strategy is only compatible with methods that rely on estimating the distance between the input instance and the decision boundary, and thus cannot be combined with methods such as KW. Adding a loss term on the adversarial examples to the loss objective of KW yields a performance between adversarial training and KW. For example, if we optimize the sum of loss objectives of KW and PGD for a CNN model on MNIST against l_∞ adversarial attacks, the robust error against PGD of the resulting model is 3.64%, the provably robust error by Fast-Lin (CRE Lin) is 8.12%. In other words, such combinations only lead to mixed performance and are weaker than KW in terms of provable robustness.

Training and Verifying Non-ReLU Networks

Some of previous methods such as MMR and MMR+at [29] utilize the piece-wise linear property of ReLU function and thus can only be applied on ReLU networks. By contrast, our methods are generally applicable to any activation functions. To validate our method's applicability to non-ReLU networks, we replace the ReLU function in FC1 models with either sigmoid or tanh functions.

In this context, MMR and MMR+at are no longer applicable. While KW [160] claims that their methods apply to non-ReLU networks, their main contribution is rather the extension of KW to a broader set of network architectures, and their public code[11] does not support non-ReLU activations. For evaluation, MIPVerify does not support sigmoid or tanh functions neither, since it works only on ReLU networks. In addition, we replace Fast-Lin and KW with CROWN [180], their extension to general activation functions, and thus report its certified

[11] Repository: https://github.com/locuslab/convex_adversarial

Methods	CTE (%)	PGD (%)	CRE CRO (%)	CRE IBP (%)	ACB CRO	ACB IBP	ACB PEC
			MNIST - FC1, Sigmoid, $l_\infty, \epsilon = 0.1$				
plain	2.04	97.80	100.00	100.00	0.0000	0.0000	0.0000
at	1.78	10.05	98.52	100.00	0.0015	0.0000	0.0055
IBP	2.06	10.58	44.14	13.65	0.0559	0.0863	0.0846
CROWN-IBP	2.88	9.83	26.04	**12.51**	0.0740	**0.0875**	0.0886
L-PER	**1.97**	7.55	12.15	84.76	0.0879	0.0152	**0.0930**
L-PER+at	2.16	**7.12**	**11.87**	88.06	**0.0881**	0.0119	0.0927
I-PER	2.15	8.35	12.79	86.99	0.0872	0.0130	0.0926
I-PER+at	2.45	8.05	12.36	88.94	0.0876	0.0111	0.0923
			MNIST - FC1, Tanh, $l_\infty, \epsilon = 0.1$				
plain	2.00	97.80	100.00	100.00	0.0000	0.0000	0.0000
at	1.28	8.89	99.98	100.00	0.0000	0.0000	0.0001
IBP	**2.04**	9.84	31.81	13.02	0.0682	0.0870	0.0864
CROWN-IBP	2.75	9.57	20.10	**11.80**	0.0799	**0.0882**	0.0894
L-PER	2.19	7.71	11.55	57.81	0.0885	0.0422	**0.0934**
L-PER+at	2.30	**7.45**	**11.39**	56.74	**0.0886**	0.0433	0.0930
I-PER	2.21	8.51	12.23	55.53	0.0878	0.0445	0.0929
I-PER+at	2.46	7.87	12.04	66.04	0.0880	0.0340	0.0929
			MNIST - FC1, Sigmoid, $l_2, \epsilon = 0.3$				
plain	2.01	10.25	30.78	94.82	0.2077	0.0155	0.2539
at	1.65	3.48	7.50	85.84	0.2775	0.0422	0.2839
IBP	1.40	3.07	6.43	9.13	0.2807	0.2726	0.2873
C-IBP	1.51	3.24	6.36	**8.73**	0.2709	**0.2738**	0.2872
L-PER	1.36	2.58	6.12	73.71	0.2816	0.0789	0.2867
L-PER+at	**0.46**	**1.03**	5.26	68.94	0.2842	0.0932	0.2905
I-PER	1.19	2.59	6.05	70.18	0.2818	0.0895	0.2871
I-PER+at	0.49	1.16	**5.03**	65.79	**0.2849**	0.1026	**0.2907**
			MNIST - FC1, Tanh, $l_2, \epsilon = 0.3$				
plain	1.94	16.46	61.66	99.64	0.1150	0.0011	0.1789
at	1.36	3.02	12.35	97.66	0.2630	0.0070	0.2735
IBP	1.57	3.17	7.21	10.44	0.2784	0.2688	0.2851
C-IBP	1.50	3.14	6.64	**9.53**	0.2801	**0.2714**	0.2861
L-PER	1.31	2.47	**5.53**	55.17	**0.2834**	0.1345	0.2880
L-PER+at	0.58	1.30	5.89	54.88	0.2823	0.1354	0.2885
I-PER	1.38	2.85	5.90	45.31	0.2823	0.1641	0.2874
I-PER+at	**0.55**	**1.17**	5.57	53.73	0.2833	0.1388	**0.2890**

Table 3.8 – Full results of 8 training schemes and 7 evaluation schemes for sigmoid and tanh networks under l_∞ attacks and l_2 attacks. The best results among provably robust training methods (plain and at excluded) are bold and underlined.

robust error (CRE CRO) and average certified bound (ACB CRO). During training, we utilize the linear bounds derived in Section 3.1, which is slightly different from the ones in CROWN. This is becausewe need an analytical form of the linear bounds in order to calculate their gradients w.r.t. the model parameters. When we verify models using CROWN, we use exactly the same

bounds as in CROWN, because it is tighter numerically.

The results on both l_∞ cases and l_2 cases are shown in Table 3.8. Similar to the ReLU networks in the previous section, our proposed methods have the best performance in all cases, in terms of both certified robust error and average certified bound. IBP can only certify IBP-trained models well and has significantly worse results on other models.

Search for Optimal Adversarial Budget

To obtain the biggest verified bound, we need to search for the optimal value of ϵ, i.e., the peak in Figure 3.6. KW [157] uses Newton's method to solve a constrained optimization problem, which is an expensive second-order method. Fast-Lin and CROWN [154, 180] apply a binary search strategy to find the optimal ϵ. Based on Figure 3.6, the optimal adversarial budget for a data point is also its optimal verified bound.

To validate the claim in Section 3.3.2 that PEC can find the optimal adversarial budget faster than Fast-Lin / CROWN by Algorithm 3.5, we compare the average number of iterations needed to find the optimal value given a predefined precision requirement ϵ_Δ. Using $\underline{\epsilon}$ and $\overline{\epsilon}$ to define the initial lower and upper estimates of the optimal value, we then need $\lceil \log_2 \frac{\overline{\epsilon}-\underline{\epsilon}}{\epsilon_\Delta} \rceil$ steps of bound calculations to obtain the optimal value by binary search in Fast-Lin / CROWN. By contrast, the number of bound calculations needed by PEC is smaller and depends on the model to certify, because the partial certified bounds obtained by PEC indicate tighter lower bounds of the optimal adversarial budget.

We show the results on both l_∞ and l_2 cases in Table 3.9. For l_∞ cases, the original interval $[\underline{\epsilon}, \overline{\epsilon}]$ is $[0, 0.4]$ for MNIST and $[0, 0.1]$ for CIFAR10. For l_2 cases, the original interval $[\underline{\epsilon}, \overline{\epsilon}]$ is $[0, 1.2]$ for MNIST and $[0, 0.4]$ for CIFAR10. In Fast-Lin / CROWN, the number of bound calculation do not depend on the model and is a constant. Note that, because PEC has almost no computational overhead compared with Fast-Lin and CROWN,[12] the number of iterations reflects the running time to obtain the optimal certified bounds. Altogether, our results in Table 3.9 show that PEC can save approximately 25% of the running time for FC1 models and 10% of the running time for CNN models.

Figure 3.9 shows the distribution of the optimal certified bounds for CIFAR10 models obtained by KW, MMR+at and C-PER+at against l_∞ attacks and l_2 attacks on the test set. We use vertical red lines to represent the target bounds ($2/255$ in the l_∞ case and 0.1 in the l_2 case), so the area on the right of this line represents the certified robust accuracy. Compared with KW, the mass of C-PER+at is more concentrated on a narrower range on the right of the red line. This evidences that there are significantly fewer points that have unnecessarily large certified bounds for the L-PER+at model than for the KW one. This is because PER+at encourages

[12] We run one-iteration PEC and Fast-Lin to certify CIFAR10-CNN models by L-PER+at for 10 times. To process the entire test set on a single GPU machine, in l_∞ cases, the mean and standard deviation of run time is 217.51 ± 1.95 seconds for Fast-Lin and 219.16 ± 3.23 seconds for PEC; in l_2 cases, it is 236.95 ± 1.64 for Fast-Lin and 239.41 ± 1.92 for PEC. Therefore the difference can be ignored.

Methods	**MNIST-FC1, l_∞**			**MNIST-CNN, l_∞**			**CIFAR10-CNN, l_∞**		
	T_{Lin}	T_{PEC}	$\frac{T_{PEC}}{T_{Lin}}$	T_{Lin}	T_{PEC}	$\frac{T_{PEC}}{T_{Lin}}$	T_{Lin}	T_{PEC}	$\frac{T_{PEC}}{T_{Lin}}$
plain		9.85	0.8207		10.56	0.8804		9.33	0.9331
at		10.77	0.8972		11.39	0.9489		9.12	0.9128
KW		8.48	0.7066		11.61	0.9674		8.43	0.8432
MMR	12	8.04	0.6703	12	10.68	0.8897	10	8.05	0.8053
MMR+at		7.68	0.6402		11.22	0.9351		8.45	0.8450
L-PER		9.34	0.7780		11.17	0.9305		8.61	0.8606
L-PER+at		9.38	0.7816		11.74	0.9784		8.68	0.8681
Methods	**MNIST-FC1, l_2**			**MNIST-CNN, l_2**			**CIFAR10-CNN, l_2**		
	T_{Lin}	T_{PEC}	$\frac{T_{PEC}}{T_{Lin}}$	T_{Lin}	T_{PEC}	$\frac{T_{PEC}}{T_{Lin}}$	T_{Lin}	T_{PEC}	$\frac{T_{PEC}}{T_{Lin}}$
plain		9.68	0.6914		13.64	0.9742		11.73	0.9775
at		10.44	0.7457		13.76	0.9829		11.67	0.9725
KW		7.72	0.5514		12.63	0.9021		10.23	0.8525
MMR	14	5.86	0.4186	14	8.52	0.6086	12	9.05	0.7542
MMR+at		5.91	0.4221		12.13	0.8664		10.33	0.8608
L-PER		11.47	0.8194		13.75	0.9819		9.13	0.7609
L-PER+at		11.34	0.8100		13.72	0.9796		10.71	0.8926

Table 3.9 – Number of bound calculation needed for the optimal ϵ in Fast-Lin (T_{Lin}) and PEC (T_{PEC}) for ReLU networks under l_∞ attacks and l_2 attacks. Note that T_{Lin} is a constant for different models given the original interval $[\underline{\epsilon}, \overline{\epsilon}]$.

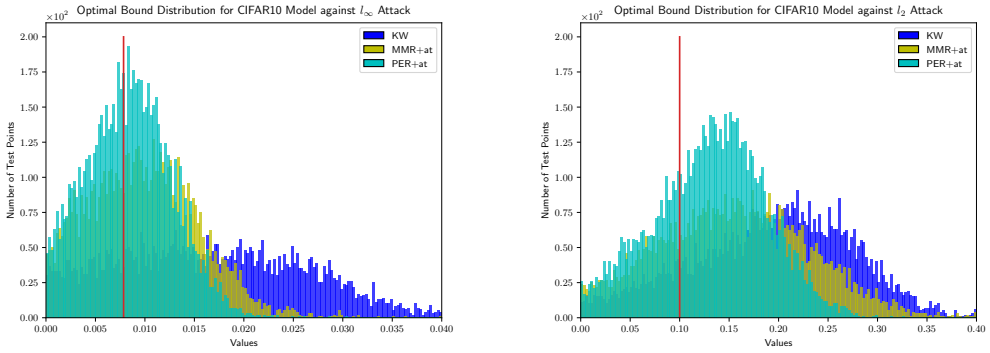


Figure 3.9 – Distribution of optimal certified bounds of CIFAR10 models trained against l_∞ attacks (left) and l_2 attacks (right). The target bounds (2/255 on the left, 0.1 on the right) are marked as red vertical lines.

robustness via a hinge-loss term. When $\widehat{d}_{ic} \geq \alpha$, the regularizer in Equation (3.20) is a constant zero and does not contribute to the parameter gradient. However, KW first estimates the bound of the worst case output logits and calculates the softmax cross-entropy loss on that. Under this training objective function, each data point is encouraged to make the lower bound of the true label's output logit bigger and the upper bound of false ones smaller, even if the current model is sufficiently robust at this point. This phenomenon also helps to explain why

KW tends to over-regulate the model while our methods do not, indicated in Figure 3.8.

We have also tried to replace the cross-entropy loss with the hinge loss in the objective of KW, but do not observe any improvement over the original KW. KW directly minimizes the gap between the logits of the true and false label, but the logits' magnitude for different instances differs, which makes it difficult or even impossible to set a unified threshold in the hinge loss. By contrast, in PER, we apply the hinge loss to the certified bound directly, which is normalized, easier to interpret and thus makes it much easier to set the threshold in the hinge loss. In practice, the value of α in Equation 3.20 is set 1.5 times the target adversarial budget.

3.4 Summary and Broader Impact

In this chapter, we focus on the robust verification problem. We study, in particular, the problem from the geometric perspective. In Section 3.2, we extend the existing verification method for verifying a non-uniform adversary-free region with a much larger volume than the uniform counterpart. In Section 3.3, we estimate the distance between the decision boundary and the input instance, and we propose a regularization framework for training provably robust neural networks.

Both of our works deepen our understandings about the geometry of the neural networks' decision boundaries. For verifying non-uniform bounds, it helps us to distinguish non-robust features from robust ones, and helps us further analyze the model's behavior and interpretability. Non-uniform bounds also enrich the definition of the adversarial budget, because features with high verified bounds are robust features, and because perturbing such features of a higher magnitude does not change the semantic meanings of the input instances. There are already some preliminary works [43] that study robustness against such non-uniform adversarial budgets. For polyhedral envelope regularization, it is the first geometry-inspired method for training provably robust neural networks with general activation functions. Our regularization framework can train models with robustness guarantees without sacrificing the performance on the clean input instances too much. The hinge-loss form of regularization, the key to avoid over-regularization, is different from most existing methods for provable robustness, and it is certainly worth further exploration.

Although verified robustness comes with a theoretical guarantee, the scalability is the key barrier to verifying industry-sized models. There are two major issues: (1) the complexity of some methods is significantly higher than the standard forward or backward passes, such as complete verifier and the methods based on linear approximation; (2) the bounds of some methods becomes inevitably looser with the increase of network's depth, including most incomplete verifiers. Consequently, for large-scale models, researchers are more interested in empirical robustness, which will be discussed in the next chapter.

4 Empirical Robustness

Following the discussion in Section 1.1, empirical robustness, which is the proportion of input instances robust against the state-of-the-art attacks, indicates the upper bound of the model's true robustness. Among the methods for achieving empirical robustness, adversarial training [98] is the most popular one in practice. However, compared with empirical risk minimization, adversarial training is much more challenging. In this chapter, we study the two challenges in adversarial training: slow convergence and large generalization gap. Before we discuss these phenomena in depth, we first briefly review what adversarial training is and how it is different from traditional empirical risk minimization.

The contents of this chapter are mainly from the following two papers. I am the primary contributor of both papers.

- Chen Liu, Mathieu Salzmann, Tao Lin, Ryota Tomioka, Sabine Süsstrunk. "On the Loss Landscape of Adversarial Training: Identifying Challenges and How to Overcome Them." Neural Information Processing Systems 2020.
- Chen Liu, Zhichao Huang, Mathieu Salzmann, Tong Zhang, Sabine Süsstrunk. "On the Impact of Hard Adversarial Instances on Overfitting in Adversarial Training." Preprint.

4.1 Adversarial Training

Adversarial training in [98] alternatively solves the inner maximization and outer minimization problem in 1.2. It first generates adversarial examples $\mathbf{x}'$, usually by PGD, and then optimize model parameters θ based on these adversarial examples. [7] studies 9 existing methods that claim to achieve robustness, adversarial training is the only one that withstand the strong adaptive attacks and does not suffer from gradient masking. After that, adversarial training becomes the de facto method in practice to achieve empirical robustness.

There are many variants that improve the original adversarial training. TRADES [178] generates adversarial perturbations by solving $\max_{\Delta \in \mathcal{S}_\epsilon^{(p)}} D(f(\theta, \mathbf{x}) || f(\theta, \mathbf{x} + \Delta))$ where D is the

metric measuring the distributional distance, such as Kullback–Leibler (KL) divergence. The optimization objective of TRADES is $\mathcal{L}(f(\theta,\mathbf{x}),y)+\lambda\max_{\Delta\in\mathcal{S}_\epsilon^{(p)}} D(f(\theta,\mathbf{x})||f(\theta,\mathbf{x}+\Delta))$ where the coefficient λ balances the trade-off between the clean accuracy and the robust accuracy. MART [153] improves TRADES by regularizing input instances that are misclassified. Adversarial weight perturbation [161] improves the performance by adversarially perturbing the model parameters θ to encourage convergence to a flat minima in the model parameter space. [114] achieves the state-of-the-art performance by model weight averaging and heuristics-driven data augmentation such as Cutout, CutMix [175].

Figure 4.1 demonstrates the learning curves of vanilla training (empirical risk minimization) and adversarial training on a 18-layer Residual Network (RN18) in [98] on CIFAR10. For vanilla training, we report the error and loss value on the clean input instances; for adversarial training, we report them on the adversarially perturbed input instances under l_∞ adversarial budget with $\epsilon = 8/255$. We can clearly see that adversarial training converges much slower on the training set than vanilla training. In addition, adversarial training has a much larger generalization gap, and its performance on the test set sees significant decay in the late phase of training.

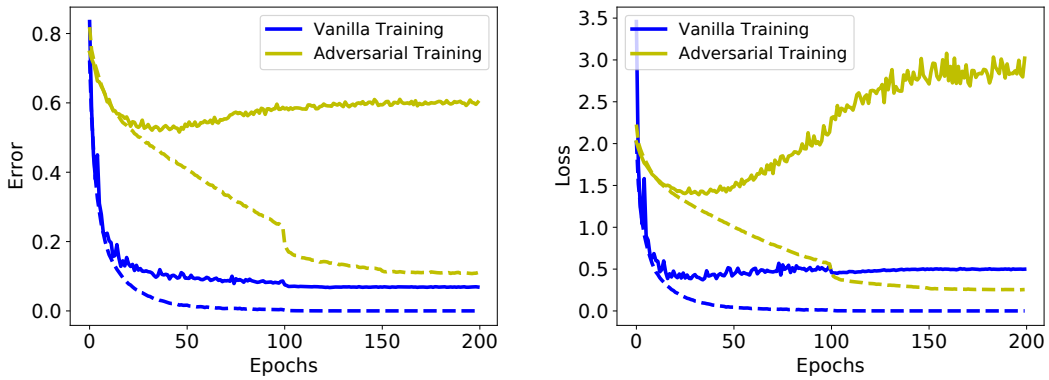


Figure 4.1 – Learning curves in error (left) and loss (right) of vanilla training and adversarial training. Dashed lines and solid lines represent the training set and the test set, respectively.

Slow convergence means higher computational cost for training. In this regard, there are a series of works that accelerate adversarial training [130, 158]. The overfitting issue of adversarial training is first discussed in [118] and it proposes early stopping as the most effective solution. In addition, knowledge distillation, model smoothing [21], instance-adaptive adversarial budget [8] and self-adaptive training [71] are shown successful to mitigate adversarial overfitting. Despite effective, the fundamental reasons that cause slow convergence and severe overfitting have not been investigated yet. In the following sections, we will take a closer look at these two phenomenons by rigorous theoretical analysis and comprehensive empirical observation.

4.2 Adversarial Loss Landscape

Different behavior of vanilla training and adversarial training indicates discrepancies in the underlying optimization landscapes. The loss landscape of vanilla training has been extensively studies in existing literatures[40, 44, 47, 49, 91], such an analysis in adversarial training remains unaddressed. Our work [93] presented in this section is the first one to investigate the properties of loss landscape in the context of adversarial training.

To simplify the notation, we focus on l_p norm based adversarial budget $\mathbb{S}_\epsilon^{(p)}$ and reparameterize the problem of robust learning in (1.2) as follows:

$$\min_\theta \mathcal{L}_\epsilon(\theta) := \frac{1}{N}\sum_{i=1}^{N} g_\epsilon(\theta, \mathbf{x}_i) \text{ where } \mathrm{g}_\epsilon(\theta, \mathbf{x}_\mathrm{i}) := \max_{\Delta_\mathrm{i} \in \mathbb{S}_\epsilon^{(\mathrm{p})}(\mathbf{x}_\mathrm{i})} \mathcal{L}(\mathrm{f}(\theta, \mathbf{x}_\mathrm{i} + \Delta_\mathrm{i}), \mathrm{y}_\mathrm{i}) \,. \tag{4.1}$$

Here, we use L_ϵ and g_ϵ to represent the adversarial loss function for the whole dataset and a specific instance, respectively. In this regard, L_0 and g_0 are the corresponding vanilla loss functions. We discard the label y_i and the superscript $^{(p)}$ for simplicity unless ambiguous.

In the following sections, we will start our analysis with a toy linear model and then generalize our findings to general nonlinear models. In addition to theoretical analysis, we also provide empirical results to confirm our claims.

4.2.1 Toy Model: Logistic Regression

Binary Classification

We start our analysis in logistic regression with the simplest case: binary classification, i.e., $K = 2$. In this case, the model parameter is a vector $\theta = \{\mathbf{w}\}$, $\mathbf{w} \in \mathbb{R}^M$, so the logit function $f(\mathbf{w}, \mathbf{x}) = \left[\mathbf{w}^T\mathbf{x}, -\mathbf{w}^T\mathbf{x}\right]$. If we use $+1$ and -1 to label both classes, then the vanilla loss function for a data set $\left\{(\mathbf{x}_i, y_i)\right\}_{i=1}^N$ is $\mathcal{L}_0(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^N \log\left(1 + e^{-y_i\mathbf{w}^T\mathbf{x}_i}\right)$. Under the adversarial budget $\mathbb{S}_\epsilon^{(p)}$, the corresponding adversarial loss function is $\mathcal{L}_\epsilon(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^N \log\left(1 + e^{-y_i\mathbf{w}^T\mathbf{x}_i + \epsilon\|\mathbf{w}\|_q}\right)$, where l_q is the dual norm of l_p. Since the magnitude of $\mathbf{w}$ does not change the results of the classifier, we can assume $\|\mathbf{w}\|_q = 1$ without loss of generality. As a result, the adversarial loss function is:

$$\mathcal{L}_\epsilon(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} \log\left(1 + e^{-y_i\mathbf{w}^T\mathbf{x}_i + \epsilon}\right) . \tag{4.2}$$

The following theorem describes the properties of $\mathcal{L}_\epsilon(\mathbf{w})$ for different values of ϵ.

Theorem 4.1 *If the dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$ is linearly separable under the adversarial budget $\mathbb{S}_{\widehat{\epsilon}}$, then for any unit vector $\mathbf{m} \in \mathbb{R}^M$ and values ϵ_1, ϵ_2 such that $\epsilon_1 \le \epsilon_2 \le \widehat{\epsilon}$, we have $\mathbf{m}^T\nabla^2_{\mathbf{w}}\mathcal{L}_{\epsilon_1}(\mathbf{w})\mathbf{m} \le \mathbf{m}^T\nabla^2_{\mathbf{w}}\mathcal{L}_{\epsilon_2}(\mathbf{w})\mathbf{m}$. More specifically, both the largest and smallest eigenvalue of $\nabla^2_{\mathbf{w}}\mathcal{L}_{\epsilon_1}(\mathbf{w})$ are no greater than those of $\nabla^2_{\mathbf{w}}\mathcal{L}_{\epsilon_2}(\mathbf{w})$.*

Since $\mathbf{m}^T \nabla^2_{\mathbf{w}} \mathcal{L}_\epsilon(\mathbf{w})\mathbf{m}$ is the curvature of $\mathcal{L}_\epsilon(\mathbf{w})$ in the direction of $\mathbf{m}$, Theorem 4.1 shows that the curvature of $\mathcal{L}_\epsilon(\mathbf{w})$ increases with ϵ in any direction if the whole dataset is linearly separable. For an individual data point $\mathbf{x}$, if $\forall \Delta \in \mathcal{S}_{\widehat{\epsilon}}$, $\mathbf{x}+\Delta$ is correctly classified, then the curvature of $g_\epsilon(\mathbf{w},\mathbf{x})$ also increases with ϵ in any direction as long as $\epsilon \leq \widehat{\epsilon}$. The assumption for an individual point here is much weaker than the one in Theorem 4.1. If the overwhelming majority of the data points are correctly classified under the adversarial budget $\mathcal{S}_{\widehat{\epsilon}}$, the conclusion of Theorem 4.1 still holds in practice. To prove Theorem 4.1, we first prove the following lemma:

Lemma 4.1 *Given a vector set* $\{\mathbf{x}_i\}_{i=1}^N$ *and scalar sets* $\{a_i\}_{i=1}^N$, $\{b_i\}_{i=1}^N$, *we define* $\mathbf{A} = \sum_{i=1}^N a_i \mathbf{x}_i \mathbf{x}_i^T$ *and* $\mathbf{B} = \sum_{i=1}^N b_i \mathbf{x}_i \mathbf{x}_i^T$. *If,* $\forall i,\ a_i \geq b_i$, *then* $\forall \mathbf{m} \in \mathbb{R}^M$, $\mathbf{m}^T\mathbf{A}\mathbf{m} \geq \mathbf{m}^T\mathbf{B}\mathbf{m}$. *In addition, the largest and the smallest eigenvalues of* $\mathbf{A}$ *is no smaller than those of* $\mathbf{B}$.

Proof: Because $\forall\ i, a_i \geq b_i$, we have $\forall \mathbf{m} \sum_{i=1}^N (a_i - b_i)(\mathbf{x}_i^T\mathbf{m})^2 \geq 0$, which can be re-organized into $\mathbf{m}^T\mathbf{A}\mathbf{m} \geq \mathbf{m}^T\mathbf{B}\mathbf{m}$. Consider $\mathbf{A}$, $\mathbf{B}$ are both symmetric matrices, we have the following for their largest eigenvalues $\lambda_1(\mathbf{A})$, $\lambda_1(\mathbf{B})$.

$$\max_{\|\mathbf{m}\|_2=1} \mathbf{m}^T\mathbf{A}\mathbf{m} = \max_{\|\mathbf{m}\|_2=1} \sum_{i=1}^N a_i (\mathbf{x}_i^T\mathbf{m})^2 \,.\ \max_{\|\mathbf{m}\|_2=1} \mathbf{m}^T\mathbf{B}\mathbf{m} = \max_{\|\mathbf{m}\|_2=1} \sum_{i=1}^N b_i (\mathbf{x}_i^T\mathbf{m})^2 \,. \tag{4.3}$$

Let $\mathbf{m}_\mathbf{B} \in \operatorname{argmax}_{\|\mathbf{m}\|_2=1} \sum_{i=1}^N b_i (\mathbf{x}_i^T\mathbf{m})^2$, then we have:

$$\lambda_1(\mathbf{B}) = \sum_{i=1}^N b_i (\mathbf{x}_i^T\mathbf{m}_\mathbf{B})^2 \leq \sum_{i=1}^N a_i (\mathbf{x}_i^T\mathbf{m}_\mathbf{B})^2 \leq \max_{\|\mathbf{m}\|_2=1} \sum_{i=1}^N a_i (\mathbf{x}_i^T\mathbf{m})^2 = \lambda_1(\mathbf{A}) \,. \tag{4.4}$$

We can use the similar way to prove the claim about the smallest eigenvalue, the smallest eigenvalue of $\mathbf{A}$ and $\mathbf{B}$ are $\lambda_M(\mathbf{A}) = \min_{\|\mathbf{m}\|_2=1} \sum_{i=1}^N a_i (\mathbf{x}_i^T\mathbf{m})^2$ and $\lambda_M(\mathbf{B}) = \min_{\|\mathbf{m}\|_2=1} \sum_{i=1}^N b_i (\mathbf{x}_i^T\mathbf{m})^2$, respectively. Let $\mathbf{m}_\mathbf{A} \in \operatorname{argmin}_{\|\mathbf{m}\|_2=1} \sum_{i=1}^N a_i (\mathbf{x}_i^T\mathbf{m})^2$, then we have:

$$\lambda_m(\mathbf{A}) = \sum_{i=1}^N a_i (\mathbf{x}_i^T\mathbf{m}_\mathbf{A})^2 \geq \sum_{i=1}^N b_i (\mathbf{x}_i^T\mathbf{m}_\mathbf{A})^2 \geq \min_{\|\mathbf{m}\|_2=1} \sum_{i=1}^N b_i (\mathbf{x}_i^T\mathbf{m})^2 = \lambda_m(\mathbf{B}) \,. \tag{4.5}$$

□

Now, we go back to prove Theorem 4.1.

Proof: We first calculate the first and second derivatives of $\mathcal{L}_\epsilon(\mathbf{w})$ as:

$$\begin{aligned} \nabla_\mathbf{w} \mathcal{L}_\epsilon(\mathbf{w}) &= \frac{1}{N} \sum_{i=1}^N -\frac{1}{1+e^{y_i \mathbf{w}^T\mathbf{x}_i - \epsilon}} y_i \mathbf{x}_i \,, \\ \nabla^2_\mathbf{w} \mathcal{L}_\epsilon(\mathbf{w}) &= \frac{1}{N} \sum_{i=1}^N \frac{e^{y_i \mathbf{w}^T\mathbf{x}_i - \epsilon}}{(1+e^{y_i \mathbf{w}^T\mathbf{x}_i - \epsilon})^2} y_i^2 \mathbf{x}_i \mathbf{x}_i^T = \frac{1}{N} \sum_{i=1}^N \frac{e^{y_i \mathbf{w}^T\mathbf{x}_i - \epsilon}}{(1+e^{y_i \mathbf{w}^T\mathbf{x}_i - \epsilon})^2} \mathbf{x}_i \mathbf{x}_i^T \,. \end{aligned} \tag{4.6}$$

The second equality of $\nabla^2_{\mathbf{w}}\mathcal{L}_\epsilon(\mathbf{w})$ is satisfied because y_i is either $+1$ or -1. The dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$ is linearly separable under adversarial budget $\mathcal{S}^{(p)}_{\widehat{\epsilon}}$, so, $\forall i,\ y_i\mathbf{w}^T\mathbf{x}_i \geq \widehat{\epsilon}$. When $\epsilon \leq \widehat{\epsilon}$, $e^{y_i\mathbf{w}^T\mathbf{x}_i-\epsilon} > 1$ and monotonically decreases with ϵ. As a result, $\frac{e^{y_i\mathbf{w}^T\mathbf{x}_i-\epsilon}}{(1+e^{y_i\mathbf{w}^T\mathbf{x}_i-\epsilon})^2}$ monotonically increases with ϵ in the range $[0,\widehat{\epsilon}]$.

Based on Lemma 4.1, $\forall \mathbf{m} \in \mathbb{R}^M$, $\mathbf{m}^T\nabla^2_{\mathbf{w}}\mathcal{L}_\epsilon(\mathbf{w})\mathbf{m}$ increases with ϵ, and so do the largest and the smallest eigenvalues of the Hessian matrix $\nabla^2_{\mathbf{w}}\mathcal{L}_\epsilon(\mathbf{w})$. □

Multi-class Classification

For logistic regression for multi-class classification, i.e., $K \geq 3$. We parameterize the model by $\theta = \{\mathbf{W}\}$, $\mathbf{W} := \{\mathbf{w}_i\}_{i=1}^K \in \mathbb{R}^{M\times K}$ and use $f(\mathbf{W}) = \left[\mathbf{w}_1^T\mathbf{x}, \mathbf{w}_2^T\mathbf{x}, ..., \mathbf{w}_K^T\mathbf{x}\right]$ to represent the model's output. Therefore, the vanilla loss is convex as $g_0(\mathbf{W},\mathbf{x}) = \log(1+\sum_{j\neq y}\exp^{(\mathbf{w}_j-\mathbf{w}_y)^T\mathbf{x}})$.

In the adversarial case, although $g_\epsilon(\mathbf{W},\mathbf{x})$ is also convex, it is no longer smooth everywhere. It is then difficult to write a unified expression of $g_\epsilon(\mathbf{W},\mathbf{x})$. So we start with the *version space* $\mathcal{V}_\epsilon$ of $g_\epsilon(\mathbf{W},\mathbf{x})$ defined as $\mathcal{V}_\epsilon := \left\{\mathbf{W}\middle|(\mathbf{w}_i-\mathbf{w}_y)(\mathbf{x}+\Delta) \leq 0, \forall i \in \{0,1,...,K-1\}, \Delta \in \mathcal{S}^{(p)}_\epsilon\right\}$.

By definition, $\mathcal{V}_\epsilon$ is the smallest convex closed set containing all solutions robust under the adversarial budget $\mathcal{S}^{(p)}_\epsilon$. The proposition below states that the version space $\mathcal{V}_\epsilon$ shrinks with larger values of ϵ.

Proposition 4.1 *Given the definition of the version space $\mathcal{V}_\epsilon$, then $\mathcal{V}_{\epsilon_2} \subseteq \mathcal{V}_{\epsilon_1}$ when $\epsilon_1 \leq \epsilon_2$.*

Proposition 4.1 is easy to prove, since $g_{\epsilon_1}(\mathbf{W},\mathbf{x}) \leq g_{\epsilon_2}(\mathbf{W},\mathbf{x})$ for $\epsilon_1 \leq \epsilon_2$, and it is clear that $\forall \mathbf{W} \in \mathcal{V}_\epsilon$, $g_\epsilon(\mathbf{W},\mathbf{x}) \leq \log K$. This shows that the set of solutions for each individual data point becomes smaller as ϵ increases.

In addition to $\mathcal{V}_\epsilon$, we define $\mathcal{T}_\epsilon := \left\{\mathbf{W}\middle|0 \in \text{argmin}_\gamma g_\epsilon(\gamma\mathbf{W},\mathbf{x})\right\}$. $\mathcal{T}_\epsilon$ is the set of all directions in which the optimal point is the origin; that is, the corresponding models in this direction are all no better than a constant classifier. $\mathcal{T}_\epsilon$ is the complement space of $\mathcal{V}_\epsilon$ in binary classification. However, this is not true anymore when $K \geq 3$. Although we cannot write the set $\mathcal{T}_\epsilon$ in roster notation, we show in the theorem below that $\mathcal{T}_\epsilon$ becomes larger as ϵ increases.

Theorem 4.2 *Given the definition of $\mathcal{T}_\epsilon$, then $\mathcal{T}_{\epsilon_2} \subseteq \mathcal{T}_{\epsilon_1}$ when $\epsilon_1 \geq \epsilon_2$. In addition, $\exists\overline{\epsilon}$ such that $\forall\epsilon \geq \overline{\epsilon}, \mathcal{T}_\epsilon = \mathbb{R}^{M\times K}$. In this case, $\mathbf{0} \in \text{argmin}_{\mathbf{W}} g_\epsilon(\mathbf{W},\mathbf{x})$.*

Theorem 4.2 indicates that when the adversarial budget is large enough, the optimal point is the original point $\mathbf{0}$. In this case, we will get a constant classifier, and training completely fails. Although Theorem 4.2 is based on the linear models, we have the same observation in the nonlinear cases. We prove Theorem 4.2 below, where we also provide a lower bound for $\overline{\epsilon}$.

Proof: Based on the convexity of $g_\epsilon(\mathbf{W},\mathbf{x})$, for any $\mathbf{W} \in \mathcal{T}_\epsilon$, the statement $\mathbf{0} \in \text{argmin}_\gamma g_\epsilon(\gamma\mathbf{W},\mathbf{x})$ is equivalent to the following statement:

$$\forall \widehat{\gamma} > 0, g_\epsilon(\widehat{\gamma}\mathbf{W},\mathbf{x}) \geq g_\epsilon(\mathbf{0},\mathbf{x}) \text{ and } g_\epsilon(-\widehat{\gamma}\mathbf{W},\mathbf{x}) \geq g_\epsilon(\mathbf{0},\mathbf{x}) \,. \tag{4.7}$$

Note that $g_\epsilon(\mathbf{0},\mathbf{x}) \equiv \log K$, which means that the loss objective is independent of both the input instance $\mathbf{x}$ and the adversarial budget $\mathcal{S}_\epsilon^{(p)}$ when $\mathbf{W} = \mathbf{0}$. Given $\epsilon_1 \geq \epsilon_2$, we have, $\forall \mathbf{x}, \mathbf{W}$, $g_{\epsilon_1}(\mathbf{W},\mathbf{x}) \geq g_{\epsilon_2}(\mathbf{W},\mathbf{x})$. Therefore, for an arbitrary $\mathbf{W} \in \mathcal{T}_{\epsilon_2}$, we have the following inequality:

$$\forall \widehat{\gamma} > 0, g_{\epsilon_1}(\widehat{\gamma}\mathbf{W},\mathbf{x}) \geq g_{\epsilon_2}(\widehat{\gamma}\mathbf{W},\mathbf{x}) \geq g_{\epsilon_2}(\mathbf{0},\mathbf{x}) = g_{\epsilon_1}(\mathbf{0},\mathbf{x}) \,. \tag{4.8}$$

The first inequality is based on $\epsilon_1 \geq \epsilon_2$, the second one is based on (4.7) and the last one arises from the fact that, $\forall \epsilon, g_\epsilon(\mathbf{0},\mathbf{x})$ is a constant. Similarly, we also have $g_{\epsilon_1}(-\widehat{\gamma}\mathbf{W},\mathbf{x}) \geq g_{\epsilon_1}(\mathbf{0},\mathbf{x})$. Therefore, we have $\mathbf{W} \in \mathcal{T}_{\epsilon_1}$, which means $\mathcal{T}_{\epsilon_2} \subseteq \mathcal{T}_{\epsilon_1}$.

To prove the second half of Theorem 4.2, one barrier is that we do not have an analytical form for $g_\epsilon(\mathbf{W},\mathbf{x})$. Instead, we introduce a lower bound $\underline{g_\epsilon}(\mathbf{W},\mathbf{x})$ of $g_\epsilon(\mathbf{W},\mathbf{x})$, which has an analytical form. We consider the perturbation $\Delta = \epsilon \frac{(\mathbf{w}_m - \mathbf{w}_y)^{\frac{q}{p}}}{\|\mathbf{w}_m - \mathbf{w}_y\|_q^{\frac{q}{p}}}$ [1], where $m = \text{argmax}_j \|\mathbf{w}_j - \mathbf{w}_y\|_q$. It can be verified that $\Delta \in \mathcal{S}_\epsilon^{(p)}$, so we let $\underline{g_\epsilon}(\mathbf{W},\mathbf{x}) = g(\mathbf{W},\mathbf{x}+\Delta)$, which is a valid lower bound of $g_\epsilon(\mathbf{W},\mathbf{x})$. Then, the analytical expression of $\underline{g_\epsilon}(\mathbf{W},\mathbf{x})$ can be written as:

$$\underline{g_\epsilon}(\mathbf{W},\mathbf{x}) = \log\left(1 + \exp^{(\mathbf{w}_m - \mathbf{w}_y)\mathbf{x} + \epsilon\|\mathbf{w}_m - \mathbf{w}_y\|_q} + \sum_{j\neq y, j\neq m} \exp^{(\mathbf{w}_j - \mathbf{w}_y)\mathbf{x} + \epsilon(\mathbf{w}_j - \mathbf{w}_y)\frac{(\mathbf{w}_m - \mathbf{w}_y)^{\frac{q}{p}}}{\|\mathbf{w}_m - \mathbf{w}_y\|_q^{\frac{q}{p}}}}\right). \tag{4.9}$$

Since $m = \text{argmax}_j \|\mathbf{w}_j - \mathbf{w}_y\|_q$, then $(\mathbf{w}_j - \mathbf{w}_y)\frac{(\mathbf{w}_m - \mathbf{w}_y)^{\frac{q}{p}}}{\|\mathbf{w}_m - \mathbf{w}_y\|_q^{\frac{q}{p}}} \leq \|\mathbf{w}_m - \mathbf{w}_y\|_q$. As a result, if ϵ is large enough, the second term inside the logarithm of (4.9) will dominate the summation and $\lim_{\epsilon\to\infty} \underline{g_\epsilon}(\mathbf{W},\mathbf{x}) = \infty$. More specifically, we can find $\overline{\epsilon} = \frac{\log(K-1) - (\mathbf{w}_m - \mathbf{w}_y)\mathbf{x}}{\|\mathbf{w}_m - \mathbf{w}_y\|_q}$, such that, $\forall \epsilon > \overline{\epsilon}$, $\mathbf{W}$, then $\underline{g_\epsilon}(\mathbf{W},\mathbf{x}) \geq \log K = g_\epsilon(\mathbf{0},\mathbf{x})$.

Now, $\forall \mathbf{W} \in \mathbb{R}^{M\times K}, \widehat{\gamma} > 0, \epsilon \geq \overline{\epsilon}$, we have $g_\epsilon(\widehat{\gamma}\mathbf{W},\mathbf{x}) \geq \underline{g_\epsilon}(\widehat{\gamma}\mathbf{W},\mathbf{x}) \geq g_\epsilon(\mathbf{0},\mathbf{x})$. Similarly, we have $g_\epsilon(-\widehat{\gamma}\mathbf{W},\mathbf{x}) \geq g_\epsilon(\mathbf{0},\mathbf{x})$. As a result, we have, $\forall \mathbf{W} \in \mathbb{R}^{M\times K}, g_\epsilon(\mathbf{W},\mathbf{x}) \geq g_\epsilon(\mathbf{0},\mathbf{x})$, which means $\mathbf{0} \in \text{argmin}_{\mathbf{W}} g_\epsilon(\mathbf{W},\mathbf{x})$. Based on (4.7), we have $\mathcal{T}_\epsilon = \mathbb{R}^{M\times K}$.

□

$\mathcal{L}_\epsilon(\mathbf{W})$ is the average of $g_\epsilon(\mathbf{W},\mathbf{x})$ over the dataset, so Theorem 4.2 and Proposition 4.1 still apply if we replace g_ϵ with $\mathcal{L}_\epsilon$ in the definition of $\mathcal{V}_\epsilon$ and $\mathcal{T}_\epsilon$. For nonlinear models like deep neural

[1] l_q is the dual norm of l_p, i.e., $\frac{1}{p} + \frac{1}{q} = 1$

networks, these conclusions will not hold because $g_\epsilon(\theta,\mathbf{x})$ is no longer convex. Nevertheless, our experiments in Section 4.2.3 evidence the same phenomena as indicated by the theoretical analysis above. That is to say, larger ϵ makes it harder for the optimizer to escape the initial suboptimal region. In some cases, adversarial training fails completely, and we obtain a constant classifier in the end.

4.2.2 Theoretical Analysis for General Models

For deep nonlinear neural networks, we cannot write the analytical form of $g_\epsilon(\theta,\mathbf{x})$ or $\mathcal{L}_\epsilon(\theta,\mathbf{x})$. To analyze such models, we follow [138] and assume the smoothness of the function g_0.

Assumption 4.1 *The vanilla loss function g_0 satisfies the following Lipschitzian smoothness conditions:*

$$\begin{aligned} \|g_0(\theta_1,\mathbf{x}) - g_0(\theta_2,\mathbf{x})\| &\le L_\theta \|\theta_1-\theta_2\|\,, \\ \|\nabla_\theta g_0(\theta_1,\mathbf{x}) - \nabla_\theta g_0(\theta_2,\mathbf{x})\| &\le L_{\theta\theta}\|\theta_1-\theta_2\|\,, \\ \|\nabla_\theta g_0(\theta,\mathbf{x}_1) - \nabla_\theta g_0(\theta,\mathbf{x}_2)\| &\le L_{\theta\mathbf{x}}\|\mathbf{x}_1-\mathbf{x}_2\|_p\,. \end{aligned} \tag{4.10}$$

Assumption 4.1 indicates the first-order smoothness and second-order smoothness of the vanilla loss function g_0. Now, we study the smoothness of the adversarial loss function $\mathcal{L}_\epsilon(\theta)$.

Proposition 4.2 *If Assumption 4.1 holds, then we have*[2]

$$\begin{aligned} \|\mathcal{L}_\epsilon(\theta_1) - \mathcal{L}_\epsilon(\theta_2)\| &\le L_\theta\|\theta_1-\theta_2\|\,, \\ \|\nabla_\theta\mathcal{L}_\epsilon(\theta_1) - \nabla_\theta\mathcal{L}_\epsilon(\theta_2)\| &\le L_{\theta\theta}\|\theta_1-\theta_2\| + 2\epsilon L_{\theta\mathbf{x}}\,. \end{aligned} \tag{4.11}$$

Proof: Consider $\mathcal{L}_\epsilon(\theta)$ is the average of $g_\epsilon(\theta,\mathbf{x})$ over the dataset. Therefore, to prove Proposition 4.2, we only need to prove the following inequality for any data point $\mathbf{x}$:

$$\begin{aligned} \|g_\epsilon(\theta_1,\mathbf{x}) - g_\epsilon(\theta_2,\mathbf{x})\| &\le L_\theta\|\theta_1-\theta_2\|\,, \\ \|\nabla_\theta g_\epsilon(\theta_1,\mathbf{x}) - \nabla_\theta g_\epsilon(\theta_2,\mathbf{x})\| &\le L_{\theta\theta}\|\theta_1-\theta_2\| + 2\epsilon L_{\theta\mathbf{x}}\,. \end{aligned} \tag{4.12}$$

To prove the first inequality, we introduce the adversarial perturbations for parameter θ_1 and θ_2:

$$\Delta_1 = \operatorname{argmax}_{\Delta\in\mathcal{S}_\epsilon^{(p)}} g_0(\mathbf{x}+\Delta,\theta_1)\,,\ \ \Delta_2 = \operatorname{argmax}_{\Delta\in\mathcal{S}_\epsilon^{(p)}} g_0(\mathbf{x}+\Delta,\theta_2)\,. \tag{4.13}$$

Therefore, $g_\epsilon(\theta_1,\mathbf{x}) = g_0(\theta_1,\mathbf{x}+\Delta_1)$ and $g_\epsilon(\theta_2,\mathbf{x}) = g_0(\theta_2,\mathbf{x}+\Delta_2)$.

[2] Strictly speaking, $\mathcal{L}_\epsilon(\theta)$ is not differentiable at some point, so $\nabla_\theta\mathcal{L}_\epsilon(\theta)$ might be ill-defined. In this paper, we use $\nabla_\theta\mathcal{L}_\epsilon(\theta)$ for simplicity. Nevertheless, the inequality holds for any subgradient $\mathbf{v}\in\partial_\theta\mathcal{L}_\epsilon(\theta)$.

By definition, we have $g_0(\theta_1, \mathbf{x}+\Delta_1) \geq g_0(\theta_1, \mathbf{x}+\Delta_2)$ and $g_0(\theta_2, \mathbf{x}+\Delta_2) \geq g_0(\theta_2, \mathbf{x}+\Delta_1)$. As a result, $\|g_\epsilon(\theta_1, \mathbf{x}) - g_\epsilon(\theta_2), \mathbf{x}\| = \|g_0(\theta_1, \mathbf{x}+\Delta_1) - g_0(\theta_2, \mathbf{x}+\Delta_2)\|$.

If $g_0(\theta_1, \mathbf{x}+\Delta_1) - g_0(\theta_2, \mathbf{x}+\Delta_2) \leq 0$, we have:

$$\begin{aligned} \|g_\epsilon(\theta_1, \mathbf{x}) - g_\epsilon(\theta_2, \mathbf{x})\| &= g_0(\theta_2, \mathbf{x}+\Delta_2) - g_0(\theta_1, \mathbf{x}+\Delta_1) \\ &\leq g_0(\theta_2, \mathbf{x}+\Delta_2) - g_0(\theta_1, \mathbf{x}+\Delta_2) \leq L_\theta \|\theta_1 - \theta_2\| . \end{aligned} \tag{4.14}$$

Similarly, if $g(\theta_1, \mathbf{x}+\Delta_1) - g(\theta_2, \mathbf{x}+\Delta_2) \geq 0$, we have:

$$\begin{aligned} \|g_\epsilon(\theta_1, \mathbf{x}) - g_\epsilon(\theta_2, \mathbf{x})\| &= g_0(\theta_1, \mathbf{x}+\Delta_1) - g_0(\theta_2, \mathbf{x}+\Delta_2) \\ &\leq g_0(\theta_1, \mathbf{x}+\Delta_1) - g_0(\theta_2, \mathbf{x}+\Delta_1) \leq L_\theta \|\theta_1 - \theta_2\| . \end{aligned} \tag{4.15}$$

The first inequality in (4.12) is then proved. The bound is tight, and equality is achieved when, for example, $\Delta_1 = \Delta_2$.

The second inequality in (4.12) is more straightforward. We have:

$$\begin{aligned} &\|\nabla_\theta g_\epsilon(\theta_1, \mathbf{x}) - \nabla_\theta g_\epsilon(\theta_2, \mathbf{x})\| \\ &\quad = \|\nabla_\theta g_0(\theta_1, \mathbf{x}+\Delta_1) - \nabla_\theta g_0(\theta_2, \mathbf{x}+\Delta_2)\| \\ &\quad = \|\nabla_\theta g_0(\theta_1, \mathbf{x}+\Delta_1) - \nabla_\theta g_0(\theta_2, \mathbf{x}+\Delta_1) + \nabla_\theta g_0(\theta_2, \mathbf{x}+\Delta_1) - \nabla_\theta g_0(\theta_2, \mathbf{x}+\Delta_2)\| \\ &\quad \leq \|\nabla_\theta g_0(\theta_1, \mathbf{x}+\Delta_1) - \nabla_\theta g_0(\theta_2, \mathbf{x}+\Delta_1)\| + \|\nabla_\theta g_0(\theta_2, \mathbf{x}+\Delta_1) - \nabla_\theta g_0(\theta_2, \mathbf{x}+\Delta_2)\| \\ &\quad \leq L_{\theta\theta}\|\theta_1 - \theta_2\| + L_{\theta\mathbf{x}}\|\Delta_1 - \Delta_2\|_p \\ &\quad \leq L_{\theta\theta}\|\theta_1 - \theta_2\| + 2\epsilon L_{\theta\mathbf{x}} . \end{aligned} \tag{4.16}$$

The last inequality in (4.16) is satisfied because both Δ_1 and Δ_2 belong to $\mathcal{S}_\epsilon^{(p)}$. This bound is tight, and equality is reached only when $\|\Delta_1 - \Delta_2\|_p = 2\epsilon$.

□

Proposition 4.2 shows that the first-order smoothness of the objective function is preserved for the adversarial case, but the second-order smoothness is not. This unsatisfying property arises from the maximization operator defined in the functions g_ϵ and $\mathcal{L}_\epsilon$. For function $g_\epsilon(\theta, \mathbf{x})$, the non-smooth points are those where the optimal adversarial perturbation Δ changes abruptly in a sufficiently small neighborhood. Formally, we use θ_1 and Δ_1 to represent the model parameters and the corresponding optimal adversarial perturbation. We assume different gradients of the model parameters for different input instances. If there exists a positive number $a > 0$ such that, $\forall \delta > 0$, we can find $\theta_2 \in \{\theta | \|\theta - \theta_1\| \leq \delta\}$, and the corresponding optimal adversarial perturbation Δ_2 satisfies $\|\Delta_1 - \Delta_2\|_p > a$, then $\lim_{\theta \to \theta_1} \nabla_\theta g_\epsilon(\theta, \mathbf{x}) \neq \nabla_\theta g_\epsilon(\theta_1, \mathbf{x})$. $\mathcal{L}_\epsilon(\theta)$ is the

aggregation of $g_\epsilon(\theta, \mathbf{x})$ over the dataset, so its non-smooth points in the parameter space are the union of the non-smooth points for each data sample. In addition, as the $2\epsilon L_{\theta\mathbf{x}}$ term in the second inequality of (4.11) indicates, the adversarial examples can change more under a larger adversarial budget. As a result, the (sub)gradients $\nabla_\theta \mathcal{L}_\epsilon(\theta)$ can change more abruptly in the neighborhood of the parameter space. That is, the (sub)gradients are more *scattered* in the adversarial loss landscape. We provide a sketch illustrating the phenomenon in Figure 4.2 below.

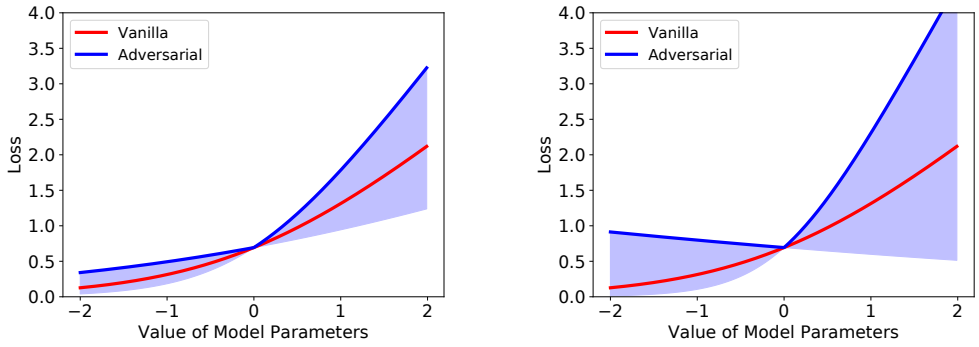


Figure 4.2 – 2D sketch diagram showing the vanilla and the adversarial loss landscape. The clean input data x is 1.0 and loss function $g_0(\theta, x) = \log(1 + \exp(\theta x))$. The landscape is shown in the parameter interval $\theta \in [-2, 2]$ under the small adversarial budget (left, $\epsilon = 0.6$) and the large adversarial budget (right, $\epsilon = 1.2$). Function $g_\epsilon(\theta, x)$ is not smooth at $\theta = 0$.

The non-smoothness nature of the adversarial loss landscape makes the optimization by stochastic gradient descent (SGD) more difficult. We study the convergence property in the following theorem, which explains why adversarial training converges slower.

Theorem 4.3 *Let Assumption 4.1 hold, the stochastic gradient $\nabla_\theta \widehat{\mathcal{L}}_\epsilon(\theta_t)$ be unbiased and have bounded variance, and the SGD update $\theta_{t+1} = \theta_t - \alpha_t \nabla_\theta \widehat{\mathcal{L}}_\epsilon(\theta_t)$ use a constant step size $\alpha_t = \alpha = \frac{1}{L_{\theta\theta}\sqrt{T}}$ for T iterations. Given the trajectory of the parameters during optimization $\{\theta_t\}_{t=1}^T$, then we can bound the asymptotic probability of large gradients for a sufficient large value of T as:*

$$\forall \gamma \geq 2, P(\|\nabla_\theta \mathcal{L}_\epsilon(\theta_t)\| > \gamma \epsilon L_{\theta\mathbf{x}}) < \frac{4}{\gamma^2 - 2\gamma + 4}\,. \tag{4.17}$$

Proof: Let σ^2 to denote the variance of stochastic gradient $\nabla_\theta \widehat{\mathcal{L}}_\epsilon(\theta)$. Based on the assumption that $\nabla_\theta \widehat{\mathcal{L}}_\epsilon(\theta)$ is unbiased, we have:

$$\begin{aligned} \mathbb{E}[\nabla_\theta \widehat{\mathcal{L}}_\epsilon(\theta)] &= \nabla_\theta \mathcal{L}_\epsilon(\theta)\,, \\ \mathbb{E}\|\nabla_\theta \widehat{\mathcal{L}}_\epsilon(\theta)\|^2 &= \|\nabla_\theta \mathcal{L}_\epsilon(\theta)\|^2 + \sigma^2\,. \end{aligned} \tag{4.18}$$

Proposition 4.2 shows that $\mathcal{L}_\epsilon(\theta)$ is continuous. Therefore, we introduce $\widetilde{\theta}_t(u) = \theta_t + u(\theta_{t+1} - \theta_t)$ and derive an upper bound of $\mathcal{L}_\epsilon(\theta_{t+1}) - \mathcal{L}_\epsilon(\theta_t)$ by first order Taylor expansion and using the

update rule $\theta_{t+1} = \theta_t - \alpha_t \nabla_\theta \widehat{\mathcal{L}}_\epsilon(\theta_t)$. This yields:

$$\begin{aligned}
\mathcal{L}_\epsilon(\theta_{t+1}) - \mathcal{L}_\epsilon(\theta_t) &= \int_0^1 \langle \theta_{t+1} - \theta_t, \nabla_\theta \mathcal{L}_\epsilon(\widetilde{\theta}_t(u)) \rangle d_u \\
&= \int_0^1 \langle -\alpha_t \nabla_\theta \widehat{\mathcal{L}}_\epsilon(\theta_t), \nabla_\theta \mathcal{L}_\epsilon(\widetilde{\theta}_t(u)) \rangle d_u \\
&= \int_0^1 \langle -\alpha_t \nabla_\theta \widehat{\mathcal{L}}_\epsilon(\theta_t), \nabla_\theta \mathcal{L}_\epsilon(\widetilde{\theta}_t(u)) - \nabla_\theta \mathcal{L}_\epsilon(\theta_t) \rangle d_u + \langle -\alpha_t \nabla_\theta \widehat{\mathcal{L}}_\epsilon(\theta_t), \nabla_\theta \mathcal{L}_\epsilon(\theta_t) \rangle \\
&\leq \int_0^1 \alpha_t \| \nabla_\theta \widehat{\mathcal{L}}_\epsilon(\theta_t) \| \| \nabla_\theta \mathcal{L}_\epsilon(\widetilde{\theta}_t(u)) - \nabla_\theta \mathcal{L}_\epsilon(\theta_t) \| d_u - \alpha_t \langle \nabla_\theta \widehat{\mathcal{L}}_\epsilon(\theta_t), \nabla_\theta \mathcal{L}_\epsilon(\theta_t) \rangle \\
&\leq \int_0^1 \alpha_t \| \nabla_\theta \widehat{\mathcal{L}}_\epsilon(\theta_t) \| (L_{\theta\theta} \| \widetilde{\theta}_t(u) - \theta_t \| + 2\epsilon L_{\theta \mathbf{x}}) d_u - \alpha_t \langle \nabla_\theta \widehat{\mathcal{L}}_\epsilon(\theta_t), \nabla_\theta \mathcal{L}_\epsilon(\theta_t) \rangle \\
&= \frac{1}{2} \alpha_t^2 L_{\theta\theta} \| \nabla_\theta \widehat{\mathcal{L}}_\epsilon(\theta_t) \|^2 + 2\epsilon L_{\epsilon \mathbf{x}} \alpha_t \| \nabla_\theta \widehat{\mathcal{L}}_\epsilon(\theta_t) \| - \alpha_t \langle \nabla_\theta \widehat{\mathcal{L}}_\epsilon(\theta_t), \nabla_\theta \mathcal{L}_\epsilon(\theta_t) \rangle \, .
\end{aligned} \tag{4.19}$$

Here, the first inequality comes from Hölder's Inequality; the second one follows the conclusion of Proposition 4.2.

By taking the expectation over the noise introduced by SGD, we have

$$\begin{aligned}
\mathbb{E}[\mathcal{L}_\epsilon(\theta_{t+1})] - \mathbb{E}[\mathcal{L}_\epsilon(\theta_t)] &\leq \frac{1}{2} \alpha_t^2 L_{\theta\theta} (\| \nabla_\theta \mathcal{L}_\epsilon(\theta_t) \|^2 + \sigma^2) + 2\epsilon L_{\theta \mathbf{x}} \alpha_t \| \nabla_\theta \mathcal{L}_\epsilon(\theta_t) \| - \alpha_t \| \nabla_\theta \mathcal{L}_\epsilon(\theta_t) \|^2 \\
&= \left(\frac{1}{2} \alpha_t^2 L_{\theta\theta} - \alpha_t \right) \| \nabla_\theta \mathcal{L}_\epsilon(\theta_t) \|^2 + 2\epsilon L_{\theta \mathbf{x}} \alpha_t \| \nabla_\theta \mathcal{L}_\epsilon(\theta_t) \| + \frac{1}{2} \alpha_t^2 \sigma^2 L_{\theta\theta} \\
&\leq -\frac{1}{2} \alpha_t \| \nabla_\theta \mathcal{L}_\epsilon(\theta_t) \|^2 + 2\epsilon L_{\theta \mathbf{x}} \alpha_t \| \nabla_\theta \mathcal{L}_\epsilon(\theta_t) \| + \frac{1}{2} \alpha_t^2 \sigma^2 L_{\theta\theta} \, .
\end{aligned} \tag{4.20}$$

We use the approximation $\mathbb{E} \| \nabla_\theta \widehat{\mathcal{L}}_\epsilon(\theta) \| \simeq \| \nabla_\theta \mathcal{L}_\epsilon(\theta) \|$ because the variance arises mainly from the term $\| \nabla_\theta \widehat{\mathcal{L}}_\epsilon(\theta_t) \|^2$. The last inequality is based on the fact that $\alpha_t = \alpha = \frac{1}{L_{\theta\theta}\sqrt{T}}$, so $\alpha_t L_{\theta\theta} = \frac{1}{\sqrt{T}} \leq 1$. Let us now sum (4.20) over $t \in \{0, 1, ..., T-1\}$, then we obtain:

$$\begin{aligned}
\sum_{t=0}^{T} \left[\frac{1}{2} \alpha_t \| \nabla_\theta \mathcal{L}_\epsilon(\theta_t) \|^2 - 2\epsilon L_{\theta \mathbf{x}} \alpha_t \| \nabla_\theta \mathcal{L}_\epsilon(\theta_t) \| \right] &\leq \mathcal{L}_\epsilon(\theta_0) - \mathbb{E}[\mathcal{L}_\epsilon(\theta_T)] + \frac{T}{2} \alpha_t^2 \sigma^2 L_{\theta\theta} \\
&\leq \mathcal{L}_\epsilon(\theta_0) - \mathcal{L}_\epsilon(\theta^*) + \frac{T}{2} \alpha_t^2 \sigma^2 L_{\theta\theta} \, .
\end{aligned} \tag{4.21}$$

We use θ^* to denote the global minimum since $\mathcal{L}_\epsilon(\theta)$ is lower bounded. By introducing $\alpha_t = \alpha = \frac{1}{L_{\theta\theta}\sqrt{T}}$ into the formulation, we obtain:

$$\begin{aligned}
\frac{1}{T} \sum_{t=0}^{T} \left[\frac{1}{2} \| \nabla_\theta \mathcal{L}_\epsilon(\theta_t) \|^2 - 2\epsilon L_{\theta \mathbf{x}} \| \nabla_\theta \mathcal{L}_\epsilon(\theta_t) \| \right] &\leq \frac{1}{\alpha T} \left[\mathcal{L}_\epsilon(\theta_0) - \mathcal{L}_\epsilon(\theta^*) \right] + \frac{1}{2} \alpha \sigma^2 L_{\theta\theta} \\
&= \frac{1}{\sqrt{T}} \left[L_{\theta\theta} (\mathcal{L}_\epsilon(\theta_0) - \mathcal{L}_\epsilon(\theta^*)) + \frac{1}{2} \sigma^2 \right] .
\end{aligned} \tag{4.22}$$

Since the right hand side of (4.22) converges to 0 as $T \to +\infty$, we have

$$\lim_{T\to+\infty} \frac{1}{T}\sum_{t=0}^{T}\left[\frac{1}{2}\|\nabla_\theta \mathcal{L}_\epsilon(\theta_t)\|^2 - 2\epsilon L_{\theta\mathbf{x}}\|\nabla_\theta \mathcal{L}_\epsilon(\theta_t)\|\right] \leq 0\,. \tag{4.23}$$

Let us define $h(\theta_t) = \frac{1}{2}\|\nabla_\theta \mathcal{L}_\epsilon(\theta_t)\|^2 - 2\epsilon L_{\theta\mathbf{x}}\|\nabla_\theta \mathcal{L}_\epsilon(\theta_t)\|$ for notation simplicity. Then, inequality (4.23) shows that $\mathbb{E}_t[h(\theta_t)] \leq 0$ when T is large enough.

Let $\|\nabla_\theta \mathcal{L}_\epsilon(\theta_t)\| = \gamma\epsilon L_{\theta\mathbf{x}}$, then we have $h(\theta_t) = \left(\frac{1}{2}\gamma^2 - 2\gamma\right)\epsilon^2 L_{\theta\mathbf{x}}^2$. $h(\theta_t)$ is monotonically increasing when $\theta_t \geq 2\epsilon L_{\theta\mathbf{x}}$, so when $\gamma \geq 2$, $h(\theta_t) \geq \left(\frac{1}{2}\gamma^2 - 2\gamma\right)\epsilon^2 L_{\theta\mathbf{x}}^2$. Considering $h(\theta_t) \geq -2\epsilon^2 L_{\theta\mathbf{x}}^2$, we then have

$$\mathbb{E}_t[h(\theta_t)] > -2\epsilon^2 L_{\theta\mathbf{x}}^2(1 - P(\|\nabla_\theta \mathcal{L}_\epsilon(\theta_t)\| > \gamma\epsilon L_{\theta\mathbf{x}})) + \left(\frac{1}{2}\gamma^2 - 2\gamma\right)\epsilon^2 L_{\theta\mathbf{x}}^2 P(\|\nabla_\theta \mathcal{L}_\epsilon(\theta_t)\| > \gamma\epsilon L_{\theta\mathbf{x}})\,. \tag{4.24}$$

Finally, by rearranging (4.24) and using $\mathbb{E}_t[h(\theta_t)] \leq 0$, we obtain

$$\forall \gamma > 2,\ P(\|\nabla_\theta \mathcal{L}_\epsilon(\theta_t)\| > \gamma\epsilon L_{\theta\mathbf{x}}) < \frac{4}{\gamma^2 - 2\gamma + 4}\,. \tag{4.25}$$

□

In vanilla training, $\epsilon = 0$ and $\mathcal{L}_0(\theta)$ is smooth. In this regard, (4.17) in Theorem 4.3 implies that $\lim_{t\to+\infty}\|\nabla_\theta g_0(\theta_t)\| = 0$ almost surely. This is consistent with the fact that SGD converges to a critical point with non-convex smooth functions. By contrast, in adversarial training, i.e., $\epsilon > 0$, we cannot guarantee convergence to a critical point any more. Instead, the gradients are non-vanishing, and we can only bound the probability of obtaining gradients whose magnitude is larger than $2\epsilon L_{\theta\mathbf{x}}$. For a fixed value of $C := \gamma\epsilon L_{\theta\mathbf{x}}$ larger than $2\epsilon L_{\theta\mathbf{x}}$, the inequality (4.17) indicates that the probability $P(\|\nabla_\theta \mathcal{L}_\epsilon(\theta_t)\| > C)$ increases quadratically with ϵ.

In deep learning practice, activation functions like sigmoid, tanh and ELU [26] satisfy the second-order smoothness in Assumption 4.1, but the most popular ReLU function does not. Nevertheless, adversarial training still causes *gradient scattering* and makes the optimization more difficult. That is, the bound of $\|\nabla_\theta \mathcal{L}_\epsilon(\theta_1) - \nabla_\theta \mathcal{L}_\epsilon(\theta_2)\|$ still increases with ϵ, and the parameter gradients change abruptly in the adversarial loss landscape. The following corollary indicates that even for non-smooth ReLU networks, adversarial training still converges slower with the increase of ϵ.

Corollary 4.1 *We assume that the function g_0 satisfies the following conditions:*

$$\begin{aligned}
\|g_0(\theta_1,\mathbf{x}) - g_0(\theta_2,\mathbf{x})\| &\leq L_\theta\|\theta_1 - \theta_2\|\,,\\
\|\nabla_\theta g_0(\theta_1,\mathbf{x}) - \nabla_\theta g_0(\theta_2,\mathbf{x})\| &\leq L_{\theta\theta}\|\theta_1 - \theta_2\| + D_{\theta\theta}\,,\\
\|\nabla_\theta g(\theta,\mathbf{x}_1) - \nabla_\theta g(\theta,\mathbf{x}_2)\| &\leq L_{\theta\mathbf{x}}\|\mathbf{x}_1 - \mathbf{x}_2\|_p + D_{\theta\mathbf{x}}\,.
\end{aligned} \tag{4.26}$$

In addition, the stochastic gradient $\nabla_\theta \widehat{\mathcal{L}}_\epsilon(\theta_t)$ is unbiased and have bounded variance. The SGD

update $\theta_{t+1} = \theta_t - \alpha_t \nabla_\theta \widehat{\mathcal{L}}_\epsilon(\theta_t)$ *use a constant step size* $\alpha_t = \alpha = \frac{1}{L_{\theta\theta}\sqrt{T}}$ *for* T *iterations. Given the trajectory of the parameters during optimization* $\{\theta_t\}_{t=1}^T$*, then we can bound the asymptotic probability of large gradients for a sufficient large value of* T *as*

$$\forall \gamma \geq 2, P\left(\|\nabla_\theta \mathcal{L}_\epsilon(\theta_t)\| > \gamma\left(\epsilon L_{\theta\mathbf{x}} + \frac{1}{2}D_{\theta\theta} + \frac{1}{2}D_{\theta\mathbf{x}}\right)\right) < \frac{4}{\gamma^2 - 2\gamma + 4}\,. \tag{4.27}$$

Corollary 4.1 can be proved in the same way as Theorem 4.3.

The second-order Lipchitz constant indicates the magnitude of the gradient change for a unit change in parameters. Therefore, it is a good quantitative metric of gradient scattering. In practice, we are more interested in the effective local Lipschitz constant, which only considers the neighborhood of the current parameters, than in the global Lipschitz constant. In this case, the effective local second-order Lipschitz constant can be estimated by the top eigenvalues of the Hessian matrix $\nabla^2_\theta \mathcal{L}_\epsilon(\theta)$.

4.2.3 Numerical Experiments

In this section, we conduct experiments on MNIST [89] and CIFAR10 [104] to empirically validate the theorems in Section 4.2.1 and Section 4.2.2. Unless specified, we use LeNet [88] on MNIST, VGG [135] and ResNet18 [63] on CIFAR10. We provide the details in Table 4.1 and use a factor w to control the width of the network. Unless specified, the LeNet models on MNIST have a width factor of 16, the VGG and ResNet18 models on CIFAR10 have a width factor of 8.

Name	Architecture
MNIST, LeNet	Conv($2w$), Conv($4w$), FC($196w$, $64w$), FC($64w$, 10)
CIFAR, VGG	Conv($4w$) × 2, M, Conv($8w$) × 2, M, Conv($16w$) × 3, M Conv($32w$) × 3, M, Conv($32w$) × 3, M, A, FC($32w$, 10)
CIFAR, ResNet18	ResNet18 in [63] is denoted of width 16

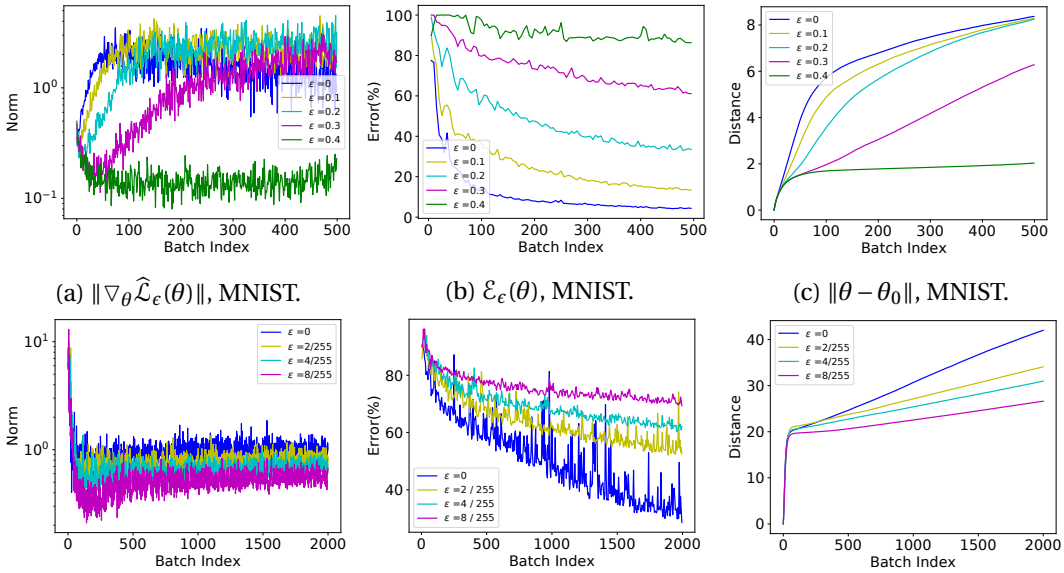
Table 4.1 – Network architectures. Conv, FC, M and A represent convolutional layers, fully-connected layers, max-pooling layers and average pooling layers, respectively. The parameter of the convolutional layers indicates the number of output channels. The parameters of the fully-connected layers indicates the number of input and output neurons. The kernel sizes of the max-pooling layers and average pooling layers are always 2. w is the width factor.

We concentrate on l_∞ adversarial budget and use PGD attacks in adversarial training. For MNIST, we set the step size of PGD to 0.01 and the number of iterations to $\epsilon/0.01 + 10$. For CIFAR10, we set the number of PGD iterations to 10 and the step size to $\epsilon/4$.

Gradient Analysis

In Section 4.2.1, we have shown that the training algorithm will get stuck at the origin and yield a constant classifier for linear models under large ϵ. For deep nonlinear models, the initial

value of the parameters is close to the origin under most popular initialization schemes [52, 62]. Although Theorem 4.2 is not applicable here, we are still interested in investigating how effective gradient-based optimization is at escaping from the suboptimal initial parameters. To this end, we track the norm of the stochastic gradient $\|\nabla_\theta \widehat{\mathcal{L}}_\epsilon(\theta)\|$, the robust error $\mathcal{E}_\epsilon(\theta)$ in the training set and the distance from the initial point $\|\theta - \theta_0\|$ during the first 500 mini-batch updates for LeNet models on MNIST and the first 2000 mini-batch updates for ResNet18 models on CIFAR10. Figure 4.3 evidences a clear difference between the models trained with different values of ϵ. When ϵ is small, the gradient magnitude is larger, and the model parameters move faster. Correspondingly, the training error decreases faster, which means that the model quickly escapes the initial suboptimal region. By contrast, when ϵ is large, the gradients are small, and the model gets stuck in the initial region. This implies that the loss landscape under a large adversarial budget impedes the escape from initial suboptimal plateaus in the early stage of training.



(a) $\|\nabla_\theta \widehat{\mathcal{L}}_\epsilon(\theta)\|$, MNIST. (b) $\mathcal{E}_\epsilon(\theta)$, MNIST. (c) $\|\theta - \theta_0\|$, MNIST.

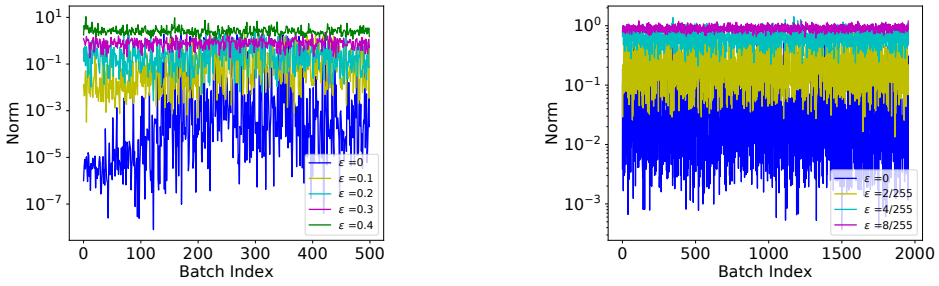
(d) $\|\nabla_\theta \widehat{\mathcal{L}}_\epsilon(\theta)\|$, CIFAR10. (e) $\mathcal{E}_\epsilon(\theta)$, CIFAR10. (f) $\|\theta - \theta_0\|$, CIFAR10.

Figure 4.3 – Norm of the stochastic gradient $\|\nabla_\theta \widehat{\mathcal{L}}_\epsilon(\theta)\|$, robust training error $\mathcal{E}_\epsilon(\theta)$, and distance from the initial point $\|\theta - \theta_0\|$ in the early phase of training. We report first 500 updates for MNIST models and first 2000 updates for CIFAR10 models.

For ReLU networks, adversarially-trained models have been found to have sparser weights and intermediate activations [29], i.e., they have more dead neurons. Dead neurons are implicitly favored by adversarial training, because the output of dead neurons is always zero and independent of the input perturbation. Note that training fails when all the neurons in one layer are dead for all training instances. The model is then effectively broken into two parts by this dead layer: the preceding layers will no longer be trained because the gradients are all blocked; the following layers do not depend on the input and thus give constant outputs. In essence, training is then stuck in a parameter space that only includes constant classifiers.

In practice, this usually happens when the model has small width and the value of ϵ is large.

Theorem 4.3 indicates that the gradients are non-vanishing in adversarial training and more likely to have large magnitude under large values of ϵ. This is validated by Figure 4.4, in which we report the norm of the stochastic gradient $\|\nabla_\theta \widehat{\mathcal{L}}_\epsilon(\theta)\|$ in the last 500 mini-batch updates for MNIST models and the last 2000 mini-batch updates for CIFAR10 models. In vanilla training, i.e., $\epsilon = 0$, the gradient is almost zero in the end, indicating that the optimizer finds a critical point. In this case $\|\nabla_\theta \widehat{\mathcal{L}}_\epsilon(\theta)\|$ is dominated by the variance introduced by stochasticity. However, $\|\nabla_\theta \widehat{\mathcal{L}}_\epsilon(\theta)\|$ increases with ϵ. When ϵ is larger, $\|\nabla_\theta \widehat{\mathcal{L}}_\epsilon(\theta)\|$ is also larger and non-vanishing, indicating that the model is still bouncing around the parameter space at the end of training.



(a) $\|\nabla_\theta \widehat{\mathcal{L}}_\epsilon(\theta)\|$, MNIST

(b) $\|\nabla_\theta \widehat{\mathcal{L}}_\epsilon(\theta)\|$, CIFAR10

Figure 4.4 – Norm of the stochastic gradient $\|\nabla_\theta \widehat{\mathcal{L}}_\epsilon(\theta)\|$ in the final phase of training. We report last 500 updates for MNIST models and last 2000 updates for CIFAR10 models.

Hessian Analysis

To study the effective local Lipschitz constant of $\mathcal{L}_\epsilon(\theta)$, we analyze the Hessian spectrum of models trained under different values of ϵ. It is known that the curvature in the neighborhood of model parameters is dominated by the top eigenvalues of the Hessian matrix $\nabla^2 \mathcal{L}_\epsilon(\theta)$. To this end, we use the power iteration method as in [171] to iteratively estimate the top 20 eigenvalues and the corresponding eigenvectors of the Hessian matrix. Furthermore, to discard the effect of the scale of function $\mathcal{L}_\epsilon(\theta)$ for different ϵ, we estimate the scale of $\mathcal{L}_\epsilon(\theta)$ by randomly sampling θ. We then normalize the top Hessian eigenvalues by the average value of $\mathcal{L}_\epsilon(\theta)$ on these random samples.

In Figure 4.5, we show the top 20 Hessian eigenvalues, both before and after normalization, of LeNet models on MNIST and ResNet18 models on CIFAR10 under different adversarial budgets. In Figure 4.6, we also provide 3D visualization of the neighborhood in directions of the top 2 eigenvectors. It is clear that the local effective second-order Lipschitz constant of the model obtained consistently increases with the value of ϵ.

Ideally, the sharpness of the minima is depicted by the condition number of its Hessian matrix. However, in the context of deep neural networks, the eigenvalue with the smallest

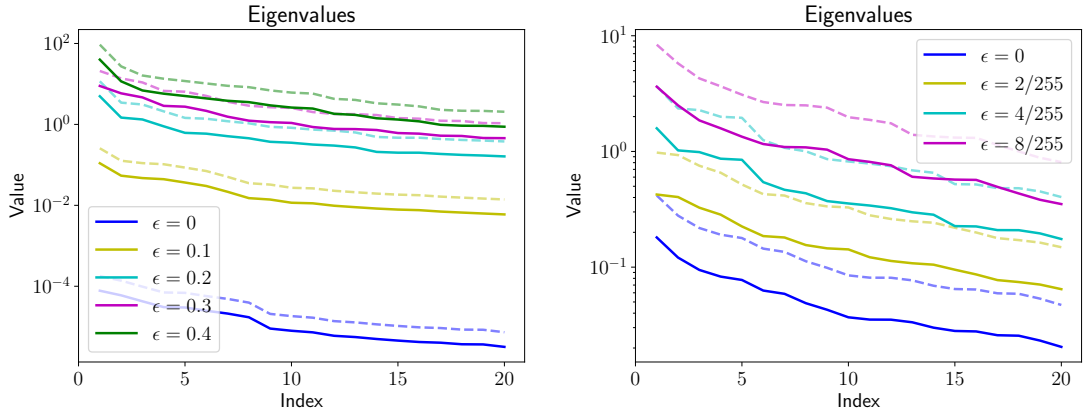
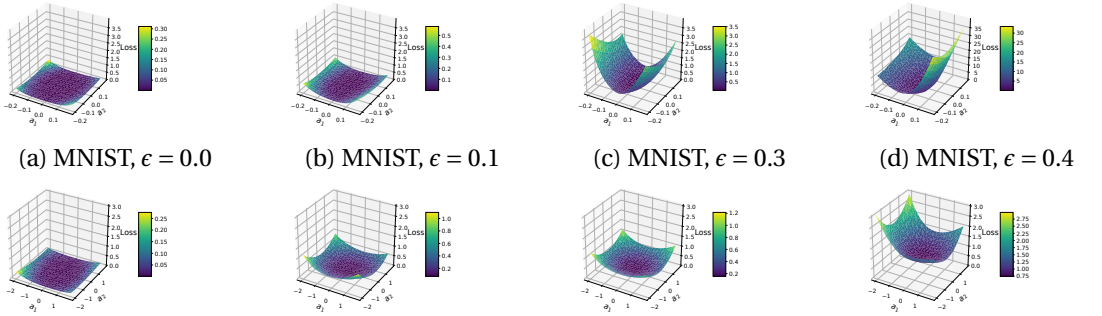


Figure 4.5 – Top 20 eigenvalues of the Hessian matrix for LeNet models on MNIST (left) and ResNet18 models on CIFAR10 (right) under different values of ϵ. We show the normalized (solid) and original (dashed) values.

absolute value of the Hessian matrix is almost zero, which imposes difficulties in calculating the condition number both algorithmically and numerically [50]. Instead, the spectral norm and the nuclear norm of the Hessian matrix are used as a quantitative metric for the sharpness of the minima [37]. To this end, Figure 4.5 demonstrates that the minima obtained is shaper when ϵ is larger.



(a) MNIST, $\epsilon = 0.0$ (b) MNIST, $\epsilon = 0.1$ (c) MNIST, $\epsilon = 0.3$ (d) MNIST, $\epsilon = 0.4$

(e) CIFAR10, $\epsilon = 0$ (f) CIFAR10, $\epsilon = 2/255$ (g) CIFAR10, $\epsilon = 4/255$ (h) CIFAR10, $\epsilon = 8/255$

Figure 4.6 – Loss landscape $\mathcal{L}_\epsilon(\theta + a_1\mathbf{v}_1 + a_2\mathbf{v}_2)$ under different adversarial budgets. θ, $\mathbf{v}_1$, $\mathbf{v}_2$ are the parameter, and the first and second unit eigenvectors of the Hessian matrix. (Note that the z-scale for $\epsilon = 0.4$ in the MNIST case differs from the others.)

To validate the claim in Section 4.2.2 that non-smoothness arises from abrupt changes of the adversarial examples, we study the similarity of adversarial perturbations Δ generated by different model parameter values in a small neighborhood. Specifically, we perturb the model parameters θ in opposite directions to $\theta + a\mathbf{v}$ and $\theta - a\mathbf{v}$, where $\mathbf{v}$ is a unit vector and a is a scalar. Let $\Delta_{a\mathbf{v}}$ and $\Delta_{-a\mathbf{v}}$ represent the adversarial perturbations generated by the corresponding model parameters. We then calculate the average cosine similarity between them over the training set.

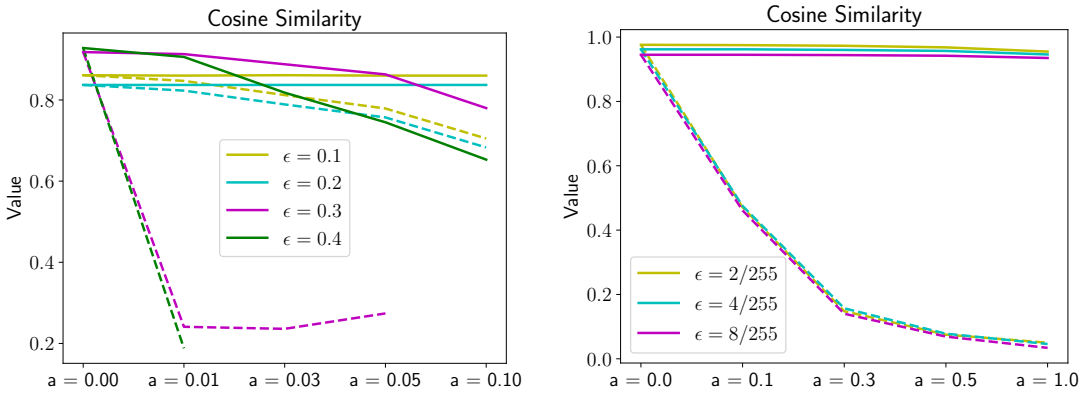


Figure 4.7 – Cosine similarity between perturbations $\Delta_{a\mathbf{v}}$ and $\Delta_{-a\mathbf{v}}$ for LeNet models on MNIST (left) and ResNet18 models on CIFAR10 (right). $\mathbf{v}$ can be either top eigenvectors (dashed) or randomly picked ones (solid). We do not plot the curves when the cosine similarity is negative.

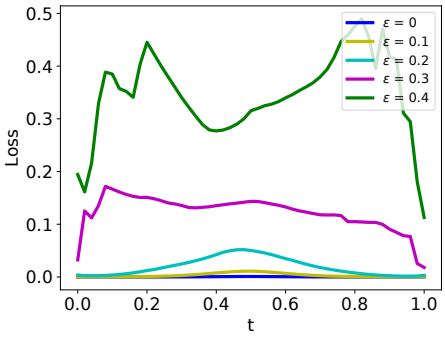
The results of LeNet models on MNIST and ResNet18 models on CIFAR10 are provided in Figure 4.7. To account for the random start in PGD, we run each experiment 4 times and report the average value. The variances of all experiments are smaller than 0.005 and thus not shown in the figure. Note that, when $\mathbf{v}$ is a random unit vector, the robust error $\mathcal{E}_\epsilon(\theta)$ of the parameters $\theta \pm a\mathbf{v}$ on both the training and test sets remains unchanged for different values of a, indicating a flat landscape in the direction $\mathbf{v}$. The adversarial examples in this case are mostly similar and have very high cosine similarity. By contrast, if $\mathbf{v}$ is the top eigenvectors of the Hessian matrix, i.e., the most curvy direction, then we see a sharp increase in the robust error $\mathcal{E}_\epsilon(\theta)$ when we increase a. Correspondingly, the cosine similarity between the adversarial perturbations is much lower, which indicates dramatic changes of the adversarial examples.

Minima Connectivity

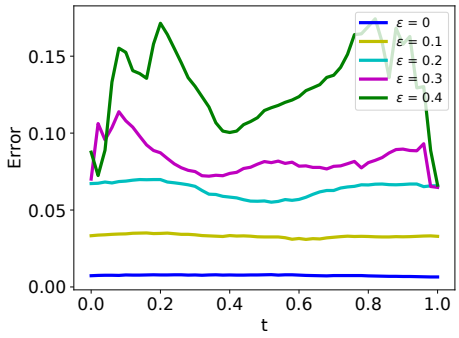
In addition to sharpness of the local minima, we study their connectivity in this section. As pointed in [40, 47], the minima found in vanilla training have been found well connected. That is, if we train two neural networks under the same settings but different initializations, there exists a path connecting the resulting two models in the parameter space such that all points along this path have low loss values.

Similarly to [47], we parameterize the path joining two minima using a *general Bezier curve*. Let θ_0 and θ_n be the parameters of two separately-trained models, and $\{\widehat{\theta}_i\}_{i=1}^{n-1}$ the parameters of $(n-1)$ trainable intermediate models. Then, an n-order Bezier curve is defined as a linear combination of these $(n+1)$ points in parameter space, i.e.,

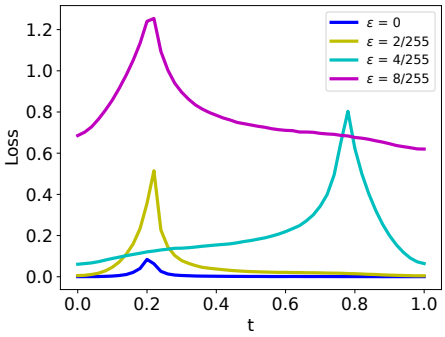
$$\mathcal{B}(t) = (1-t)^n\theta_0 + t^n\theta_n + \sum_{i=1}^{n-1}\binom{n}{t}(1-t)^{n-i}t^i\widehat{\theta}_i\,. \tag{4.28}$$



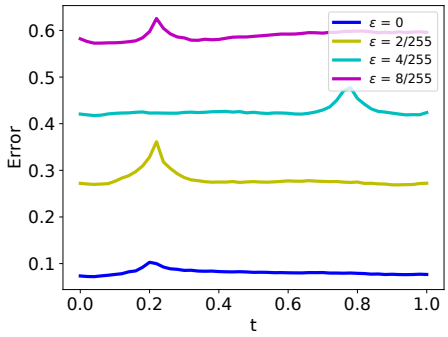
(a) MNIST, LeNet, training loss.



(b) MNIST, LeNet, test error.



(c) CIFAR10, ResNet18, training loss.



(d) CIFAR10, ResNet18, test error.

Figure 4.8 – Training loss and test error along the path connecting the minima of two independently-trained models.

$\mathcal{B}(t)$ is a smooth curve, and $\mathcal{B}(0) = \theta_0$ and $\mathcal{B}(1) = \theta_n$. We train the parameters $\{\widehat{\theta}_i\}_{i=1}^{n-1}$ by minimizing the average loss along the path: $\mathbb{E}_{t\sim U[0,1]}\mathcal{L}_\epsilon(\mathcal{B}(t))$, where $U[0,1]$ is the uniform distribution between 0 and 1. We use the Monte Carlo method to estimate the gradient of this expectation-based function and minimize it using gradient-based optimization. We use second-order Bezier curves to connect MNIST model pairs and fourth-order Bezier curves to connect CIFAR10 model pairs. When evaluating the models on the learned curves, we re-estimate the running mean and variance in the batch normalization layer based on the training set. The results are reported based on the evaluation mode of the models, and we turn off data augmentation to avoid stochasticity.

In Figure 4.8, following [47], we plot the training loss and test error along the learned curve, as a function of t in Equation (4.28). Same as previous experiments, we use LeNet models for MNIST and ResNet18 models for CIFAR10. For vanilla training or when the adversarial budget is small, we can easily find flat curves connecting different minima. This is consistent with the findings in [40, 47]. However, the learned curves are not flat anymore when the strength of the adversarial budget increases. This indicates that the minima are less well-connected under adversarial training, and that it is more difficult for the optimizer to find a minimum.

4.2.4 Periodic Adversarial Scheduling

In previous sections, we have theoretically and empirically shown that the adversarial loss landscape becomes less favorable to optimization under large adversarial budgets. Specifically, gradients are more scattered, local minima are sharper and less connected. In this section, we introduce methods to utilize these properties and overcome the challenges.

Inspired by the learning rate warmup heuristic used in deep learning [54, 70], we introduce warmup for the adversarial budget. Let d be the current epoch index and D be the warmup period's length. We define a cosine scheduler $\epsilon_{\cos}$ and a linear scheduler ϵ_{lin}, parameterized by $\epsilon_{\max}$ and $\epsilon_{\min}$, as:

$$\epsilon_{\cos}(d) = \frac{1}{2}\left(1 - \cos\frac{d}{D}\pi\right)(\epsilon_{\max} - \epsilon_{\min}) + \epsilon_{\min}, \quad \epsilon_{\text{lin}}(d) = (\epsilon_{\max} - \epsilon_{\min})\frac{d}{D} + \epsilon_{\min}\,. \tag{4.29}$$

We clip $\epsilon_{\cos}(d)$ and $\epsilon_{\text{lin}}(d)$ between 0 and ϵ_{target}, the target value of ϵ. If $\epsilon_{\min} \leq 0$ and $\epsilon_{\max} > \epsilon_{\text{target}}$, the value of ϵ starts from 0, gradually increases to ϵ_{target} and remains constant then.

This warmup strategy allows us to overcome the fact, highlighted in the previous sections, that adversarial training is more sensitive to the learning rate under a large budget because the gradients are more scattered. This is evidenced by Figure 4.9, which compares the robust test error of MNIST models relying on different adversarial budget scheduling schemes. For all models, we used $\epsilon = 0.4$, and report results after 100 epochs with different but constant learning rates in Adam [81]. Our linear and cosine schedulers perform better than using a constant value of ϵ during training and yield good performance for a broader range of learning rates: in the small learning rate regime, they speed up training; in the large learning rate regime, they stabilize training and avoid divergence. Note that, as shown in Figure 4.10, warmup of the learning rate does not yield similar benefits.

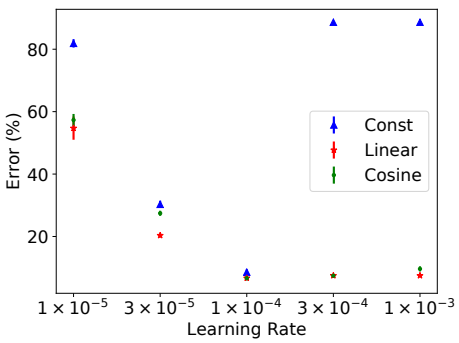


Figure 4.9 – Mean and standard deviation of the test error under different learning rates with Adam and adversarial budget scheduling.

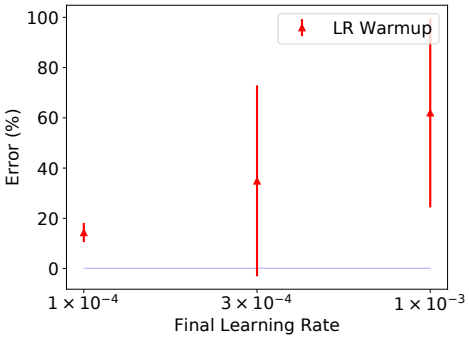
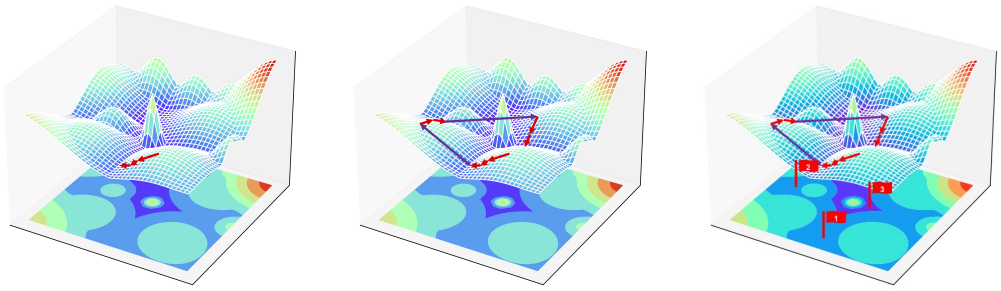


Figure 4.10 – Mean and standard derivation of the test error on MNIST models when we use learning rate warmup but constant ϵ. The best performance by constant learning rate but adversarial budget warmup is highlighted by a blue bar.



(a) Optimization in 1st period. (b) 3 periods during training. (c) Ensemble of 3 checkpoints

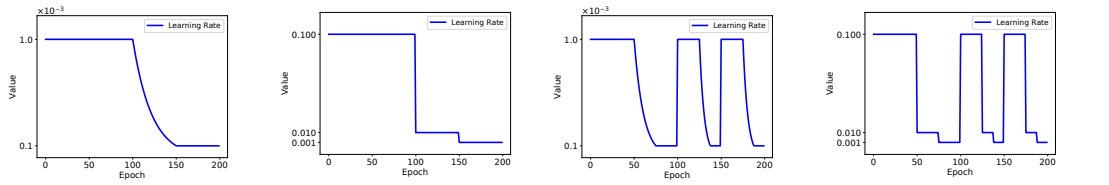
Figure 4.11 – Different phases of periodic adversarial scheduling (PAS). Red arrows means parameter updates within a period; purple arrows means adversarial budget and learning rate resets. The example has 3 phases, and the outcome is the ensemble of 3 checkpoints in the end of each period.

As shown in [70], periodic learning rates enable model ensembling to improve the performance. Especially, in the context of adversarial training, different local minima are not well connectivity, ensembling these minima of potential greater diversity can further boost the performance. Here, we can follow the same strategy but also for the adversarial budget. To this end, we divide the training phase into several periods and store one model at the end of each period. We make final predictions based on the ensemble of these models. This periodic scheme has no computational overhead. We call it periodic adversarial scheduling (PAS).

As before, we run experiments on MNIST and CIFAR10. For MNIST, we use LeNet with width $w = 16$. We train each model for 100 epochs and do not use a periodic scheduling for the learning rate, which we found not to improve the results. For CIFAR10, we use VGG and ResNet18 with width $w = 8$. We train each model for 200 epochs. When there are no learning rate resets, our results indicate the final model after 200 epochs. When using a periodic learning rate, we divide the 200 epochs into 3 periods, i.e., we reset the learning rate and the adversarial budget after 100 and 150 epochs, and compute the results using an ensemble of 3 models.

The value of learning rate and the adversarial budget size are calculated based on the ratio of the current epoch index to the current period length. We fine-tune the weight-decay factor, choosing 1×10^{-3} as the optimal value. In periodic settings, the learning rate and the adversarial budget is reset after 100 and 150 epochs. The scheduling in each period is scaled proportionally. We plot the learning rate scheduling curves for VGG and ResNet18 in Figure 4.12 for both the vanilla and periodic settings.

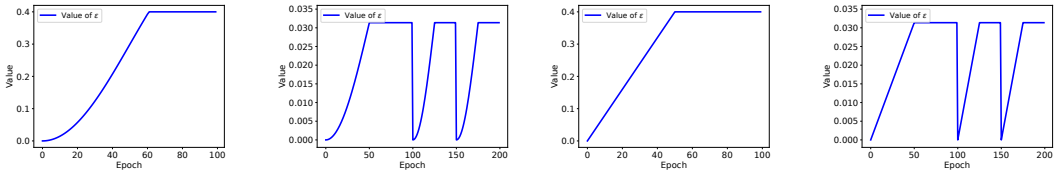
Regarding scheduling of ϵ, we do not fully explore the value range of the hyper-parameters in the cosine and linear schedulers. We use $\epsilon_{\text{min}} = 0.$ for all experiments. For the MNIST experiments, we set $\epsilon_{\text{max}} = 0.6$ for the cosine scheduler and $\epsilon_{\text{max}} = 0.8$ for the linear scheduler. For the CIFAR10 experiments, we set $\epsilon_{\text{max}} = 16/255$ for both the cosine and linear schedulers.



(a) CIFAR10, VGG, Vanilla (b) CIFAR10, ResNet18, Vanilla (c) CIFAR10, VGG, Periodic (d) CIFAR10, ResNet18, Periodic

Figure 4.12 – Learning rate scheduling for VGG and ResNet18 for CIFAR10 classification.

We plot the curves for $\epsilon_{\cos}(d)$ and $\epsilon_{\text{lin}}(d)$ in Figure 4.13.



(a) MNIST, Cosine (b) CIFAR10, Cosine (c) MNIST, Linear (d) CIFAR10, Linear

Figure 4.13 – Adversarial budget scheduling for MNIST and CIFAR10 models.

Task	Periodic Learning Rate	ϵ Scheduler	Clean Error (%)	Robust Error (%) PGD (%)	APGD100 CE (%)	APGD100 DLR (%)	Square5K (%)
MNIST LeNet $\epsilon = 0.4$	No	Constant	1.56(17)	8.58(89)	15.18(155)	14.70(136)	19.58(45)
		Cosine	1.08(2)	6.64(70)	14.36(134)	13.46(129)	16.78(25)
		Linear	1.06(6)	6.69(59)	13.91(150)	13.17(120)	17.05(47)
CIFAR10 VGG $\epsilon = 8/255$	No	Constant	28.25(47)	56.22(43)	58.18(46)	58.65(69)	54.37(29)
		Cosine	25.06(19)	56.06(48)	57.83(45)	58.88(16)	53.95(15)
		Linear	23.56(95)	56.09(14)	57.74(16)	58.39(18)	53.66(24)
	Yes	Constant	28.33(81)	54.24(28)	55.45(26)	56.56(4)	52.85(18)
		Cosine	23.91(21)	53.18(21)	54.44(16)	55.80(24)	51.41(37)
		Linear	21.88(33)	53.03(14)	54.32(17)	55.63(17)	51.28(4)
CIFAR10 ResNet18 $\epsilon = 8/255$	No	Constant	18.62(6)	55.00(8)	57.26(13)	56.60(25)	50.59(19)
		Cosine	18.43(26)	53.95(23)	56.16(18)	55.77(24)	49.60(18)
		Linear	18.55(14)	53.46(20)	55.69(17)	55.45(22)	49.66(28)
	Yes	Constant	21.00(5)	48.98(25)	50.29(27)	50.98(6)	46.84(9)
		Cosine	19.90(18)	48.57(25)	49.71(22)	50.54(9)	46.19(11)
		Linear	20.26(28)	48.60(13)	49.73(9)	50.68(11)	46.47(26)

Table 4.2 – Comparison between different adversarial budget schedulers under different adversarial attacks. *Cosine* / *Linear schedulers* are consistently better than *constant schedulers*. The number between brackets indicate the standard deviation across different runs. Specifically, for example, 1.56(17) stands for 1.56 ± 0.17.

We compare different scheduler in adversarial budget under different tasks and settings. We evaluate the robustness of our trained models by different kinds of attacks. First we evaluate the models under the PGD attack used in training (PGD), i.e., 50-iteration PGD for MNIST

models and 10-iteration PGD for CIFAR10 models. Then, we increase the number of iterations in PGD and compute the robust error under 100-iteration PGD. To solve the issue of suboptimal step size, we also evaluate our models using the state-of-the-art AutoPGD attack [33], which searches for the optimal step sizes. We run AutoPGD for 100 iterations for evaluation, based on either cross-entropy loss (APGD100 CE) or the difference of logit ratio loss (APGD100 DLR). To avoid gradient masking, we also run the state-of-the-art black-box SquareAttack [5] for 5000 iterations (Square5K). The hyperparameters of these attacks are the same as in [33].

The results are summarized in Table 4.2, where we compare the clean and robust accuracy under different adversarial attacks on the test set. It is clear that our proposed cosine or linear schedulers yield better performance, in both clean accuracy and robust accuracy, than using a constant adversarial budget in all cases. For MNIST, warmup not only makes training robust to different choices of learning rate, but also improves the final robust accuracy. For CIFAR10, model ensembling enabled by the periodic scheduler improves the robust accuracy.

4.2.5 Discussion

So far, we focuses on the impact of the strength ϵ of the adversarial budget on the optimization in adversarial training. In this section, we briefly discuss other factors that also affect the adversarial loss landscape.

Model Capacity

The capacity of the model can greatly affects the adversarial loss landscape and thus the performance of adversarial training. Adversarial training needs higher model capacity in two aspects: if we decrease the model capacity, adversarial training will fail to converge while vanilla training still works [98]; if we increase the model capacity, the robust accuracy of adversarial training continues to rise while the clean accuracy of normal training saturates [166].

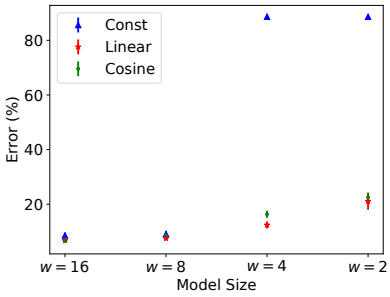


Figure 4.14 – The mean and standard derivation of the test error under LeNet models on MNIST of different width w by different ϵ scheduler.

In Figure 4.14, we report the performance of LeNet models of different width factors w by different ϵ schedulers. We use $\epsilon = 0.4$ at test time. We set the learning rate in Adam to be 10^{-4}, because, for constant ϵ during training, it yields the best performance. Both Cosine and Linear schedulers outperform using a constant ϵ in all cases. When the model size is small, e.g., $w = 4$ and $w = 2$, using a constant ϵ during training fails to converge, but the Cosine and Linear schedulers still yield competitive results.

As a result, warmup in adversarial budget is also necessary for small models. In many cases, the parameter space of small models has good minima in terms of robustness, but adversarial training with a constant value of ϵ fails to find them. For example, one can obtain small but robust models by pruning large ones [58, 172].

Architecture

The network architecture encodes the parameterization of the model, so it greatly affects the adversarial loss landscape. In Table 4.2, ResNet18 has fewer trainable parameters but better performance than VGG on CIFAR10, indicating that ResNet18 has a better parameterization in terms of robustness. Since the optimal architecture for adversarial robustness is not necessarily the same as the one for clean accuracy, we believe that finding architectures inherently favorable to adversarial training is an interesting but challenging topic for future research.

Adversarial Example Generation

We approximate the adversarial loss using adversarial examples generated by PGD, which is a good estimate of the inner maximization in (1.2). PGD-based adversarial training updates model parameters by near-optimal adversarial examples. However, recent works [130, 158] have shown that robust models can also be trained by suboptimal adversarial examples, which are faster to obtain. The formulation of these methods differs from (1.2), because the inner maximization problem is not approximately solved. Understanding why models (partially) trained on suboptimal adversarial examples are resistant to stronger adversarial examples needs more investigation.

4.3 Adversarial Overfitting

Last section focuses on the slower convergence of adversarial training, this section studies another phenomenon: adversarial overfitting. Figure 4.1 in Section 4.1 clearly demonstrates that adversarial training has much larger generalization gap, particularly in the late phase of training, when the model's performance on the test set decays significantly. So far, researchers found adversarial overfitting can be mitigated by early stopping [118] or model smoothing [21], the reason behind the overfitting of adversarial training remains poorly understood.

We study the adversarial overfitting phenomenon from the aspect of training instances, i.e., training input-target pairs. We introduce a quantitative metric to measure the relatively difficulty of an instance within a set. Then, we analyze the model's behavior on training instances of different difficulties. This lets us discover that the model's generalization performance decays significantly when it fits the hard adversarial instances in the later training phase.

To more rigorously study this phenomenon, we conduct theoretical analyses on both linear and nonlinear models. For linear models, we study logistic regression on a Gaussian mixture

model, in which we can calculate the analytical expression of the model parameters upon convergence and thus the robust test accuracy. Our theorem demonstrates that adversarial training on harder instances leads to larger generalization gaps. Furthermore, the difference in robust accuracy between the models trained by the hard instances and the ones trained by the easy instances increases with the size of the adversarial budget. In the case of nonlinear models, we derive the lower bound of the model's Lipschitz constant when the model is well fit to the training instances under adversarial attacks. This bound increases with the difficulty level of the training instances and the size of the adversarial budget. Since a larger Lipschitz constant indicates a higher adversarial vulnerability [120, 154, 155], our theoretical analysis confirms our empirical observations.

Our empirical and theoretical analyses indicate that avoiding to fit the hard training instances can mitigate adversarial overfitting. We therefore study this in three different scenarios: standard adversarial training, fast adversarial training and adversarial fine-tuning with additional training data. We show that existing approaches to mitigating adversarial overfitting [8, 21, 71] implicitly avoid fitting the hard adversarial input-target pairs, by either adaptive inputs or adaptive targets. By contrast, the methods that focus on fitting hard adversarial [181] instances are not truly robust under adaptive attacks [67].

4.3.1 Measuring Instancewise Difficulty

Parametric models, such as f parameterized by θ, are trained to minimize a loss objective based on several input-target pairs called training set, and are then evaluated on a held-out set called test set. By comparing the loss value of each instance, we can understand which ones, in either the training or the test set, are more difficult for the model to fit. Therefore, we introduce a metric for instance difficulty based on its loss. Let $\widehat{g_\epsilon}(\theta, \mathbf{x})$ denote the average loss of the instance $(\mathbf{x}, y)$ under the adversarial budget $\mathcal{S}_\epsilon^{(p)}$ across all training epochs. Same as in Equation 4.1, the label y and superscript (p) are discarded for notation simplicity.

Now, we define the difficulty of the instance $(\mathbf{x}, y)$ within a finite set $\mathcal{D}$ as:

$$d_\epsilon(\mathbf{x}) = \mathbb{P}(\widehat{g_\epsilon}(\mathbf{x}) < \widehat{g_\epsilon}(\widehat{\mathbf{x}}) | \widehat{\mathbf{x}} \sim U(\mathcal{D})) + \frac{1}{2}\mathbb{P}(\widehat{g_\epsilon}(\mathbf{x}) = \widehat{g_\epsilon}(\widehat{\mathbf{x}}) | \widehat{\mathbf{x}} \sim U(\mathcal{D})) \,, \tag{4.30}$$

Here, $\widehat{\mathbf{x}} \sim U(\mathcal{D})$ indicates that $\widehat{\mathbf{x}}$ is uniformly sampled from the finite set $\mathcal{D}$. By definition, $d_\epsilon(\mathbf{x})$ is a bounded function, close to 0 for the hardest instances and to 1 for the easiest ones.

To study the factors affecting the difficulty function defined in (4.30), let us denote by $d_\epsilon^{(1)}$, $d_\epsilon^{(2)}$ the difficulty functions obtained under two different training settings, such as different network architectures or adversarial attack strategies. We then define the difficulty distance (*D-distance*) between two such functions in the following equation. In this regard, the expected D-distance between two random difficulty functions is 0.375.

$$D\left(d_\epsilon^{(1)}, d_\epsilon^{(2)}\right) = \mathbb{E}_{\mathbf{x} \sim U(\mathcal{D})} \left| d_\epsilon^{(1)}(\mathbf{x}) - d_\epsilon^{(2)}(\mathbf{x}) \right| . \tag{4.31}$$

We then study the properties of the difficulty metric of Equation (4.30) by performing experiments on the CIFAR10 [83] and CIFAR10-C [64] dataset, varying factors of interest. In particular, we first study the influence of the network by using either a ResNet18 model [158], trained for either 100 or 200 epochs (RN18-100 or RN18-200), or a WideResNet34 model [98] trained for 200 epochs (WRN34). To generate adversarial attacks, we make use of PGD with an adversarial budget based on the l_∞ norm with $\epsilon = 8/255$. PGD uses a step size of 2/255 and runs for 10 iterations. We use stochastic gradient descent (SGD) with a mini-batch size of 128 and a momentum to optimize the model parameters, we also use weight decay whose factor is 0.0005. The momentum factor is 0.9, the learning rate starts with 0.1 and is divided by 10 in the 1/2 and 3/4 of the whole training duration. This corresponds to the settings used in other works [64, 98].

In the left part of Table 4.3, we report the D-distance for all pairs of settings. Each result is averaged over 4 runs, and the variances are all below 0.012. The D-distances in all scenarios are all very small and close to 0, indicating the architecture and the training duration have little influence on instance difficulty based on our definition.

$d_\epsilon^{(1)} \backslash d_\epsilon^{(2)}$	RN18-100	RN18-200	WRN34
RN18-100	0.0189	0.0232	0.0355
RN18-200	0.0232	0.0159	0.0299
WRN34	0.0355	0.0299	0.0178

$d_\epsilon^{(1)} \backslash d_\epsilon^{(2)}$	Clean	FGSM	PGD
Clean	0.0189	0.0607	0.1713
FGSM	0.0607	0.0843	0.1677
PGD	0.1713	0.1677	0.0857

Table 4.3 – D-distances between the difficulty functions in different settings, including different model architectures and training duration (left), and different types of perturbations (right).

We then perform experiments by varying the attack strategies using a ResNet18 network. As shown by the D-distances reported in the right portion of Table 4.3, the discrepancy between values obtained with clean, FGSM-perturbed and PGD-perturbed inputs is much larger, thus indicating that the function d_ϵ correctly reflects the influence of an attack on an instance.

In addition, Table 4.4 demonstrates the D-distance between the difficulty functions based on clean instances, FGSM-perturbed instance, PGD-perturbed instances and different common corruptions from CIFAR10-C [64]. Note that [64] only provides corrupted instances on the test set, so we train models on the clean training set and test model on corrupted test set in these cases. We use ResNet18 architecture and train it for 100 epochs in all cases, results are reported on the test set. Compared with the results in the left half of Table 4.3, the D-distance is much larger here. This indicates the difficulty function depends on the perturbation type applied to the input, including the common corruptions.

To summarize, the results in Table 4.3 and 4.4 indicate that our difficulty metric mainly depends on the data itself and on the perturbation type, i.e., attack methods; not the model architecture or the training duration.

In the definition of our difficulty metric in Equation (4.30), the difficulty of one instance is based on its average loss values during the training procedure. It is intuitive, because the

$d_1 \backslash d_2$	brightness	contrast	defocus	elastic	fog	gaussian_blur
Clean	0.1279	0.3219	0.2646	0.2115	0.2324	0.3069
FGSM	0.1303	0.3128	0.2642	0.2098	0.2289	0.3064
PGD	0.1873	0.3082	0.2616	0.2319	0.2414	0.2959

$d_1 \backslash d_2$	glass_blur	jpeg	motion_blur	pixelate	gaussian_noise	impulse_noise
Clean	0.2809	0.1838	0.2520	0.2365	0.2999	0.2869
FGSM	0.2760	0.1853	0.2520	0.2417	0.2918	0.2807
PGD	0.2825	0.2026	0.2605	0.2551	0.2980	0.2866

$d_1 \backslash d_2$	saturate	shot_noise	snow	spatter	zoom_blur	speckle_noise
Clean	0.1335	0.2832	0.2033	0.1930	0.2654	0.2829
FGSM	0.1329	0.2754	0.2003	0.1946	0.2657	0.2759
PGD	0.1932	0.2841	0.2148	0.2297	0.2711	0.2901

Table 4.4 – D-distances between difficulty functions of vanilla / FGSM / PGD training and training based on 18 different corruptions on CIFAR10-C. We run each experiment for 4 times and report the average value.

values of the loss objective represents the cost that model needs to fit the corresponding data point. The bigger this cost is, the more difficulty this instance will be. To make the metric stable and prevent the metric from being sensitive to the stochasticity in the training dynamics, we use the average value of the loss objective for each instance to define its difficulty. In addition to the average loss objectives, we can also use the average 0-1 error to define the difficulty function. In Figure 4.15, we plot the relationship between the difficulty metric based on the average loss values and the one based on the average 0-1 error for instances in the CIFAR10 training set when we train a ResNet18 model for 100 epochs and WideResNet34 model for 200 epochs. We can see a strong correlation between them for both models. The correlation of the difficulty measured by two metrics for the same instance is 0.9466 in the ResNet18 case and 0.9545 in the WideResNet34 case. The high correlation indicates we can use either metric to measure the difficulty. Since the loss objective values are continuous and finer-grained, we choose it as the basis of the difficulty function we use in this paper.

4.3.2 Empirical Observation

Based on the discussion in the last section, we conclude that the function d_ϵ defined in (4.30) mainly depends on the data and the perturbation applied; the model architecture and the training duration have negligible effects on it. We can then use $d_\epsilon(\mathbf{x})$ to represent the difficulty of $\mathbf{x}$ within a set under a specific type of perturbation.

In Figure 4.16, we show some samples of the easiest and the hardest instances of each class in CIFAR10 [83] and SVHN [104], respectively. In both cases, the easiest instances are visually highly similar, whereas the hardest ones are much more diverse, some of them being ambiguous or even incorrectly labeled. Now, we study how easy and hard instances impact the

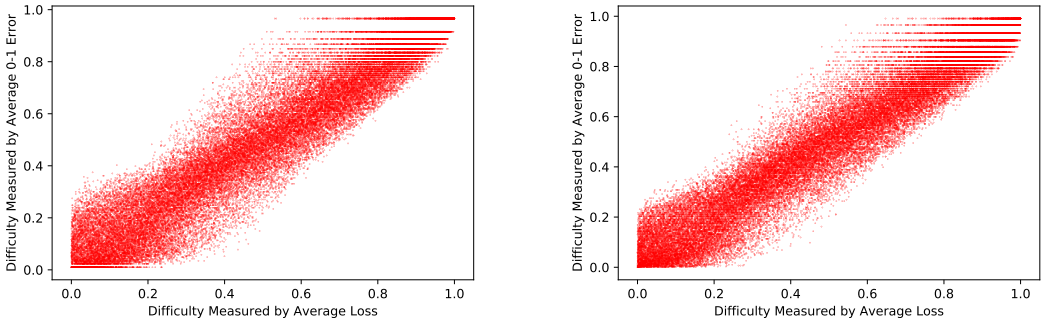
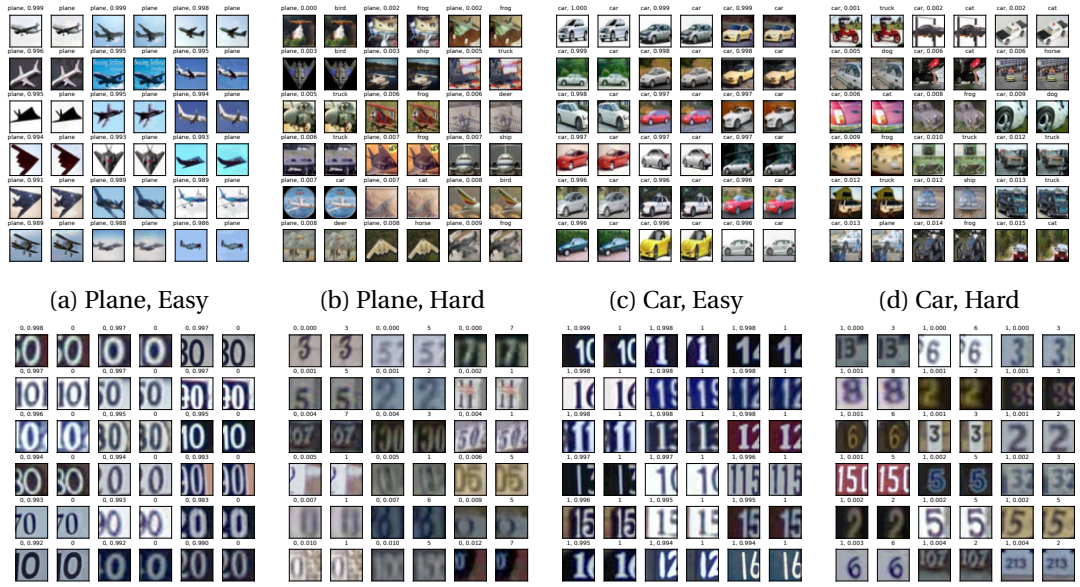


Figure 4.15 – The relationship between the difficulty function based on the average loss values and the one based on the average 0-1 errors. The left figure is based on the RN18-200 model; the right figure is based on the WRN34 model. The correlation between these two metrics are 0.9466 (left) and 0.9545 (right), respectively.

performance of adversarial training, with a focus on the adversarial overfitting phenomenon.



(a) Plane, Easy (b) Plane, Hard (c) Car, Easy (d) Car, Hard

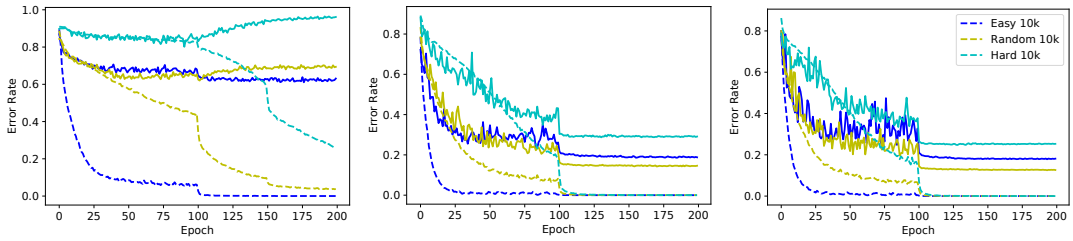
(e) 0, Easy (f) 0, Hard (g) 1, Easy (h) 1, Hard

Figure 4.16 – Easy and hard examples in the first two categories of both CIFAR10 (first row) and SVHN (second row) dataset. In each subfigure, odd columns present the original images, and even columns present the PGD-perturbed images. Above each image, we provide the normalized difficulty defined in Equation (4.30) as well as the labels: true labels for the original images and the predicted labels for the perturbed images.

Using a Subset of Training Data

We start by training ResNet18 models for 200 epochs using either the 10000 easiest, random or hardest instances of the CIFAR10 training set via either vanilla training, FGSM or PGD adversarial training. The adversarial budget is based on the l_∞ norm and $\epsilon = 8/255$. Note that the instance's difficulty is defined under the corresponding perturbation type, and we enforce these subsets to be class-balanced. For example, the easiest 10000 instances consist of the easiest 1000 instances in each class.

We provide the learning curves under different settings in Figure 4.17.



(a) PGD Adversarial Training. (b) FGSM Adversarial Training. (c) Vanilla Training.

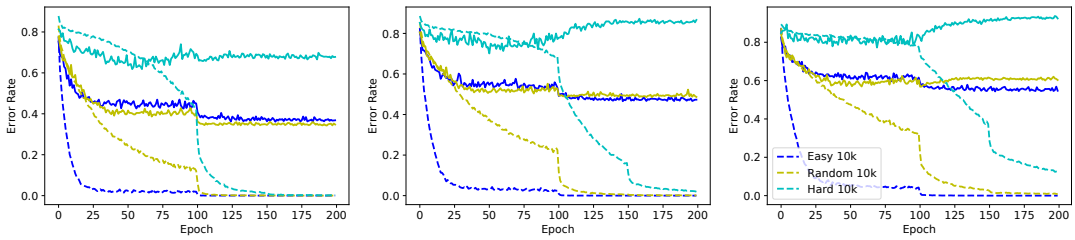
Figure 4.17 – Learning curves obtained by training on the 10000 easiest, random and hardest instances of CIFAR10 under different scenarios. The training error (dashed lines) is the error on the selected instances, and the test error (solid lines) is the error on the whole test set.

For PGD adversarial training, in Figure 4.17a, while we observe adversarial overfitting when using the random instances, as in [118], no such phenomenon occurs when using the easiest instances: the performance on the test set does not degrade during training. However, PGD adversarial training fails and suffers more severe overfitting when using the hardest instances.

In contrast to PGD, FGSM adversarial training and vanilla training (Figure 4.17b, 4.17c), through which the model does not achieve true robustness [98], do not suffer from severe adversarial overfitting. In these cases, the models trained with the hardest instances also achieve non-trivial test accuracy. Furthermore, the gaps in robust test accuracy between the models trained by easy instances and by hard ones are much smaller.

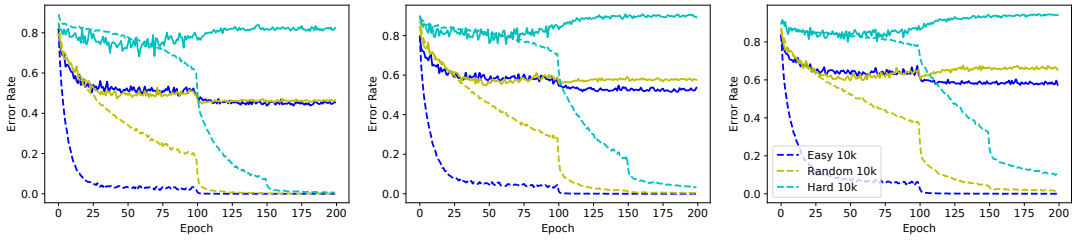
In Figure 4.18, we demonstrate the learning curves of PGD adversarial training using l_∞ norm based adversarial budget with different values of ϵ. With the increase in the size of the adversarial budget, we can see a clear transition from the vanilla training: more and more severe generalization decay when training on the random or the hardest subset. In Figure 4.19 confirms the same phenomenon in the case of l_2 norm based adversarial budget.

Finally, we experiment with training models using increasingly many training instances, start with the easiest ones. We demonstrate our results in Figure 4.20 for both CIFAR10 and SVHN, under the l_∞ norm based adversarial budget with $\epsilon = 8/255$ and $\epsilon = 0.02$, respectively. Our results indicate that if we do model selection on a validation set as in [118], the selected models are still better on both CIFAR10 and SVHN when they are trained with more data, although the



(a) $\epsilon = 2/255$. (b) $\epsilon = 4/255$. (c) $\epsilon = 6/255$.

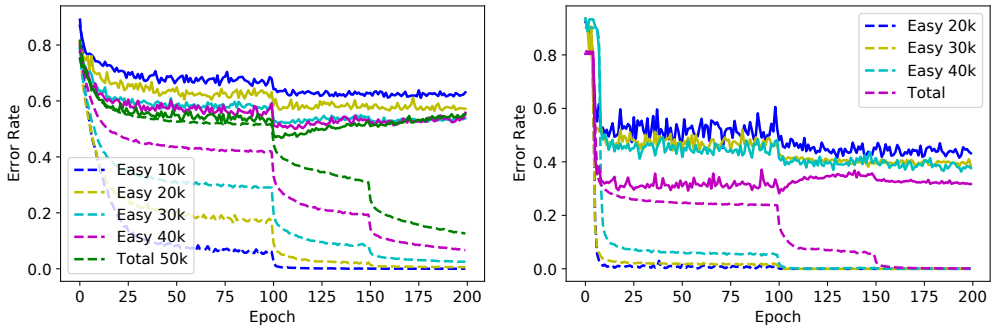
Figure 4.18 – Learning curves obtained by training on the 10000 easiest, random and hardest instances of CIFAR10 by PGD adversarial training with l_∞ adversarial budget using different values of ϵ. The training error (dashed lines) is the error on the selected instances, and the test error (solid lines) is the error on the whole test set.



(a) $\epsilon = 0.50$. (b) $\epsilon = 0.75$. (c) $\epsilon = 1.00$.

Figure 4.19 – Learning curves obtained by training on the 10000 easiest, random and hardest instances of CIFAR10 by PGD adversarial training with l_2 adversarial budget using different values of ϵ. The training error (dashed lines) is the error on the selected instances, and the test error (solid lines) is the error on the whole test set.

final models in these cases are not necessarily better. This indicates that the hard instances can still benefit adversarial training, but need to be utilized in an adaptive manner.



(a) CIFAR10 (b) SVHN

Figure 4.20 – The learning curves of RN18 models on CIFAR10 and SVHN, with more and more training data. The training accuracy and test accuracy are ploted as dashed lines and solid lines, respectively. The training accuracy is based on the selected subset of the training data, while the test accuracy is based on the whole test set.

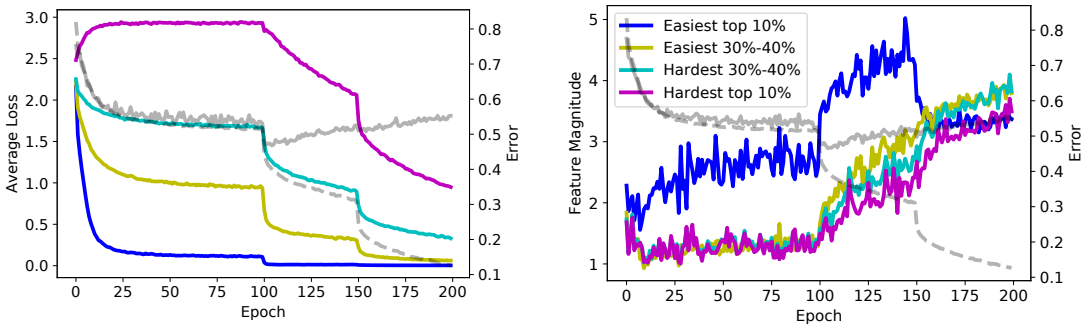
Hard Instances Lead to Overfitting

Let us now turn to the more standard setting where we train the model with the entire training set. To nonetheless analyze the influence of instance difficulty in this scenario, we divide the training set $\mathcal{D}$ into 10 non-overlapping groups $\{\mathcal{G}_i\}_{i=0}^{9}$ with:

$$\mathcal{G}_i = \left\{\mathbf{x} \in \mathcal{D} | 0.1 \times i \leq d_\epsilon(\mathbf{x}) < 0.1 \times (i+1)\right\} . \tag{4.32}$$

That is, $\mathcal{G}_0$ is the hardest group, whereas $\mathcal{G}_9$ is the easiest one. We then train a ResNet18 model on the entire CIFAR10 training set by PGD adversarial training and monitor the training behavior of the different groups. In particular, in Figure 4.21a, we plot the average loss of the instances in the groups $\mathcal{G}_0$, $\mathcal{G}_3$, $\mathcal{G}_6$ and $\mathcal{G}_9$. The results show that, in the early training stages, the model first fits the easy instances, as evidenced by the average loss of group $\mathcal{G}_9$ decreasing much faster than that of the other groups. By contrast, in the late training phase, the model tries to fit the more difficult instances, with the average loss of groups $\mathcal{G}_0$ and $\mathcal{G}_3$ decreasing much faster than that of the other groups. In this period, however, the robust test error (solid grey line) increases, which indicates that adversarial overfitting arises from the model's attempt to fit the hard adversarial instances.

In addition to average losses, inspired by [76], which showed that the penultimate layer's activations of a robust model correspond to its *robust features* that cannot be misaligned by adversarial attacks, we monitor the group-wise average magnitudes of the penultimate layer's activations. As shown in Figure 4.21b, the model first focuses on extracting robust features for the easy instances, as evidenced by the comparatively large activations of the instances in $\mathcal{G}_9$. In the late phase of training, the norm of the activations of the hard instances increases significantly, bridging the gap between easy and hard instances. This further indicates that the model focuses more on the hard instances in the later phase, at which point it starts overfitting.



(a) (b)

Figure 4.21 – Analysis on the groups $\mathcal{G}_0$, $\mathcal{G}_3$, $\mathcal{G}_6$ and $\mathcal{G}_9$ in the training set. The right vertical axis corresponds to the training (dashed grey line) and test (solid grey line) error under adversarial attacks for both plots. **Left plot:** The left vertical axis represents the average loss of different groups. **Right plot:** The left vertical axis represents the average l_2 norm of features extracted during training for different groups.

4.3.3 Toy Model: Logistic Regression

We now study the relationship between adversarial overfitting and instance difficulty from a theoretical viewpoint. This section focuses on a toy model: logistic regression. We use $\{\mathbf{x}_i, y_i\}_{i=1}^N$ to represent the training data, and $(\mathbf{X}, \mathbf{y})$ as its matrix form. For notation simplicity, $\{\mathbf{x}'_i, y_i\}_{i=1}^n$ and $(\mathbf{X}', \mathbf{y})$ are their adversarial counterparts. That is to say $\mathbf{x}'_i = \mathbf{x}_i + \Delta_i$. Here, $\boldsymbol{x}_i \in \mathbb{R}^M$, $y_i \in \{-1, +1\}$, $\mathbf{X} \in \mathbb{R}^{N \times M}$ and $\mathbf{y} \in \{-1, +1\}^n$.

We study the logistic regression model under an l_2 norm based adversarial budget. In this case, the model is parameterized by $\mathbf{w} \in \mathbb{R}^M$ and outputs $sign(\mathbf{w}^T \mathbf{x}'_i)$ given the adversarial example $\mathbf{x}'_i$ of the input $\mathbf{x}_i$. Therefore, the loss function for this instance is $\frac{1}{1+e^{y_i \mathbf{w}^T \mathbf{x}'_i}}$. We assume over-parameterization, which means $N < M$.

The following theorem shows that, under mild assumptions, the parameters of the adversarially trained model converge to the l_2 max-margin direction of the training data.

Theorem 4.4 *For a dataset $\{\mathbf{x}_i, y_i\}_{i=1}^N$ that is linearly separable under the adversarial budget $\mathcal{S}^{(2)}(\epsilon)$, any initial point $\mathbf{w}_0$ and step size $\alpha \leq 2\|\mathbf{X}\|^{-2}$, the gradient descent $\mathbf{w}_{u+1} = \mathbf{w}_u - \alpha \nabla_{\mathbf{w}} \mathcal{L}_{\mathbf{w}_u}(\mathbf{X}')$ converges asymptotically to the l_2 max-margin vector of the training data. That is,*

$$\lim_{u \to \infty} \frac{\mathbf{w}_u}{\|\mathbf{w}_u\|} = \frac{\widehat{\mathbf{w}}}{\|\widehat{\mathbf{w}}\|}, \text{ where } \widehat{\mathbf{w}} = \operatorname{argmin}_{\mathbf{w}} \|\mathbf{w}\| \quad s.t. \ \forall i \in \{1, 2, ..., n\}, \ \mathbf{w}^T \mathbf{x}_i \geq 1 \,. \tag{4.33}$$

Theorem 4.4 extends the conclusion in [141], which only studies the non-adversarial case. It also indicates that the optimal parameters are only determined by the support vectors of the training data, which are the ones with the smallest margin. According to the loss function, the smallest margin means the largest loss values and thus the hardest training instances based on our definition in Section 4.3.1.

Similar to [141], we can assume all instances are positive without the loss of generality, this is because we can always redefine $y_i \mathbf{x}_i$ as the input. In this regard, the loss to optimize in a logistic regression model under the adversarial budget $\mathcal{S}^{(2)}(\epsilon)$ is:

$$g_\epsilon(\mathbf{w}, \mathbf{X}) = \sum_{i=1}^N l(\mathbf{w}^T \mathbf{x}_i - \epsilon \|\mathbf{w}\|) \tag{4.34}$$

Here $l(\cdot)$ is the logistic function: $l(x) = \frac{1}{1+e^{-x}}$. The loss function $g_\epsilon(\mathbf{w}, \mathbf{X})$ is $\|\mathbf{X}\|^2$-smooth, where $\|\mathbf{X}\|^2$ is the maximal singular value of $\mathbf{X}$. Since function g_ϵ is convex on $\mathbf{w}$, so gradient descent of step size smaller than $2\|\mathbf{X}\|^{-2}$ will asymptotically converge to the global infimum of the function g_ϵ on $\mathbf{w}$.

Before proving Theorem 4.4, we first introduce the following lemma:

Lemma 4.2 *Consider the max-margin vector $\widehat{\mathbf{w}}$ of the vanilla case defined in Equation (4.33), we then introduce the max margin vector $\widehat{\mathbf{w}'}$ defined under the adversarial attack of budget $\mathcal{S}^{(2)}(\epsilon)$ as follows:*

$$\widehat{\mathbf{w}'} = \text{argmin}_{\mathbf{w}} \|\mathbf{w}\| \;\; s.t. \; \forall i \in \{1,2,...,N\}, \; \mathbf{w}^T\mathbf{x}_i - \epsilon\|\mathbf{w}\| \geq 1 \tag{4.35}$$

Then we have $\widehat{\mathbf{w}'}$ is collinear with $\widehat{\mathbf{w}}$, i.e., $\frac{\widehat{\mathbf{w}'}}{\|\widehat{\mathbf{w}'}\|} = \frac{\widehat{\mathbf{w}}}{\|\widehat{\mathbf{w}}\|}$.

Proof: We show that $\widehat{\mathbf{w}} = \frac{1}{1+\epsilon\|\widehat{\mathbf{w}'}\|}\widehat{\mathbf{w}'}$ and prove it by contraction.

Let's assume $\exists \mathbf{v}, \; s.t. \; \|\mathbf{v}\| < \frac{\|\widehat{\mathbf{w}'}\|}{1+\epsilon\|\widehat{\mathbf{w}'}\|}$ and $\forall i \in \{1,2,...,N\}, \; \mathbf{v}^T\mathbf{x}_i \geq 1$, then we can consider $\mathbf{v}' = (1+\|\widehat{\mathbf{w}'}\|)\mathbf{v}$. The l_2 norm of $\mathbf{v}'$ is smaller than that of $\widehat{\mathbf{w}'}$, and we have

$$\forall i \in \{1,2,...,N\}, \mathbf{v}'^T\mathbf{x}_i - \epsilon\|\mathbf{v}'\| = (1+\epsilon\|\widehat{\mathbf{w}'}\|)\mathbf{v}^T\mathbf{x}_i - \epsilon\|\mathbf{v}'\| > (1+\epsilon\|\widehat{\mathbf{w}'}\|) - \epsilon\|\widehat{\mathbf{w}'}\| = 1 \tag{4.36}$$

Inequality (4.36) shows we can construct a vector $\mathbf{v}'$ whose l_2 norm is smaller than $\widehat{\mathbf{w}'}$ and satisfying the condition (4.35), this contracts with the optimality of $\widehat{\mathbf{w}'}$. Therefore, there is no solution of condition (4.33) whose norm is smaller than $\frac{\|\widehat{\mathbf{w}'}\|}{1+\epsilon\|\widehat{\mathbf{w}'}\|}$.

On the other hand, $\frac{1}{1+\epsilon\|\widehat{\mathbf{w}'}\|}\widehat{\mathbf{w}'}$ satisfies the condition (4.33) and its l_2 norm is $\frac{\|\widehat{\mathbf{w}'}\|}{1+\epsilon\|\widehat{\mathbf{w}'}\|}$. As a result, we have $\widehat{\mathbf{w}} = \frac{1}{1+\epsilon\|\widehat{\mathbf{w}'}\|}\widehat{\mathbf{w}'}$. That means $\widehat{\mathbf{w}}$ and $\widehat{\mathbf{w}'}$ are collinear. □

With Lemma 4.2, Theorem 4.4 is more straightforward, whose proof is shown below. Regarding the convergence analysis of the logistic regression model in non-adversarial cases, we encourage the readers to find more details in [77, 141].

Proof: Theorem 1 in [77] and Theorem 3 in [141] proves the convergence of the direction of the logistic regression parameters in different cases. In this regard, we can let $\mathbf{w}_\infty = \lim_{u\to\infty}\frac{\mathbf{w}(u)}{\|\mathbf{w}(u)\|}$. That is to say, for sufficiently large u, the direction of the parameter $\mathbf{w}(u)$ can be considered fixed. As a result, the adversarial perturbations of each data instance $\mathbf{x}_i$ is fixed, i.e., $\epsilon\mathbf{w}_\infty$.

We can then apply the conclusion of Theorem 3 in [141] here, the only difference is the data points are $\{\mathbf{x}_i - \epsilon\mathbf{w}_\infty\}_{i=1}^n$. Therefore, the parameter $\mathbf{w}(u)$ will converge to the l_2 max margin of the dataset $\{\mathbf{x}_i - \epsilon\mathbf{w}_\infty\}_{i=1}^n$. When $t\to\infty$, we have $\mathbf{w}(u)^T(\mathbf{x}_i - \epsilon\mathbf{w}_\infty) = \mathbf{w}(u)^T\boldsymbol{x}_i - \epsilon\|\mathbf{w}(u)\|$. This is exactly the adversarial max margin condition in (4.35). Based on Lemma 4.2, we have $\lim_{u\to\infty}\frac{\mathbf{w}(u)}{\|\mathbf{w}(u)\|} = \frac{\widehat{\mathbf{w}'}}{\|\widehat{\mathbf{w}'}\|} = \frac{\widehat{\mathbf{w}}}{\|\widehat{\mathbf{w}}\|}$ □

To further study how the training instances' difficulty influences the model's generalization performance, we assume that the data points are drawn from a I-mode Gaussian mixture model (GMM). Specifically, the i-th component has a probability p_i of being sampled and is

formulated as:

$$\mathbf{x}_i \sim \mathcal{N}(y_i r_k \boldsymbol{\eta}, \mathbf{I}) \tag{4.37}$$

Here, $\boldsymbol{\eta} \in \mathbb{R}^M$ is the unit vector indicating the mean of the positive instances, and $r_i \in \mathbb{R}^+$ controls the average distance between the positive and negative instances. The mean values of all modes in this GMM are colinear, so r_i indicates the difficulty of instances sampled from the i-th component. Without the loss of generality, we assume $r_1 < r_2 < ... < r_{I-1} < r_I$. As in Section 4.3.2, we consider models trained with the subsets of the training data, e.g., N instances from the l-th component. $l = 1$ then indicates training on the hardest examples, while $l = I$ means using the easiest. In matrix form, we have $\mathbf{X} = r_l \mathbf{y}\boldsymbol{\eta}^T + \mathbf{Q}$ for the instances sampled from the l-th component, where the rows of noise matrix $\mathbf{Q}$ are sampled from $\mathcal{N}(\mathbf{0}, \mathbf{I})$.

Although the max-margin direction in Equation (4.33), where the parameters converge based on Theorem 4.4, does not have an analytical expression, the results in [149] indicate that, in the over-parameterization regime and when the training data is sampled from a GMM, the max-margin direction is the min-norm interpolation of the data with high probability. Since the latter has an analytical form given by $\mathbf{X}^T(\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{y}$, we can then calculate the exact generalization performance of the trained model as stated in the following theorem.

Theorem 4.5 *If a logistic regression model is adversarially trained on N separable training instances sampled from the l-th component of the GMM model described in (4.37). If $\frac{M}{N\log N}$ is sufficiently large*[3]*, then with probability $1 - \mathcal{O}\left(\frac{1}{N}\right)$, the expected adversarial test error $\mathcal{R}$ under the adversarial budget $\mathcal{S}_\epsilon^{(2)}$, which is a function of r_l and ϵ, on the whole GMM model described in (4.37) is given by:*

$$\begin{aligned} \mathcal{R}(r_l, \epsilon) &= \sum_{i=1}^{I} p_i \Psi\left(r_i g(r_l) - \epsilon\right) \\ \text{where } g(r_l) &= \left(C_1 - \frac{1}{C_2 r_l^2 + o(r_l^2)}\right)^{\frac{1}{2}},\ C_1, C_2 \geq 0. \end{aligned} \tag{4.38}$$

C_1, C_2 are independent of ϵ and r_l. The function Ψ is defined as $\Psi(x) = \mathbb{P}(Z > x),\ Z \sim \mathcal{N}(0,1)$.

To prove the theorem above, we first calculate the robust error for the i-th component of the GMM model defined in (4.37).

Lemma 4.3 *The 0-1 classification error of a linear classifier $\mathbf{w}$ under the adversarial attack of the budget $\mathcal{S}_\epsilon^{(2)}$ for the i-th component of the GMM model defined in (4.37) is:*

$$\widehat{\mathcal{R}}_i(\epsilon) = \Psi\left(\frac{r_i \mathbf{w}^T \boldsymbol{\eta}}{\|\mathbf{w}\|} - \epsilon\right) \tag{4.39}$$

where $\Psi(x) = \mathbb{P}(Z > x), Z \sim \mathcal{N}(0,1)$.

[3] Specifically, M and N need to satisfy $M > 10N\log N + N - 1$ and $M > CNr_l\sqrt{\log 2N}\|\boldsymbol{\eta}\|$. The constant C is derived in the proof of Theorem 1 in [149].

Proof: For a random drawn data instance $(\mathbf{x}, y)$, the adversarial perturbation is $-y\epsilon\frac{\mathbf{w}}{\|\mathbf{w}\|}$. Let's decompose $\mathbf{x}$ as $r_i y\boldsymbol{\eta} + \mathbf{z}$, where $\mathbf{z} \sim \mathcal{N}(0, \mathbf{I})$. Then, we have

$$\begin{aligned}\widehat{\mathcal{R}}_k(\epsilon) &= \mathbb{P}(y\mathbf{w}^T(\mathbf{x} - y\epsilon\frac{\mathbf{w}}{\|\mathbf{w}\|}) < 0) = \mathbb{P}(y\mathbf{w}^T(r_i y\boldsymbol{\eta} + \mathbf{z} - y\epsilon\frac{\mathbf{w}}{\|\mathbf{w}\|}) < 0) \\ &= \mathbb{P}(-y\mathbf{w}^T\mathbf{z} > r_i\mathbf{w}^T\boldsymbol{\eta} - \epsilon\|\mathbf{w}\|)\end{aligned} \tag{4.40}$$

Since $\mathbf{z} \sim \mathcal{N}(0, \mathbf{I})$, we have $-y\mathbf{w}^T\mathbf{z} \sim \mathcal{N}(0, (-y\mathbf{w}^T)^T(-y\mathbf{w}^T)) = \mathcal{N}(0, \mathbf{w}^T\mathbf{w})$. Furthermore $\frac{-y\mathbf{w}^T\mathbf{z}}{\|\mathbf{w}\|} \sim \mathcal{N}(0,1)$, and we can further simplify $\widehat{\mathcal{R}}_i(\epsilon)$ as follows:

$$\widehat{\mathcal{R}}_k(\epsilon) = \mathbb{P}\left(\frac{-y\mathbf{w}^T\mathbf{z}}{\|\mathbf{w}\|} > \frac{r_i\mathbf{w}^T\boldsymbol{\eta}}{\|\mathbf{w}\|} - \epsilon\right) = \Psi\left(\frac{r_i\mathbf{w}^T\boldsymbol{\eta}}{\|\mathbf{w}\|} - \epsilon\right) \tag{4.41}$$

□

With Lemma 4.3, we can straightforwardly calculate the robust error for all components of the GMM model defined in (4.37):

$$\widehat{\mathcal{R}}(\epsilon) = \sum_{i=1}^{I} p_i\Psi\left(\frac{r_i\mathbf{w}^T\boldsymbol{\eta}}{\|\mathbf{w}\|} - \epsilon\right) \tag{4.42}$$

On the other hand, Theorem 4.4 indicates the parameter $\mathbf{w}$ will converge to the l_2 max margin. However, for arbitrary training set, we do not have the closed form of $\mathbf{w}$, which is a barrier for the further analysis. Nevertheless, results from [149] indicates in the over-parameterization regime, the parameter $\mathbf{w}$ will converge to min-norm interpolation of the data with high probability.

Lemma 4.4 *(Directly from Theorem 1 in [149]) Assume N training instances drawn from the l-th mode of the described distribution in (4.37) and each of them is a M-dimensional vector. If $\frac{M}{N\log N}$ is sufficiently large*[4]*, then the l_2 max margin vector in Equation (4.33) will be the same as the solution of the min-norm interpolation described below with probability at least $\left(1 - \mathcal{O}\left(\frac{1}{N}\right)\right)$.*

$$\overline{\mathbf{w}} = \text{argmin}_{\mathbf{w}}\|\mathbf{w}\| \;\; s.t.\; \forall\, i \in \{1, 2, ..., N\},\; y_i = \mathbf{w}^T\mathbf{x}_i \tag{4.43}$$

Since the min-norm interpolation has a closed solution $\overline{\mathbf{w}} = \mathbf{X}^T(\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{y}$, Lemma 4.4 will greatly facilitate the calculation of $\mathbb{R}(\mathbf{w})$ in Theorem 4.5. To simplify the notation, we first define the following variables.

$$\mathbf{U} = \mathbf{Q}\mathbf{Q}^T,\; \mathbf{d} = \mathbf{Q}\boldsymbol{\eta},\; s = \mathbf{y}^T\mathbf{U}^{-1}\mathbf{y},\; t = \mathbf{d}\mathbf{U}^{-1}\mathbf{d},\; v = \mathbf{y}^T\mathbf{U}^{-1}\mathbf{d} \tag{4.44}$$

[4] Specifically, M and N need to satisfy $M > 10N\log N + N - 1$ and $M > CNr_l\sqrt{\log 2N}\|\boldsymbol{\eta}\|$. The constant C is derived in the proof of Theorem 1 in [149].

Now, we are ready to prove Theorem 4.5.

Proof: Based on (4.42), the key is to simplify the term $\frac{\mathbf{w}^T\boldsymbol{\eta}}{\|\mathbf{w}\|}$, let's denote it by A, then we have:

$$A^2 = \frac{\boldsymbol{\eta}^T\mathbf{w}\mathbf{w}^T\boldsymbol{\eta}}{\mathbf{w}^T\mathbf{w}} = \frac{(\mathbf{y}^T(\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{X}\boldsymbol{\eta})^2}{\mathbf{y}^T(\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{y}} \tag{4.45}$$

The key challenge here is to calculate the term $(\mathbf{X}\mathbf{X}^T)^{-1}$ where $\mathbf{X} = r_l\mathbf{y}\boldsymbol{\eta}^T + \mathbf{Q}$. Here we utilize Lemma 3 of [149] and Woodbury identity [69], we have:

$$\mathbf{y}^T(\mathbf{X}\mathbf{X})^{-1} = \mathbf{y}^T\mathbf{U}^{-1} - \frac{(r_l^2 s\|\boldsymbol{\eta}\|^2 + r_l^2\upsilon^2 + r_l\upsilon - r_l^2 st)\mathbf{y}^T + r_l s\mathbf{d}^T}{r_l^2 s(\|\boldsymbol{\eta}\|^2 - t) + (r_l\upsilon + 1)^2}\mathbf{U}^{-1} \tag{4.46}$$

Here, s, t, υ, $\mathbf{U}$ and $\mathbf{d}$ are defined in Equation (4.44). The scalar divisor comes from the matrix inverse calculation. Based on Equation (4.46), we can then calculate $\mathbf{y}^T(\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{y}$ and $\mathbf{y}^T(\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{X}\boldsymbol{\eta}$.

$$\begin{aligned}\mathbf{y}^T(\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{y} &= s - \frac{(r_l^2 s\|\boldsymbol{\eta}\|^2 + r_l^2\upsilon^2 + r_l\upsilon - r_l^2 st)s + r_l s\upsilon}{r_l^2 s(\|\boldsymbol{\eta}\|^2 - t) + (r_l\upsilon + 1)^2} \\ &= \frac{s}{r_l^2 s(\|\boldsymbol{\eta}\|^2 - t) + (r_l\upsilon + 1)^2}\end{aligned} \tag{4.47}$$

$$\begin{aligned}\mathbf{y}^T(\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{X}\boldsymbol{\eta} &= \mathbf{y}^T(\mathbf{X}\mathbf{X}^T)^{-1}(r_l\mathbf{y}\boldsymbol{\eta}^T + Q)\boldsymbol{\eta} \\ &= r_l\|\boldsymbol{\eta}\|^2\mathbf{y}^T(\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{y} + \mathbf{y}^T(\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{d} \\ &= \frac{r_l s(\|\boldsymbol{\eta}\|^2 - t) + r_l\upsilon^2 + \upsilon}{r_l^2 s(\|\boldsymbol{\eta}\|^2 - t) + (r_l\upsilon + 1)^2}\end{aligned} \tag{4.48}$$

Plug Equation (4.47) and (4.48) into (4.45), we have:

$$\begin{aligned}A^2 &= \frac{\left(r_l s(\|\boldsymbol{\eta}\|^2 - t) + r_l\upsilon^2 + \upsilon\right)^2}{s\left(r_l^2 s(\|\boldsymbol{\eta}\|^2 - t) + (r_l\upsilon + 1)^2\right)} \\ &= \frac{s(\|\boldsymbol{\eta}\|^2 - t) + \upsilon^2}{s} - \frac{\|\boldsymbol{\eta}\|^2 - t}{r_l^2 s(\|\boldsymbol{\eta}\|^2 - t) + (r_l\upsilon + 1)^2} \\ &= \frac{s(\|\boldsymbol{\eta}\|^2 - t) + \upsilon^2}{s} - \frac{1}{\left(\frac{s(\|\boldsymbol{\eta}\|^2 - t) + \upsilon^2}{\|\boldsymbol{\eta}\|^2 - t}\right)r_l^2 + \frac{2\upsilon}{\|\boldsymbol{\eta}\|^2 - t}r_l + \frac{1}{\|\boldsymbol{\eta}\|^2 - t}}\end{aligned} \tag{4.49}$$

Plug (4.49) into (4.42), we then obtain the robust error on all components of the GMM defined

in (4.37):

$$\mathcal{R}(r_l,\epsilon) = \sum_{i=1}^{I} p_i \Psi\left(r_i g(r_l) - \epsilon\right),\ g(r_l) = \left(C_1 - \frac{1}{C_2 r_l^2 + C_3}\right)^{\frac{1}{2}}$$
$$C_1 = \frac{s(\|\boldsymbol{\eta}\|^2 - t) + v^2}{s},\ C_2 = \frac{s(\|\boldsymbol{\eta}\|^2 - t) + v^2}{\|\boldsymbol{\eta}\|^2 - t},\ C_3 = \frac{2v}{\|\boldsymbol{\eta}\|^2 - t} r_l + \frac{1}{\|\boldsymbol{\eta}\|^2 - t}. \tag{4.50}$$

We study the sign of C_1 and C_2. Consider $\mathbf{U} = \mathbf{Q}\mathbf{Q}^T$ is a positive semidefinite matrix, so $s = \mathbf{y}\mathbf{U}^{-1}\mathbf{y}^T \geq 0$. In addition, we have $\|\boldsymbol{\eta}\|^2 - t = \boldsymbol{\eta}^T\left(\mathbf{I} - (\mathbf{Q}\mathbf{Q}^T)^{-1}\right)\boldsymbol{\eta}$. Since $\mathbf{I} - (\mathbf{Q}\mathbf{Q}^T)^{-1} = (\mathbf{I} - (\mathbf{Q}\mathbf{Q}^T)^{-1})^T(\mathbf{I} - (\mathbf{Q}\mathbf{Q}^T)^{-1})$ is a positive semidefinite matrix, we can obtain $\mathbf{I} - (\mathbf{Q}\mathbf{Q}^T)^{-1}$ is also a positive semidefinite matrix. As a result, C_1 and C_2 are both non-negative. □

We calculate the exact expression of C_1 and C_2 in the proof above. Since C_1 and C_2 are independent of r_l, and $\Psi(x)$ is a monotonically decreasing function, we can conclude that the robust test error $\mathcal{R}(r_l,\epsilon)$ becomes smaller when r_l increases. That is, when the training instances become easier, the corresponding generalization error under the adversarial attack becomes smaller.

Theorem 4.5 holds only if the training data is separable under the adversarial budget. The following corollary shows that the difference in the robust test error between models trained with easy instances and the ones with hard ones increases when ϵ becomes larger.

Corollary 4.2 *Under the conditions of Theorem 4.5 and the definition of* $\mathcal{R}$ *in Equation (4.38), if* $\epsilon_1 < \epsilon_2$*, then we have* $\forall\ 0 \leq i < j \leq I, \mathcal{R}(r_i,\epsilon_1) - \mathcal{R}(r_j,\epsilon_1) < \mathcal{R}(r_i,\epsilon_2) - \mathcal{R}(r_j,\epsilon_2)$.

$\mathcal{R}(r_i,\epsilon) - \mathcal{R}(r_j,\epsilon)$ is the gap in robust accuracy between the models trained on the easy instances and the ones on the hard instances under the adversarial budget $\mathbb{S}_\epsilon^{(2)}$. Corollary 4.2 shows that such a gap increases with the size of the adversarial budget. This indicates that, compared with training on the clean inputs, i.e., $\epsilon = 0$, the generalization performance of adversarial training with $\epsilon > 0$ is more sensitive to the difficulty of the training instances. Such sensitivity also increases with ϵ. This is consistent with our empirical observations in Figure 4.17, Figure 4.18 and Figure 4.19.

To prove Corollary 4.2, we first prove the following lemma:

Lemma 4.5 *Under the condition of Theorem 4.5 and* $\mathcal{R}$ *in Equation (4.38),* $\frac{\partial \mathcal{R}(r_l,\epsilon)}{\partial r_l}$ *is negative and monotonically decreases with* ϵ.

Proof: Based on Equation (4.50), we have:

$$\frac{\partial \mathcal{R}(r_l,\epsilon)}{\partial r_l} = \sum_{i=1}^{I} p_i \Psi'(r_i g(r_l) - \epsilon) \frac{\partial g(r_l)}{\partial r_l} \tag{4.51}$$

Since the training data is separable, we have $\forall i, r_i \mathbf{w}^T \boldsymbol{\eta} - \epsilon \|\mathbf{w}\| > 0$, which is equivalent to the following:

$$\forall i, r_i g(r_l) - \epsilon > 0 \tag{4.52}$$

First, p_i is a positive number by definition. Consider function $\Psi(x)$ monotonically decrease with x and is convex when $x > 0$, so $\forall i, \Psi'(r_i g(r_l) - \epsilon)$ is negative and decreases with ϵ. In addition, $g(r_l)$ increases with r_l and is independent on ϵ, so $\frac{\partial g(r_l)}{\partial r_l}$ can be considered as a positive constant. Therefore, $\frac{\partial \mathcal{R}(r_l,\epsilon)}{\partial r_l}$ is negative and monotonically decreases with ϵ.

□

Now, we are ready to prove Corollary 4.2:

Proof: We subtract the left hand side from the right hand side in the inequality of Corollary 4.2:

$$\begin{aligned}\big[\mathcal{R}(r_j,\epsilon_1) - \mathcal{R}(r_i,\epsilon_1)\big] - \big[\mathcal{R}(r_j,\epsilon_2) - \mathcal{R}(r_i,\epsilon_2)\big] &= \int_{r_i}^{r_j} \frac{\partial \mathcal{R}(r_l,\epsilon_1)}{\partial r_l} d_{r_l} - \int_{r_i}^{r_j} \frac{\partial \mathcal{R}(r_l,\epsilon_2)}{\partial r_l} d_{r_l} \\ &= \int_{r_i}^{r_j} \left[\frac{\partial \mathcal{R}(r_l,\epsilon_1)}{\partial r_l} - \frac{\partial \mathcal{R}(r_l,\epsilon_2)}{\partial r_l} \right] d_{r_l} \\ &> 0\end{aligned} \tag{4.53}$$

The last inequality is based on $r_j > r_i$, $\epsilon_2 > \epsilon_1$, Lemma 4.5, which indicates $\left[\frac{\partial \mathcal{R}(r_l,\epsilon_1)}{\partial r_l} - \frac{\partial \mathcal{R}(r_l,\epsilon_2)}{\partial r_l} \right]$ is always positive. We reorganize (4.53) and obtain $\mathcal{R}(r_i,\epsilon_1) - \mathcal{R}(r_j,\epsilon_1) < \mathcal{R}(r_i,\epsilon_2) - \mathcal{R}(r_j,\epsilon_2)$.

□

4.3.4 Theoretical Analysis for General Models

In this section, we study the binary classification problem on a general nonlinear model with b parameters, i.e., $\theta \in \mathbb{R}^b$. Without loss of generality, we assume the output of the function f to lie in $[-1,+1]$. Furthermore, we assume isoperimetry of the data distribution:

Assumption 4.2 *The data distribution μ is a composition of I c-isoperimetric distributions on $\mathbb{R}^M$, each of which has a positive conditional variance. That is, $\mu = \sum_{i=1}^{I} \alpha_i \mu_i$, where $\alpha_i > 0$ and $\sum_{i=1}^{I} \alpha_i = 1$. We define $\sigma_i^2 = \mathbb{E}_{\mu_i}\left[Var[y|\mathbf{x}]\right]$, and without loss of generality assume that $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_I > 0$. Furthermore, given any L-Lipschitz function f, i.e., $\forall \mathbf{x}_1, \mathbf{x}_2, \|f(\theta,\mathbf{x}_1) - f(\theta,\mathbf{x}_2)\| \leq L\|\mathbf{x}_1 - \mathbf{x}_2\|$, we have the following inequality satisfied $\forall i \in \{1,...,I\}$*

$$\mathbb{P}\left(\mathbf{x} \sim \mu_i, \left\| f(\theta,\mathbf{x}) - \mathbb{E}_{\mu_i}\left(f(\theta,\cdot)\right)\right\| \geq t\right) \leq 2e^{-\frac{mt^2}{2cL^2}}. \tag{4.54}$$

This is a benign assumption; the data distribution is a mixture of I components and each of them contains samples from a sub-Gaussian distribution. These components correspond to training instances of different difficulty levels measured by the conditional variance. We then study the property of the model f under adversarial attacks.

Definition 4.1 *Given the dataset* $\{\mathbf{x}_i, y_i\}_{i=1}^N$*, the model* f *parameterized by* θ*, the adversarial budget* $\mathcal{S}_\epsilon^{(p)}$ *and a positive constant* C*, we define the function* $h(C,\epsilon)$ *as*

$$\begin{aligned} h(C,\epsilon) &= \min_{\theta\in\mathcal{U}(C,\epsilon)} \min_i h_{i,\theta}(\epsilon) \\ \text{where } \mathcal{U}(C,\epsilon) &= \left\{\theta | \frac{1}{N}\sum_{i=1}^N (f(\theta,\mathbf{x}_i') - y_i)^2 \le C\right\}, \\ h_{i,\theta}(\epsilon) &= \max\zeta,\ s.t.\ \left[f(\theta,\mathbf{x}_i)-\zeta, f(\theta,\mathbf{x}_i)+\zeta\right] \subset \left\{f(\theta,\boldsymbol{x}_i+\Delta) | \Delta\in\mathcal{S}_\epsilon^{(p)}\right\}. \end{aligned} \tag{4.55}$$

The function h has the following property.

Lemma 4.6 $\forall C, \epsilon_1 < \epsilon_2,\ h(C,\epsilon_1) \le h(C,\epsilon_2);\ \forall \epsilon, C_1 < C_2,\ h(C_1,\epsilon) \ge h(C_2,\epsilon)$.

Proof: By definition, $h_{i,\theta}(\epsilon) \ge 0$ depicts the bandwidth ζ of the model's output range in the domain of the adversarial budget on a training instance. $h(C,\epsilon)$ is the minimum bandwidth among the models whose mean squared error on the adversarial training set is smaller than C. Based on the definitions of $\mathcal{U}$ and $h_{i,\theta}$, and for a fixed value of C, we have $\forall \epsilon_1 < \epsilon_2$, $h_{i,\theta}(\epsilon_1) \le h_{i,\theta}(\epsilon_2)$ and $\mathcal{U}(C,\epsilon_2) \subset \mathcal{U}(C,\epsilon_1)$. As a result, $\forall \epsilon_1 < \epsilon_2$, $h(C,\epsilon_1) \le h(C,\epsilon_2)$. In addition, since $\forall C_1 < C_2$, $\mathcal{U}(C_1,\epsilon) \subset \mathcal{U}(C_2,\epsilon)$ for a fixed value of ϵ, we have $\forall C_1 < C_2$, $h(C_1,\epsilon) \ge h(C_2,\epsilon)$. □

Therefore, $h(C,\epsilon)$ is a monotonically non-decreasing function on ϵ and a monotonically non-increasing function on C. In practice, when f represents a deep neural network, $h(C,\epsilon)$ increases with ϵ almost surely, because the attack algorithm usually generates adversarial examples at the boundary of the adversarial budget. Based on the monotonicity properties of h, We then state our main theorem below.

Theorem 4.6 *Given* N M*-dimensional training pairs* $\{\mathbf{x}_i, y_i\}_{i=1}^N$ *sampled from the* l*-th component* μ_l *of the distribution in Assumption 4.2, the model* f *parameterized by* $\theta \in \mathbb{R}^b$*, the adversarial budget* $\mathcal{S}_\epsilon^{(p)}$ *and the corresponding function* h *defined in Definition 4.1, we assume that the model* f *is in the function space* $\mathcal{F} = \{f(\theta,\cdot), \theta\in\mathcal{W}\}$ *with* $\mathcal{W} \subset \mathbb{R}^b$ *having a finite diameter* $diam(\mathcal{W}) \le W$ *and,* $\forall \mathbf{w}_1, \mathbf{w}_2 \in \mathcal{W}, \|f(\theta_1,\cdot) - f(\theta_2,\cdot)\|_\infty \le J\|\theta_1 - \theta_2\|_\infty$*. We train the model* f *adversarially using these* N *data points. Let* $\mathbf{x}'$ *be the adversarial example of the data point* $\mathbf{x}$ *and* $\delta \in (0,1)$*. If we have* $\frac{1}{N}\sum_{i=1}^N (f(\theta,\boldsymbol{x}_i') - y_i)^2 = C$ *and* $\gamma := \sigma_l^2 + h^2(C,\epsilon) - C \ge 0$*, then with probability at least* $1-\delta$*, the Lipschitz constant of* $f(\theta,\cdot)$ *is lower bounded as*

$$Lip\big(f(\theta,\cdot)\big) \ge \frac{\gamma}{2^7}\sqrt{\frac{NM}{c\big(b\log(4WJ\gamma^{-1}) - \log(\delta/2 - 2e^{-2^{-11}N\gamma^2})\big)}}. \tag{4.56}$$

where $Lip\big(f(\theta,\cdot)\big)$ *is the Lipschitz constant of* $f(\theta,\cdot)$*:*

$$\forall \mathbf{x}_1, \mathbf{x}_2, \|f(\theta,\mathbf{x}_1) - f(\theta,\mathbf{x}_2)\| \leq Lip(f(\theta,\cdot))\|\mathbf{x}_1 - \mathbf{x}_2\| . \tag{4.57}$$

Theorem 4.6 extends the results in [16] to the case of adversarial training. The Lipschitz constant is widely used to bound a model's adversarial vulnerability [120, 154, 155]; larger Lipschitz constants indicate higher adversarial vulnerability. Recall that γ needs to be non-negative, so C is upper bounded. That is to say, our theorem is based on the condition that the model is well fit to the training set, so the adversarial vulnerability is approximately the generalization gap. Note that modern deep neural network models typically have millions of parameters, so $b \gg \max\{c, M, N\}$. In this case, we can approximate the lower bound (4.56) by $Lip(f(\theta,\cdot)) \gtrsim \frac{\gamma}{2^7}\sqrt{\frac{NM}{bc\log(4WJ\gamma^{-1})}}$, and the right hand side increases with γ.

Since $\gamma := \sigma_l^2 + h^2(C,\epsilon) - C$, we can conclude that the Lipschitz upper bound and thus the adversarial vulnerability is affected by three factors: it increases with both σ_l and ϵ but decreases as C increases. This means the adversarial vulnerability of a model increases with the size of the adversarial budget and the difficulty level of the training instances; it also increases as the training mean squared error decreases. That is 1) under the same adversarial budget, the adversarial vulnerability increases with the instances' difficulty, measured by σ_l in our theorem; 2) using the same training instances, the adversarial vulnerability increases with the adversarial budget measured by ϵ; 3) using the same training instances and the adversarial budget, as adversarial training progresses, the mean squared error C on the adversarial training instances becomes smaller, which makes the Lipschitz bound larger, and thus the adversarial vulnerability increases. These three points are consistent with our empirical observations: 1) higher robust test error under the hard training instances and the larger adversarial budget; 2) the increase of the robust test error in the late phase of training when the training loss is small.

Now, we begin to prove Theorem 4.6. We start with the following lemma.

Lemma 4.7 *Given the assumptions of Theorem 4.6, we define* $g(\mathbf{x}) = \mathbb{E}(y|\mathbf{x})$, $z(\mathbf{x}) = y - g(\mathbf{x})$ *and consider* $\gamma = \sigma_l^2 + h^2(C,\epsilon) - C$*, then the following inequality holds.*

$$\begin{aligned}
&\forall a \in (0,1), \mathbb{P}\left(\exists f(\theta,\cdot) \in \mathcal{F} : \frac{1}{N}\sum_{i=1}^{N}\left(y_i - f(\theta,\boldsymbol{x}_i')\right)^2 \leq C\right) \\
&\leq 2e^{-\frac{Na^2\gamma^2}{8}} + \mathbb{P}\left(\exists f(\theta,\cdot) \in \mathcal{F} : \frac{1}{N}\sum_{i=1}^{N} f(\theta,\boldsymbol{x}_i) z(\boldsymbol{x}_i) \geq \frac{1}{2}(1-3a)\gamma\right)
\end{aligned} \tag{4.58}$$

Proof: Given the definition of $h(C,\epsilon)$, we have:

$$\begin{aligned}
(y_i - f(\theta,\mathbf{x}_i'))^2 &= [(y_i - f(\theta,\mathbf{x}_i)) + (f(\theta,\mathbf{x}_i) - f(\theta,\mathbf{x}_i'))]^2 \\
&\geq (y_i - f(\theta,\mathbf{x}_i))^2 + (f(\theta,\mathbf{x}_i) - f(\theta,\mathbf{x}_i'))^2 \\
&\geq (y_i - f(\theta,\mathbf{x}_i))^2 + h^2(C,\epsilon)
\end{aligned} \tag{4.59}$$

For the first inequality, $\mathbf{x}_i'$ is the adversarial example which tries to maximize the loss objective, $y_i \in \{-1,+1\}$ and the range of f is $[-1,+1]$, so $\langle y_i - f(\theta,\mathbf{x}_i), f(\theta,\mathbf{x}_i) - f(\theta,\mathbf{x}_i')\rangle \geq 0$. The second inequality is based on the definition of $h^2(C,\epsilon)$ in Definition 4.1. As a result, we can simplify the left hand side of (4.58) as follows:

$$\mathbb{P}\left(\exists f(\theta,\cdot)\in\mathcal{F}: \frac{1}{N}\sum_{i=1}^{N}(y_i - f(\theta,\boldsymbol{x}_i'))^2 \leq C\right) \leq \mathbb{P}\left(\exists f(\theta,\cdot)\in\mathcal{F}: \frac{1}{N}\sum_{i=1}^{N}(y_i - f(\theta,\boldsymbol{x}_i))^2 \leq C - h^2(C,\epsilon)\right) \tag{4.60}$$

We consider the sequence $\{z(\mathbf{x}_i)\}_{i=1}^N$, it is i.i.d with $\mathbb{E}_{\mu_l}(z(\mathbf{x})^2) = \mathbb{E}_{\mu_l}[Var(y|\mathbf{x})] = \sigma_l^2$. Since the range of the prediction is $[-1,+1]$, so $z^2(\mathbf{x}) \in [0,4]$. Then, we have the following inequality by Hoeffding's inequality [68].

$$\forall a \in (0,1), \mathbb{P}\left(\frac{1}{N}\sum_{i=1}^{N} z^2(\mathbf{x}_i) \leq \sigma_l^2 - a\gamma\right) \leq e^{-\frac{na^2\gamma^2}{8}} \tag{4.61}$$

Similarly, we consider the sequence $\{z(\mathbf{x}_i)g(\mathbf{x}_i)\}_{i=1}^N$, the following inequality holds based on the Hoeffding's inequality and the fact $\mathbb{E}(z(\mathbf{x})g(\mathbf{x})) = 0$, $z(\mathbf{x})g(\mathbf{x}) \in [-2,+2]$.

$$\forall a \in (0,1), \mathbb{P}\left(\frac{1}{N}\sum_{i=1}^{N} z(\mathbf{x}_i)g(\mathbf{x}_i) \leq a\gamma\right) \leq e^{-\frac{Na^2\gamma^2}{8}} \tag{4.62}$$

Now we study the right hand side of (4.60):

$$\begin{aligned}\frac{1}{N}\sum_{i=1}^{N}(y_i - f(\theta,\mathbf{x}_i))^2 &= \frac{1}{N}\sum_{i=1}^{N}\left(z^2(\mathbf{x}_i) + (g(\mathbf{x}_i) - f(\theta,\mathbf{x}_i))^2 + 2z(\mathbf{x}_i)(g(\mathbf{x}_i) - f(\theta,\mathbf{x}_i))\right) \\ &\geq \frac{1}{N}\sum_{i=1}^{N}\left(z^2(\mathbf{x}_i) + 2z(\mathbf{x}_i)g(\mathbf{x}_i) - 2z(\mathbf{x}_i)f(\theta,\mathbf{x}_i)\right)\end{aligned} \tag{4.63}$$

Consider the following reasoning:

$$\begin{cases} \frac{1}{N}\sum_{i=1}^{N}(y_i - f(\theta,\mathbf{x}_i))^2 \leq C - h^2(C,\epsilon) = \sigma_l^2 - \gamma \\ \frac{1}{N}\sum_{i=1}^{N} z^2(\mathbf{x}_i) \geq \sigma_l^2 - a\gamma \\ \frac{1}{N}\sum_{i=1}^{N} z(\mathbf{x}_i)g(\mathbf{x}_i) \geq -a\gamma \end{cases} \Longrightarrow \frac{1}{N}\sum_{i=1}^{N} z(\mathbf{x}_i)f(\theta,\boldsymbol{x}_i) \geq \frac{1}{2}(1-3a)\gamma \tag{4.64}$$

As a result, we have:

$$\begin{aligned}
&\mathbb{P}\left(\exists f(\theta,\cdot)\in\mathcal{F}:\frac{1}{N}\sum_{i=1}^{N}(y_i-f(\theta,\mathbf{x}_i)\leq C-h^2(C,\epsilon))\right)\\
\leq&\mathbb{P}\left(\exists f(\theta,\cdot)\in\mathcal{F}:\frac{1}{N}\sum_{i=1}^{N}z^2(\mathbf{x}_i)\leq\sigma_l^2-a\gamma\right)+\mathbb{P}\left(\exists f(\theta,\cdot)\in\mathcal{F}:\frac{1}{N}\sum_{i=1}^{N}z(\mathbf{x}_i)g(\mathbf{x}_i)\geq-a\gamma\right)+\\
&\mathbb{P}\left(\exists f(\theta,\cdot)\in\mathcal{F}:\frac{1}{N}\sum_{i=1}^{N}z(\mathbf{x}_i)f(\theta,\mathbf{x}_i)\geq\frac{1}{2}(1-3a)\gamma\right)\\
\leq&2e^{-\frac{Na^2\gamma^2}{8}}+\mathbb{P}\left(\exists f(\theta,\cdot)\in\mathcal{F}:\frac{1}{N}\sum_{i=1}^{N}z(\mathbf{x}_i)f(\theta,\mathbf{x}_i)\geq\frac{1}{2}(1-3a)\gamma\right)
\end{aligned}\tag{4.65}$$

The first inequality is based on the reasoning of (4.64). The second inequality is based on (4.61) and (4.62).

Based on the inequality (4.60) and (4.65), we prove the Lemma 4.7. □

To further simplify the right hand side of (4.58), $\mathbb{P}\left(\exists f(\theta,\cdot)\in\mathcal{F}:\frac{1}{N}\sum_{i=1}^{N}z(\mathbf{x}_i)f(\theta,\mathbf{x}_i)\geq\frac{1}{2}(1-3a)\gamma\right)$ needs to be bounded, and this is solved by the following lemma. We demonstrate the corresponding lemma followed by its proof.

Lemma 4.8 *Given the assumptions of Theorem 4.6 and the definition of $g(\mathbf{x})$, $z(\mathbf{x})$ in Lemma 4.7, then the following inequality holds.*

$$\begin{aligned}
&\forall a\in(0,1),a_1>0,a_2>0\text{ and }a_1+a_2=\frac{1}{2}(1-3a),\\
&\mathbb{P}\left(\exists f(\theta,\cdot)\in\mathcal{F}:\frac{1}{N}\sum_{i=1}^{N}z(\mathbf{x}_i)f(\theta,\mathbf{x}_i)\geq\frac{1}{2}(1-3a)\gamma\right)\leq 2|\mathcal{F}|e^{-\frac{NM}{144cL^2}a_1^2\gamma^2}+2e^{-\frac{N}{8}a_2^2\gamma^2}
\end{aligned}\tag{4.66}$$

Proof: We recall that the data points $\{\mathbf{x}_i,y_i\}_{i=1}^N$ are sampled from the distribution μ_l, which is c-isoperimetric. For any L-Lipschitz function f, we have:

$$\forall t,\mathbb{P}\left(|f(\theta,\mathbf{x})-\mathbb{E}_{\mu_l}(f(\theta,\cdot))|\geq t\right)\leq 2e^{-\frac{Mt^2}{2cL^2}}\tag{4.67}$$

Since $z(\mathbf{x})=y-g(\mathbf{x})\in[-2,+2]$, we can then bound $\mathbb{P}\left(z(\mathbf{x})(f(\theta,\mathbf{x})-\mathbb{E}_{\mu_l}(f(\theta,\cdot)))\geq t\right)$:

$$\begin{aligned}
\forall t,\mathbb{P}\left(z(\mathbf{x})(f(\theta,\mathbf{x})-\mathbb{E}_{\mu_l}(f(\theta,\cdot)))\geq t\right)&\leq\mathbb{P}\left(|z(\mathbf{x})(f(\theta,\mathbf{x})-\mathbb{E}_{\mu_l}(f(\theta,\cdot)))|\geq t\right)\\
&\leq\mathbb{P}\left(|(f(\theta,\mathbf{x})-\mathbb{E}_{\mu_l}(f(\theta,\cdot)))|\geq\frac{t}{2}\right)\leq 2e^{-\frac{Mt^2}{8cL^2}}
\end{aligned}\tag{4.68}$$

Here we utilize the proposition in [148, 146][5], which claims *if* $\{X_i\}_{i=1}^N$ *are independent variables and all* C*-subgaussian, then* $\frac{1}{\sqrt{N}}\sum_{i=1}^N X_i$ *is* $18C$*-subgaussian.* Therefore, we have:

$$\forall t, \mathbb{P}\left(\frac{1}{\sqrt{N}}\sum_{i=1}^{N} z(\mathbf{x}_i)(f(\theta,\mathbf{x}_i) - \mathbb{E}_{\mu_l}(f(\theta,\cdot))) \geq t\right) \leq 2e^{-\frac{Mt^2}{144cL^2}} \tag{4.69}$$

Let $t = a_1\gamma\sqrt{N}$, then we have:

$$\mathbb{P}\left(\frac{1}{N}\sum_{i=1}^{N} z(\mathbf{x}_i)(f(\theta,\mathbf{x}_i) - \mathbb{E}_{\mu_l}(f(\theta,\cdot))) \geq a_1\gamma\right) \leq 2e^{-\frac{NM}{144cL^2}a_1^2\gamma^2} \tag{4.70}$$

In addition, we can bound $\mathbb{P}\left(\frac{1}{N}\sum_{i=1}^N z(\mathbf{x}_i)\mathbb{E}_{\mu_l}(f(\theta,\cdot)) \geq a_2\gamma\right)$ by:

$$\mathbb{P}\left(\exists f(\theta,\cdot)\in\mathcal{F}: \frac{1}{N}\sum_{i=1}^{N} z(\mathbf{x}_i)\mathbb{E}_{\mu_l}(f(\theta,\cdot)) \geq a_2\gamma\right) \leq \mathbb{P}\left(\frac{1}{N}\sum_{i=1}^{N} |z(\mathbf{x}_i)| \geq a_2\gamma\right) \leq 2e^{-\frac{N}{8}a_2^2\gamma^2} \tag{4.71}$$

The first inequality is based on the fact $\mathbb{E}_{\mu_l}(f(\theta,\cdot)) \in [-1,+1]$; the second inequality is based on Hoeffding's inequality.

Now, we are ready to bound the probability $\mathbb{P}\left(\exists f(\theta,\cdot)\in\mathcal{F}: \frac{1}{N}\sum_{i=1}^N z(\mathbf{x}_i)f(\theta,\boldsymbol{x}_i) \geq \frac{1}{2}(1-3a)\gamma\right)$.

$$\begin{aligned}
&\mathbb{P}\left(\exists f(\theta,\cdot)\in\mathcal{F}: \frac{1}{N}\sum_{i=1}^{N} z(\mathbf{x}_i)f(\theta,\boldsymbol{x}_i) \geq \frac{1}{2}(1-3a)\gamma\right)\\
\leq&\mathbb{P}\left(\exists f(\theta,\cdot)\in\mathcal{F}: \frac{1}{N}\sum_{i=1}^{N} z(\mathbf{x}_i)(f(\theta,\boldsymbol{x}_i) - \mathbb{E}_{\mu_l}(f(\theta,\cdot))) \geq a_1\gamma\right)\\
&+\mathbb{P}\left(\exists f(\theta,\cdot)\in\mathcal{F}: \frac{1}{N}\sum_{i=1}^{N} z(\mathbf{x}_i)\mathbb{E}_{\mu_l}(f(\theta,\cdot)) \geq a_2\gamma\right)\\
\leq&2|\mathcal{F}|e^{-\frac{NM}{144cL^2}a_1^2\gamma^2} + 2e^{-\frac{N}{8}a_2^2\gamma^2}
\end{aligned} \tag{4.72}$$

The first inequality is based on the fact $a_1 + a_2 = \frac{1}{2}(1-3a)$; the second inequality is based on the Boole's inequality [13], inequality (4.70) and (4.71). □

To simplify the constant notation, we let $a = \frac{1}{8}$, $a_1 = \frac{3}{16}$ and $a_2 = \frac{1}{8}$. We plug this into the inequality (4.58) and (4.66), then:

$$\mathbb{P}\left(\exists f(\theta,\cdot)\in\mathcal{F}: \frac{1}{N}\sum_{i=1}^{N} (y_i - f(\theta,\boldsymbol{x}_i'))^2 \leq C\right) \leq 4e^{-\frac{N\gamma^2}{2^9}} + 2|\mathcal{F}|e^{-\frac{NM\gamma^2}{2^{12}cL^2}} \tag{4.73}$$

[5] Proposition 2.6.1 in [148] and Exercise 3.1 in [146]

Now we turn to the proof of Theorem 4.6.

Proof: We let $\mathcal{F}_L = \{f(\theta,\cdot)|\theta \in \mathcal{W}, Lip(f(\theta,\cdot)) \leq L\}$, $\mathcal{F}_\gamma = \{f(\theta,\cdot)|\theta \in \mathcal{W}, \theta = \frac{\gamma}{4J} \odot \mathbf{z}, \mathbf{z} \in \mathbb{Z}^b\}$ and $\mathcal{F}_{\gamma,L} = \mathcal{F}_\gamma \cap \mathcal{F}_L$. Correspondingly, we let $\mathcal{W}_L = \{\theta|\theta \in \mathcal{W}, Lip(f(\theta,\cdot)) \leq L\}$, $\mathcal{W}_\gamma = \{\theta|\theta \in \mathcal{W}, \theta = \frac{\gamma}{4J} \odot \mathbf{z}, \mathbf{z} \in \mathbb{Z}^b\}$ and $\mathcal{W}_{\gamma,L} = \mathcal{W}_\gamma \cap \mathcal{W}_L$. Because the diameter of $\mathcal{W}$ is W, we have $|\mathcal{F}_{\gamma,L}| \leq |\mathcal{F}_\gamma| \leq \left(\frac{4WJ}{\gamma}\right)^b$. Here, $\odot$ means the element-wise multiplication.

Note that the inequality (4.73) is valid for any values of C as long as it satisfies $\gamma \geq 0$. Based on this, we apply the substitution $\begin{cases} C \leftarrow C + \frac{1}{2}\gamma \\ \gamma \leftarrow \frac{1}{2}\gamma \end{cases}$, then:

$$\begin{aligned} \mathbb{P}\left(\exists f(\theta,\cdot) \in \mathcal{F}_{\gamma,L} : \frac{1}{N}\sum_{i=1}^{N}(y_i - f(\theta,\mathbf{x}_i'))^2 \leq C + \frac{1}{2}\gamma\right) &\leq 4e^{-\frac{N\gamma^2}{2^{11}}} + 2|\mathcal{F}|e^{-\frac{NM\gamma^2}{2^{14}cL^2}} \\ &\leq 4e^{-\frac{N\gamma^2}{2^{11}}} + 2e^{b\log(\frac{4WJ}{\gamma}) - \frac{NM\gamma^2}{2^{14}cL^2}} \end{aligned} \tag{4.74}$$

Based on the definition of $\mathcal{W}_{\gamma,L}$, we can conclude that $\forall\theta_1 \in \mathcal{W}_L, \exists\theta_2 \in \mathcal{W}_{\gamma,L}\ s.t. \|\theta_1 - \theta_2\|_\infty \leq \frac{\gamma}{8J}$. Therefore, $\forall f(\theta_1,\cdot) \in \mathcal{F}_L, \exists f(\theta_2,\cdot) \in \mathcal{F}_{\gamma,L} s.t. \|f(\theta_1,\cdot) - f(\theta_2,\cdot)\|_\infty \leq \frac{\gamma}{8}$. Let choose such $f(\theta_2,\cdot) \in \mathcal{F}_{\gamma,L}$ given an arbitrary $f(\theta_1,\cdot) \in \mathcal{F}_L$, then:

$$\begin{aligned} (y - f(\theta_1,\mathbf{x}))^2 &= (y - f(\theta_2,\mathbf{x}))^2 + (2y - f(\theta_1,\boldsymbol{x}) - f(\theta_2,\mathbf{x}))(f(\theta_2,\mathbf{x}) - f(\theta_1,\mathbf{x})) \\ &\geq (y - f(\theta_2,\mathbf{x}))^2 - \frac{\gamma}{8}|(2y - f(\theta_1,\mathbf{x}) - f(\theta_2,\mathbf{x}))| \\ &\geq (y - f(\theta_2,\mathbf{x}))^2 - \frac{\gamma}{2} \end{aligned} \tag{4.75}$$

The first inequality in (4.75) is based on Hölder's inequality; the second inequality is based on $y \in \{-1,+1\}$ and the range of $\forall f(\theta,\cdot) \in \mathcal{F}$ is $[-1,+1]$.

We combine (4.73) with (4.75), then:

$$\begin{aligned} \mathbb{P}\left(\exists f(\theta,\cdot) \in \mathcal{F}_L : \frac{1}{N}\sum_{i=1}^{N}(y_i - f(\theta,\boldsymbol{x}_i'))^2 \leq C\right) &\leq \mathbb{P}\left(\exists f(\theta,\cdot) \in \mathcal{F}_{\gamma,L} : \frac{1}{N}\sum_{i=1}^{N}(y_i - f(\theta,\boldsymbol{x}_i'))^2 \leq C + \frac{1}{2}\gamma\right) \\ &\leq 4e^{-\frac{N\gamma^2}{2^{11}}} + 2e^{b\log(\frac{4WJ}{\gamma}) - \frac{NM\gamma^2}{2^{14}cL^2}} \end{aligned} \tag{4.76}$$

Note that $\mathcal{F}_L$ is the set of functions in $\mathcal{F}$ whose Lipschitz constant is no larger than L. We set the right hand side of (4.76) to be δ and then get $L = \frac{\gamma}{2^7}\sqrt{\frac{NM}{c\left(b\log(4WJ\gamma^{-1}) - \log(\delta/2 - 2e^{-2^{-11}N\gamma^2})\right)}}$. This concludes the proof. □

Numerical Validation

Training Set	Lipschitz in l_∞ Cases ($\times 10^4$)			Lipschitz in l_2 Cases ($\times 10^4$)		
	$\epsilon = 2/255$	$\epsilon = 4/255$	$\epsilon = 8/255$	$\epsilon = 0.50$	$\epsilon = 0.75$	$\epsilon = 1.00$
Easy10K	5.91	6.06	14.54	3.34	3.67	3.91
Random10K	28.98	79.96	93.63	30.01	31.28	39.34
Hard10K	72.42	117.60	567.24	60.62	80.06	77.55

Table 4.5 – Upper bound of the Lipschitz constant under different settings of ϵ and training instances.

We conduct empirical analyses to confirm the validity of Theorem 4.6 in our settings. To this end, we use the CIFAR10 dataset and an ResNet18 network architecture. Since calculating the Lipschitz constant of a deep neural network is NP-hard [125], exactly calculating the Lipschitz constant [78] can only be achieved for simple multi-layer perceptron (MLP) models, not for modern deep networks. Instead, we therefore estimate the upper bound of the Lipschitz constant numerically, as in [125].

Table 4.5 provides the upper bound of the Lipschitz constant of models trained by different subsets of the training data and different adversarial budget. Due to the stochasticity introduced by the algorithm of [125], we run it 20 times and report the average; we observed the variance to be negligible. Based on the results in Table 4.5, it is clear that the models adversarially trained on the hard training instances have a much larger Lipschitz constant than the ones trained on the easy instances.

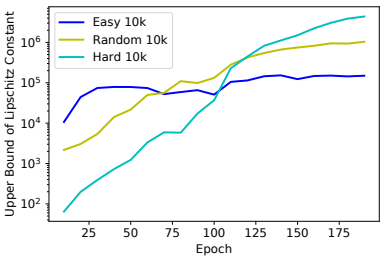
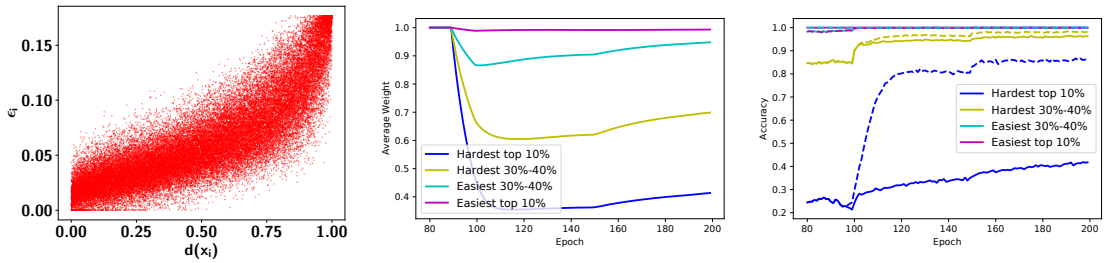


Figure 4.22 – Curves of the Lipschitz upper bound when the model is adversarially trained by the easiest, random and the hardest 10000 instances. The y-axis is in log-scale.

Figure 4.22 depicts the curves of the Lipschitz upper bound when the model is adversarially trained by the easiest, random, and the hardest 10000 instances. The adversarial budget is based on the l_∞ norm with $\epsilon = 8/255$. We can clearly see that, as training progresses, the Lipschitz upper bound increases in all cases. Furthermore, compared with training on easy instances, the Lipschitz upper bound of the models adversarially trained on hard instances increases much faster. These results are consistent with Theorem 4.6.

4.3.5 Case Studies

Our empirical observation and theoretical analyses indicate that fitting hard adversarial leads to adversarial overfitting. In this section, we first study existing approaches to mitigating adversarial overfitting, and show that they implicitly avoid fitting hard adversarial instances, which



(a) (b) (c)

Figure 4.23 – Results of the case study. The model is always ResNet18 and the target adversarial budget's size $\epsilon = 8/255$. (a) Relationship between instance difficulty $d_\epsilon(\boldsymbol{x}_i)$ and its adversarial budget size in IAT for the CIFAR10 training set. (b) Average weights of different groups in the CIFAR10 training set during training in SAT. The warmup period is 90 epochs, and SAT is enabled after that. (c) Training accuracy of different groups on the CIFAR10 training set during training in SAT. The solid lines and the dashed lines represent the accuracy on the ground truth and on the adaptive targets, respectively. The warmup period is 90 epochs, and SAT is enabled after that.

provides an explanation for their success. We also show that the methods that encourage fitting hard adversarial instances fail to yield truly robust models. In addition to standard adversarial training, we also study fast adversarial training, as well as adversarial fine-tuning with additional training data. Our results indicate that avoiding fitting hard adversarial instances also improves the performance in these cases.

A New Perspective on Existing Methods

Existing methods aiming to mitigate adversarial overfitting can be generally divided into two categories: those that use adaptive inputs, such as [8], and those that rely on adaptive targets, such as [21, 71]. We show below that both categories implicitly aim to prevent the model from fitting hard input-target pairs.

We use instance-wise adversarial training (IAT) [8] and self-adaptive training (SAT) [71] as examples of these two categories. IAT uses an instance-adaptive adversarial budget during training. That is, it adaptively adjusts the size of the adversarial budget for each training instance. SAT uses self-supervised adaptive targets instead of the ground truth during training. We run both algorithms using the settings in their original papers, except that we set the training duration to be 200 epochs for a consistent comparison.

Let us study how these algorithms adaptively use instances of different difficulty levels. For IAT, we plot the relationship between the instance difficulty $d_\epsilon(\mathbf{x}_i)$ and its adaptive adversarial budget's size ϵ_i in Figure 4.23a, which shows a high correlation (0.884) between them. Specifically, we find that the hard instances are assigned smaller adversarial budgets for training, which indicates that IAT prevents the model from fitting the hard adversarial instances. For SAT, we

show the average weights assigned to the instances in each group of $\{\mathcal{G}_i\}_{i=0}^{9}$ during training in Figure 4.23b. The hard instances are clearly assigned much smaller weights to calculate the loss, which indicates that they are downplayed during training. We also provide the average accuracy of each group during training in Figure 4.23c, given both the ground truth or the adaptive target. [6] We observe that the hard instances have much higher accuracy on their adaptive targets compared with the ground truth, while such a difference is much smaller for the easy instances. Our results thus indicate that the adaptive targets used by SAT are much easier to fit, which avoids having to directly fit the hard adversarial input-target pairs.

In addition to IAT and SAT, other methods have introduced regularization terms to mitigate adversarial overfitting, such as [178] and [21]. These regularization terms calculate the distance between the adversarial output logits and their anchor points. The anchor points can be considered the adaptive targets, and can be the clean output logits in [178] or a teacher network's outputs in [21]. The regularizers used in these methods encourage the adversarial output logits to be closer to the anchor points other than to the ground truth for the hard instances. In other words, these methods also use adaptive targets to avoid fitting the hard input-target pairs.

In contrast to the methods above, [181] proposed an instance-adaptive reweighting strategy which assigns larger weights to the training instances that PGD breaks in fewer iterations. In other words, this approach assigns larger weights to the hard adversarial instances, which contrasts with what our analysis revealed. As a matter of fact, this method was recently shown to be vulnerable to adaptive attacks [67].

Alternative Training Scenarios

Our findings are applicable to other scenarios than standard adversarial training. In this regard, we conduct preliminary analyses on two examples: fast adversarial training and fine-tuning a pre-trained model using additional data.

Fast Adversarial Training

Adversarial training in [98] introduces a significant computational overhead, so it is desirable to accelerate this method. Our experiments in this section are based on adversarial training with transferable adversarial examples (ATTA in [182]), which stores the adversarial perturbation for each training instance as an initial point for the next epoch. We show that adaptively utilizing the easy and hard training instances not only mitigates adversarial overfitting, but also significantly improves the performance of the final model.

First, we use a reweighting scheme to assign lower weights to hard instances when calculating the loss objective. Specifically, each training instance is assigned a weight equaling to the adversarial output probability of the true label. Then this weight is normalized to ensure

[6] For the adaptive target $\mathbf{t}$, the prediction $\mathbf{o}$ is considered correct if and only if $\text{argmax}_i \mathbf{t}_i = \text{argmax}_i \mathbf{o}_i$.

that the weights in a mini-batch sum to 1. Note that our reweighting scheme is based on the adversarial output instead of the clean output, because the adversarial output probability will also be used to calculate the loss objective. As a result, the computational overhead of the reweighting scheme is negligible.

In addition to reweighting, we follow the idea of SAT [71] and use adaptive targets to improve the performance. For each training instance $(\mathbf{x}, y)$, we maintain an adaptive moving average target $\widetilde{\mathbf{t}}$. $\widetilde{\mathbf{t}}$ is updated in an exponential average manner in each epoch $\widetilde{\mathbf{t}} \leftarrow \rho\widetilde{\mathbf{t}} + (1-\rho)\mathbf{o}'$ where ρ is the momentum factor and $\mathbf{o}'$ is the output probability of the adversarial input. This is similar to the target in [71], but, similarly to the reweighting scheme, we use the adversarial output $\mathbf{o}'$ instead of the clean output $\mathbf{o}$ to avoid an increase in computational complexity. The final adaptive target we use is $\mathbf{t} = \beta\mathbf{1}_y + (1-\beta)\widetilde{\mathbf{t}}$ and thus the loss objective is $\mathcal{L}(f(\theta, \mathbf{x}+\Delta), \mathbf{t})$. The factor β controls how "adaptive" our target is: $\beta = 0$ yields a fully adaptive moving average target $\widetilde{\mathbf{t}}$ and $\beta = 1$ yields a one-hot target $\mathbf{1}_y$. We provide the pseudocode as Algorithm 4.1 below.

Algorithm 4.1: One epoch of the accelerated adversarial training we use.

Input: training data $\mathcal{D}$, model f, batch size B, PGD step size α, adversarial budget $\mathbb{S}_\epsilon^{(p)}$, coefficient ρ, β.
for Sample a mini-batch $\{\mathbf{x}_i, y_i\}_{i=1}^B \sim \mathcal{D}$ **do**
  $\forall i$, obtain the initial perturbation Δ_i as in [182].
  $\forall i$, one step PGD update: $\Delta_i \leftarrow \Pi_{\mathbb{S}_\epsilon^{(p)}}\left[\Delta_i + \alpha\text{sign}(\nabla_{\Delta_i}\mathcal{L}(f(\theta, \mathbf{x}_i+\Delta_i), y_i))\right]$.
  $\forall i$, update the cached adversarial perturbation Δ_i as in [182].
  if use reweight **then**
    $\forall i$, weight $w_i = \text{softmax}[f(\mathbf{x}_i+\Delta_i)]_{y_i}$
  else
    $\forall i$, weight $w_i = 1$
  end if
  $\forall i$, query the adaptive target $\widetilde{\mathbf{t}}_i$ and update: $\widetilde{\mathbf{t}}_i \leftarrow \rho\widetilde{\mathbf{t}}_i + (1-\rho)\text{softmax}[f(\mathbf{x}_i+\Delta_i)]$.
  $\forall i$, the final adaptive target $\mathbf{t}_i = \beta\mathbf{1}_{y_i} + (1-\beta)\widetilde{\mathbf{t}}_i$
  Calculate the loss $\frac{1}{\sum_i^B w_i}\sum_i^B w_i\mathcal{L}(f(\theta, \mathbf{x}_i+\Delta_i), \mathbf{t}_i)$ and update the parameters.
end for

Our experiment is on CIFAR10 and $\epsilon = 8/255$, the standard setting where most fast adversarial training algorithms are benchmarked [30]. The step size α of the perturbation update is $4/255$, same as [182]. The average coefficient ρ and β is 0.9 and 0.1 unless explicitly stated. Our learning rate scheduler also follows [182]: we train the model for 38 epochs, the learning rate is 0.1 on the first 30 epochs, it decays to 0.01 in the next 6 epochs and further decays to 0.001 in the last 2 epochs. When we use adaptive targets, the first 5 epochs are the warmup period in which we use fixed targets. Since the goal here is to accelerate adversarial training, we do not use a validation set to do model selection as in [118]. We use the standard data augmentation on CIFAR10: random crop and random horizontal flip. We evaluate the model's robust accuracy on the test set by AutoAttack [33], the popular and reliable attack for evaluation.

Method	Model	Epochs	Complexity	AA
[130]	WideResNet34	200	2	41.17
[158]	ResNet18	15	4	43.21
[182]	WideResNet34	38	4	44.48
[177]	WideResNet34	105	3	44.83
[20]	WideResNet34	100	7	51.12
Reweighting (Ours)	WideResNet34	38	4	46.15
Adaptive Target (Ours)	WideResNet34	38	4	51.17

Table 4.6 – Comparison between different accelerated adversarial training methods in robust test accuracy against AutoAttack (AA). The baseline results are from RobustBench. *Complexity* shows the number of forward passes and backward passes in one mini-batch update.

The results are provided in Table 4.6, where the results of the baseline methods are taken from RobustBench [30]. We also report the number of epochs and the number of forward and backward passes in a mini-batch update of each method. The product of these two values indicates the training complexity. We can clearly see that both reweighting and adaptive targets improve the performance on top of ATTA [182]. Note that our method based on adaptive targets achieve the best performance while needing only 1/4 of the training time of [20], the strongest baseline. [158] is the only baseline consuming less training time than ours, but its performance is much worse than ours; it suffers from catastrophic overfitting when using a WideResNet34 model.

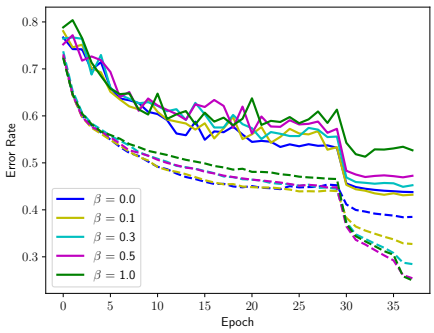


Figure 4.24 – The learning curves with different values of β. The solid curve and the dashed curve represent the robust test error and the robust training error, respectively.

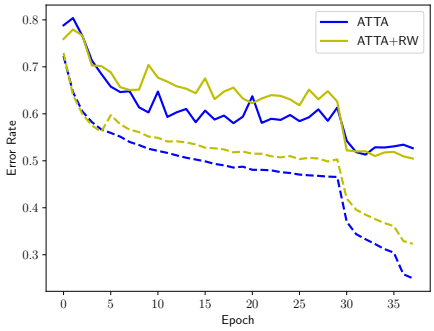


Figure 4.25 – The learning curves of ATTA with and without reweighting. The solid curve and the dashed curve represent the robust test error and the robust training error, respectively.

We conduct ablation study in the context of fast adversarial training. In Figure 4.24, we change the value of β in Algorithm 4.1 and plot the corresponding learning curves. Lower the value of β is, more weights assigned to the adaptive part of the target: $\beta = 0$ means we only utilize the moving average target as the final target, $\beta = 1$ means we use the one-hot groundtruth label. Figure 4.24 clearly shows us the generalization gap decreases with the decrease in β. That is to

say, the adaptive target can indeed improve the generalization performance.

Figure 4.25 compare the learning curves of ATTA [182] with and without reweighting. The first 5 epochs are the warmup period. The results confirm that the reweighting scheme can prevent adversarial overfitting and decrease the generalization gap.

To confirm that the algorithm we use is consistent with our theoretical and empirical analysis, we study the relationship between the instance difficulty and the weight assigned to them when using reweighting, as well as the soft target when using adaptive targets. Since the evaluation of model robustness is based on the PGD attack, the difficulty value here is also based on the PGD perturbation. In Figure 4.26, we demonstrate the relationship between the difficulty value and the average assigned weight for each instance when using reweighting. We calculate the correlation between these two values on the training set, it is 0.8900. This indicates we indeed assign smaller weights for hard training instances and assign bigger weights for easy training instances. In Figure 4.27, we show the relationship between the difficulty value and the average value of the true label's probability in the soft target when we use the adaptive targets. Similarly, we calculate the correlation between these two values on the training set, it is 0.9604. This indicates the adaptive target is similar to the ground-truth one-hot target for the easy training instances, while the adaptive target is very different from the ground-truth one-hot target for the hard training instances. This means, adaptive targets prevent the model from fitting hard training instances while encourage the model to fit the easy training instances.

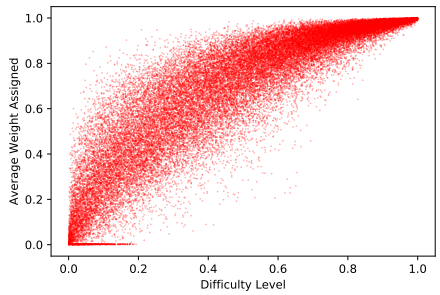


Figure 4.26 – The relationship between the difficulty value and the weight assigned to each instances when using reweighting. We use the average weight across epochs. The correlation between them is 0.8900.

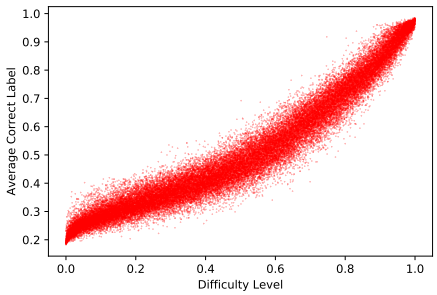


Figure 4.27 – The relationship between the difficulty value and the average value of the true label's probability when using the adaptive targets. The correlation between them is 0.9604.

Adversarial Fine-tuning with Additional Data

We observe that adversarial overfitting occurs in the small learning rate regime. To further study this, we propose to fine-tune an adversarially pretrained model using additional training data, because we also use small learning rate to fine-tune a model. While additional training data was shown to be beneficial in [3, 18], we demonstrate that letting the model adaptively fit

the easy and hard instances can further improve the performance.

We conduct experiments on both CIFAR10 and SVHN, using WideResNet34 and ResNet18 models, respectively. For CIFAR10, we use 500000 images from 80 Million Tiny Images dataset [144] with pseudo labels in [18] [7]. For SVHN, we use the extra held-out set provided by SVHN itself, which contains 531131 somewhat less difficult samples. When we construct a mini-batch, half of its instances are sampled from the original training set and the other half are sampled from the additional data. The experimental settings are the same as [18] except the learning rate. We tune the learning rate and find that fixing it to 10^{-3} is the best choice. The model is fine-tuned for either 1 epoch or 5 epochs, which means that each additional training instance is used either 5 times or only once. This is because we observed the performance of vanilla adversarial training to start decaying after 5 epochs. As such, methods requiring many epochs such as [8] and [71] are not applicable here.

Our first technique, reweighting, is the same as in the previous section. In addition to reweighting, we can also add a KL regularization term measuring the KL divergence between the output probability of the clean instance and of the adversarial instance. The KL term encourages the adversarial output to be close to the clean one. In other words, the clean output probability serves as the adaptive target. For hard instances, the clean and adversarial inputs are usually both misclassified. Therefore, the clean outputs of these instances constitute simpler targets compared with the ground-truth labels. Ultimately, the loss objective of a mini-batch $\{\mathbf{x}_i, y_i\}_{i=1}^B$ used for fine-tuning is expressed as $\mathcal{L}_{FT}\left(\theta, \{\mathbf{x}_i, y_i\}_{i=1}^B\right) = \sum_{i=1}^B w_i \left[\mathcal{L}(f(\theta, \mathbf{x}_i'), y_i) + \lambda KL(\mathbf{o}_i || \mathbf{o}_i')\right]$ where w_i is the adaptive weight when we use re-weighting, or $1/B$ otherwise. $\mathbf{x}_i'$ and $\mathbf{o}_i'$ represent the adversarial input and adversarial output, respectively. λ is 6 when using the regularization term and 0 otherwise.

Duration	Method	AutoAttack	Duration	Method	AutoAttack
WRN34 on CIFAR10, $\epsilon = 8/255$			**RN18 on SVHN,** $\epsilon = 0.02$		
No Fine Tuning		52.01	No Fine Tuning		67.77
1 Epoch	Vanilla AT	54.11	1 Epoch	Vanilla AT	70.81
	RW	54.69		RW	70.83
	KL	54.73		KL	72.29
	RW + KL	54.69		RW + KL	72.53
5 Epoch	Vanilla AT	55.49	5 Epoch	Vanilla AT	72.18
	RW	56.41		RW	72.72
	KL	56.55		KL	73.17
	RW + KL	56.99		RW + KL	73.35

Table 4.7 – Robust accuracy of fine-tuned models against AutoAttack(AA). We conduct ablation study on both reweighting (RW) and KL regularization (KL).

We use both reweighting and KL regularization to fine-tune the model. Our results are shown in Table 4.7, where the robust test accuracy is also evaluated by AutoAttack. It is clear that

[7] Data available for download on https://github.com/yguooo/semisup-adv.

both reweighting and the KL regularization term benefit the performance of the finetuned model. All these results show that avoiding fitting hard adversarial examples helps to improve the generalization performance in adversarial fine-tuning with additional training data.

4.4 Summary and Broader Impact

In this chapter, we have studied empirical robustness and, in particular, adversarial training, the most popular method to achieve empirical robustness. We have conducted a theoretical analysis to understand two of the unsatisfying properties of adversarial training: slow convergence and large generalization gap. Using our theoretical understanding, we designed algorithms to mitigate these challenges and to explain the existing successful methods.

For adversarial loss landscape, our work is the first (to the best of our knowledge) to discover its non-smooth properties in the general case. [79] follows our work and points out that the adversarial loss landscape can be smooth in some special cases, such as binary linear classification under l_2 adversarial attack. We find that, in Figure 4.6, local minima found in adversarial training are sharper than training on clean inputs. In this regard, [161] proposes adversarial weight perturbation to enable the optimizer to find flatter local minima in order to achieve better generalization. Furthermore, Proposition 4.2 indicates scattered gradients in adversarial training. [38] follows our work and proposes methods for improving gradient stability. For adversarial overfitting, our work is a theory-backed analysis in the lens of training data. We have given explanations, based on whether or not to avoid fitting hard adversarial instances, on the success and failure of existing methods. We have shown that, in particular, our discovery still holds in different forms of adversarial training. We believe our findings can be applied in different scenarios and are beneficial for designing new algorithms.

In addition to convergence and generalization, adversarial training still has other challenges in efficiency. In the next chapter, we discuss the efficiency issues in adversarial training. We design methods that achieve light models and low computational costs.

5 Efficient Robust Learning

In Section 1.3, we point out that there are several challenges in robust learning compared with the non-robust counterpart: degradation in clean accuracy, slower convergence, larger generalization gap, larger model capacity requirements, and more training data needs. In Section 4.2 and 4.3, we conduct theoretical analyses to understand the reason for slower convergence and for the larger generalization gap. In this chapter, we focus on designing algorithms to solve or mitigate the challenges we face in robust learning. We improve, in particular, the efficiency of adversarial training in two aspects: (1) in the decrease of the model size, and (2) in the reduction of the training time.

In Section 5.1, we extend the *Lottery Ticket Hypothesis* [45] to the adversarial cases. By introducing adaptive pruning strategy and the binary initialization scheme, we design the algorithm to find a robust sub-network inside a large randomly initialized binary network, without updating its weights. In Section 5.2, we first review the methods for accelerating adversarial training and the issue of *catastrophic overfitting* in this context. Based on instance-adaptive step size in the adversarial example generation, our proposed methods are shown free of catastrophic overfitting with little computational overhead.

The contents of this chapter are mainly from the following papers. I am the primary contributor of the first paper, I contribute to the motivation and theoretical proofs of the second paper.

- Chen Liu, Ziqi Zhao, Sabine Süsstrunk, Mathieu Salzmann. "Robust Binary Models by Pruning Randomly-initialized Networks." Preprint.
- Zhichao Huang, Yanbo Fan, Chen Liu, Weizhong Zhang, Yong Zhang, Mathieu Salzmann, Sabine Süsstrunk, Jue Wang. "Fast Adversarial Training with Adaptive Step Size." Preprint.

5.1 Robust Subnetwork inside Randomly-initialized Networks

We introduce methods that contain compressed and robust models by pruning randomly-initialized networks [96]. Our methods are based on the *Strong Lottery Ticket Hypothe-*

sis [113, 183], which studies the non-robust cases. To improve the efficiency and the performance against adversarial attacks, we introduce the adaptive pruning strategy and the binary initialization scheme as shown in the following sections. We then conduct comprehensive experiments to demonstrate the advantages of our methods.

5.1.1 Lottery Ticket Hypothesis

Our method is motivated by the *Lottery Ticket Hypothesis* [45] and the *Strong Lottery Ticket Hypothesis* [113, 183]. The *Lottery Ticket Hypothesis* is the hypothesis that overparameterized neural networks contain sparse subnetworks that can be trained in isolation to achieve competitive performance. These competitive subnetworks are called the *winning tickets*. The *Strong Lottery Ticket Hypothesis* further shows that there exist winning tickets with competitive performance even without training. Motivated by the *Strong Lottery Ticket Hypothesis*, we aim to find the winning tickets that achieve not only competitive performance but also robustness against adversarial attacks.

Extending the *Strong Lottery Ticket Hypothesis* to the adversarial cases is meaningful, because adversarial robustness is shown to require larger model capacity [98, 166], finding the robust "winning tickets" will compress and robustify the models at the same time. In this regard, our method follows a fundamentally different philosophy from typical adversarial training: instead of using adversarial examples to search for the optimal model parameters, we search for a robust network structure by pruning a randomly-initialized network. Intrinsically, the resulting learned models are lightweight and robust.

Mathematically, instead of learning the model parameters θ in adversarial training, our methods learn the binary masks $\mathbf{m}$ representing the architectures of the subnetworks. Correspondingly, given the pruning rate r, the dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$ and the adversarial budget $\mathcal{S}_\epsilon^{(p)}$, we search for $\mathbf{m}$ that solves the following optimization problem:

$$\begin{aligned} &\min_{\mathbf{m}} \frac{1}{N} \sum_{i=1}^{N} \max_{\Delta_i \in \mathcal{S}_\epsilon^{(p)}} \mathcal{L}\left(f(\theta \odot \mathbf{m}, \mathbf{x}_i + \Delta_i), y_i\right) \\ &s.t.\ \mathbf{m} \in \{0,1\}^n, \mathrm{sum}(\mathbf{m}) = (1-r)n. \end{aligned} \tag{5.1}$$

Here, n is the number of parameters and sum indicates the function that calculates the summation of all the elements in a vector. In contrast to adversarial training, we do not optimize the model parameters θ in (5.1); instead θ represents the randomly-initialized parameters that are kept fixed during optimization.

Since the mask $\mathbf{m}$ is a discrete vector, it cannot be directly optimized by gradient-based methods. To overcome this, we replace it with a continuous "score" variable, $\mathbf{s} \in \mathbb{R}^n$, from which we calculate the mask as

$$\mathbf{m} = \mathrm{mask}(\mathbf{s}, r)\,, \tag{5.2}$$

where $mask$ is a binarization function, which constructs a binary mask from the continuous-

valued scores $\mathbf{s}$ based on the pruning strategy and the required pruning rate r. Specifically, the pruning strategy first determines the number of parameters retained in each layer. Then, for a layer assigned m post-pruned parameters, we retain the parameters with the top m highest scores and prune the rest.

To update the scores $\mathbf{s}$, we use the same *edge-popup* strategy as in [113], and we first generate adversarial perturbations of the input using PGD as in [46]. The strategy works by exploiting the $mask$ function to construct the binary mask in the forward pass, while treating $mask$ as the identity function in the backward pass. This is called *straight-through estimator* [11] and allows the gradient to pass through and update all elements in the scores $\mathbf{s}$, although we only use a subnetwork in the forward pass. Note that we need the gradient of the score $\frac{\partial \mathcal{L}}{\partial \mathbf{s}}$ only in the outer minimization of (5.1). Therefore, the approximation of the function $mask$ in the back-propagation does not affect solving the inner maximization in (5.1), which utilizes $\frac{\partial \mathcal{L}}{\partial \Delta}$ instead. In practice, we can effectively generate adversarial examples by PGD.

We provide the pseudo-code of the edge pop-up algorithm for adversarial robustness as Algorithm 5.1, where we use PGD to generate adversarial attacks. $\Pi_{\mathcal{S}_\epsilon^{(p)}}$ mean projection into the adversarial budget $\mathcal{S}_\epsilon^{(p)}$.

Algorithm 5.1: Edge pop-up algorithm for adversarial robustness.

Input: training set $\mathcal{D}$, batch size B, PGD step size α and iteration number T, adversarial budget $\mathcal{S}_\epsilon^{(p)}$, pruning rate r, mask function $mask$, the optimizer.
Random initialize the model parameters θ and the scores $\mathbf{s}$.
for Sample a mini-batch $\{\mathbf{x}_i, y_i\}_{i=1}^{B} \sim \mathcal{D}$ **do**
  for i = 1, 2, ..., B **do**
    Sample a random noise δ within the adversarial budget $\mathcal{S}_\epsilon^{(p)}$.
    $\mathbf{x}_i^{(0)} = \mathbf{x}_i + \delta$
    for j = 1, 2, ..., T **do**
      $\mathbf{x}_i^{(j)} = \mathbf{x}_i^{(j-1)} + \alpha \nabla_{\mathbf{x}_i^{(j-1)}} \mathcal{L}(f(\theta \odot \text{mask}(\mathbf{s}, r), \mathbf{x}_i^{(j-1)}), y_i)$
      $\mathbf{x}_i^{(j)} = \mathbf{x}_i + \Pi_{\mathcal{S}_\epsilon^{(p)}} \left(\mathbf{x}_i^{(j)} - \mathbf{x}_i \right)$
    end for
  end for
  Calculate the gradient $\mathbf{g} = \frac{1}{B} \sum_{i=1}^{B} \nabla_{\mathbf{s}} \mathcal{L}(f(\theta \odot \text{mask}(\mathbf{s}, r), \mathbf{x}_i^{(T)}), y_i)$
  Update the score $\mathbf{s}$ using the optimizer.
end for
Output: the pruning mask: mask($\mathbf{s}, r$).

5.1.2 Adaptive Pruning

As shown in (5.1) in the previous section, how the function $mask$ works, i.e. the pruning strategy, can greatly affect the performance and stability of the edge-popup method, particularly in the case of adversarial training, as adversarial training solves a more challenging optimization problem than training on clean samples [93]. For the edge-popup method in the

non-adversarial cases, [113] uses the fixed pruning rate for each layer. In this adversarial cases, we found this pruning strategy does not work well when the pruning rate r is high. To tackle this issue, we introduce an *adaptive pruning* strategy that relies on an adaptive rate for layers of different sizes. We will later show the adaptive pruning strategy is a necessary ingredient to achieve competitive performance in the context of adversarial attacks.

Let use consider an L-layer neural network with n_1, n_2, ..., n_L trainable parameters, and we retain m_1, m_2, ..., m_L correspondingly after pruning. Therefore, the *fixed pruning rate* strategy means $1-r=\frac{m_1}{n_1}=\frac{m_2}{n_2}=...=\frac{m_L}{n_L}$. Since for each layer i, we select m_i trainable parameters out of n_i, the total number of combinations $\Pi_{i=1}^{L}\binom{n_i}{m_i}$ is the size of the search space for sub-networks. The following theorem indicates that the *fixed pruning rate* strategy is the strategy which approximates the maximization of the total number of combinations.

Theorem 5.1 *Consider an L-layer neural network with n_1, n_2, ..., n_L parameters in each layer, we retain m_1, m_2, ..., m_L parameters after pruning. Given a predefined pruning rate $r=1-\frac{\sum_{i=0}^{L}m_i}{\sum_{i=0}^{L}n_i}$, the optimal numbers of post-pruning parameters $\{m_i\}_{i=1}^{L}$ that maximizing the total number of combinations $\Pi_{i=1}^{L}\binom{n_i}{m_i}$ satisfy the following inequality:*

$$\forall 1\leq j,k\leq L,\ \left|\frac{m_j}{n_j}-\frac{m_k}{n_k}\right|<\frac{1}{n_j}+\frac{1}{n_k} \tag{5.3}$$

Specifically, we let n_k in (5.3) be the largest layer in the network without the loss of generality, we then have the following inequality:

$$\forall 1\leq j\leq k\leq L, j\neq k,\left|m_j-\frac{m_k}{n_k}n_j\right|<\frac{n_j}{n_k}+1\leq 2 \tag{5.4}$$

m_j is the number of retained parameters and thus an integer, so Theorem 5.1 indicates the pruning rate of each layer is close to each other when we aim to maximize the total number of combinations. That is to say, the *fixed pruning rate* strategy, i.e., $1-r=\frac{m_1}{n_1}=\frac{m_2}{n_2}=...=\frac{m_L}{n_L}$, is an approximation to maximize the search space for sub-networks. We provide its proof as follows.

Proof: We pick arbitrary $0<j,k\leq L$ and generates two sequences $\{\widehat{m}_i\}_{i=1}^{L}$, $\{\widetilde{m}_i\}_{i=1}^{L}$ as follows:

$$\begin{aligned}&\widehat{m}_j=m_j-1,\widehat{m}_k=m_k+1,\widehat{m}_i=m_i\forall i\neq j,i\neq k.\\&\widetilde{m}_j=m_j+1,\widetilde{m}_k=m_k-1,\widetilde{m}_i=m_i\forall i\neq j,i\neq k.\end{aligned} \tag{5.5}$$

Consider $\{m_i\}_{i=1}^{L}$ the optimality that maximizes the combination number $\Pi_{i=1}^{L}\binom{n_i}{m_i}$. We have

the following inequality:

$$1 > \frac{\Pi_{i=1}^{L}\binom{n_i}{\widehat{m}_i}}{\Pi_{i=1}^{L}\binom{n_i}{m_i}} = \frac{m_j}{n_j - m_j + 1}\frac{n_k - m_k}{m_k + 1},\ 1 > \frac{\Pi_{i=1}^{L}\binom{n_i}{\widetilde{m}_i}}{\Pi_{i=1}^{L}\binom{n_i}{m_i}} = \frac{n_j - m_j}{m_j + 1}\frac{m_k}{n_k - m_k + 1} \tag{5.6}$$

Reorganize the inequalities above, we obtain:

$$-\left(\frac{1}{n_k} + \frac{m_k - m_j + 1}{n_j n_k}\right) < \frac{m_k}{n_k} - \frac{m_j}{n_j} < \left(\frac{1}{n_j} + \frac{m_j - m_k + 1}{n_j n_k}\right) \tag{5.7}$$

Consider $1 \leq m_j \leq n_j$ and $1 \leq m_k \leq n_k$, we have $\frac{m_k - m_j + 1}{n_j n_k} \leq \frac{1}{n_j}$ and $\frac{m_j - m_k + 1}{n_j n_k} \leq \frac{1}{n_k}$. As a result, we have the following inequality:

$$\forall\, j, k, -\left(\frac{1}{n_j} + \frac{1}{n_k}\right) < \frac{m_k}{n_k} - \frac{m_j}{n_j} < \left(\frac{1}{n_j} + \frac{1}{n_k}\right) \tag{5.8}$$

This concludes the proof. □

The fixed pruning rate strategy is intuitive, it simply applies the same pruning rate to each layer without the consideration of their topology. In practice, the number of parameters in different layers can vary widely. For example, residual networks [63] have much fewer parameters in the first and last layers than in the middle ones. Using the fixed pruning rate thus yields very few parameters after pruning within such small layers. For example, when $r = 0.99$, only 17 parameters are left after pruning for a convolutional layer with 3 input channels, 64 output channels and a kernel size of 3. Such a small number of parameters has two serious drawbacks: 1) It greatly limits the expression power of the network; 2) it makes the edge-popup algorithm less stable, because adding or removing a single parameter then has a large impact on the network's output. This instability becomes even more pronounced in the presence of adversarial samples, because the gradients of the model parameters are more scattered than when training on clean inputs [93].

To overcome these drawbacks, we study an alternative strategy aiming to maximize the total number of paths from the input to the output in the pruned network. For a feedforward network, the total number of such paths is upper bounded by $\Pi_{i=1}^{L} m_i$. Note that, for convolutional layers, we do not repeatedly count the paths of the same parameters but through different pixels of the feature maps. The following theorem demonstrates that the pruning strategy that maximizes this upper bound consists of retaining the same number of parameters in every layer, except for the layers that initially have too few parameters, for which all parameters should then be retained.

Theorem 5.2 *Consider an L-layer feedforward neural network with $n_1, n_2, ..., n_L$ parameters in its successive layers, from which we retain $m_1, m_2, ..., m_L$ parameters, respectively, after pruning. Given a predefined sparsity ratio $r = 1 - \frac{\sum_{i=1}^{L} m_i}{\sum_{i=1}^{L} n_i}$, the numbers of post-pruning parameters $\{m_i\}_{i=1}^{L}$ that maximize the upper bound of the total number of the input-output paths $\Pi_{i=1}^{L} m_i$ have the following property: $\forall 1 \leq j \leq L$, m_j satisfies either of the following two conditions:*
1) $m_j = n_j$; 2) $\forall 1 \leq k \leq L, m_j \geq m_k - 1$.

The two conditions in Theorem 5.2 mean we retain the same number of parameters for each layer except for ones totally unpruned. We refer to the corresponding pruning strategy as the *fixed number of parameters*. We prove Theorem 5.2 by contradiction.

Proof: We assume the optimal $\{m_i\}_{i=1}^{L}$ does not satisfy the proporty mentioned in Theorem 5.2. This means $\exists 1 \leq j \leq L$ such that $m_j < n_j$ and $\exists 1 \leq k \leq L, m_j < m_k - 1$. Based on this, we then construct a new sequence $\{\widehat{m}_i\}_{i=1}^{L}$ as follows:

$$\widehat{m}_j = m_j + 1; \widehat{m}_k = m_k - 1; \forall\, i \neq j, i \neq k, \widehat{m}_i = m_i. \tag{5.9}$$

We then calculate the ratio of $\Pi_{i=1}^{L} \widehat{m}_i$ and $\Pi_{i=1}^{L} m_i$:

$$\frac{\Pi_{i=1}^{L} \widehat{m}_i}{\Pi_{i=1}^{L} m_i} = \frac{(m_j + 1)(m_k - 1)}{m_j m_k} = 1 + \frac{m_k - m_j - 1}{m_j m_k} > 1 \tag{5.10}$$

The last inequality is based on the assumption $m_j < m_k - 1$. (5.10) indicates $\Pi_{i=1}^{L} \widehat{m}_i > \Pi_{i=1}^{L} m_i$, which contradicts the optimality of $\{m_i\}_{i=1}^{L}$. □

While this *fixed number of parameters* strategy addresses the problem of obtaining too small layers arising in the *fixed pruning rate* one, it suffers from overly emphasizing the influence of the small layers. That is, the smaller layers end up containing too many parameters. In the extreme case, some layers are totally unpruned when the pruning rate r is small. This is problematic in our settings, since the model parameters are random and not updated. The unpruned layers based on random parameters provide a large amount of noise in the forward process. Furthermore, this strategy significantly sacrifices the expression power of the big layers.

In other words, the two strategies discussed above are two extremes: the *fixed pruning rate* one suffers when r is big, whereas the *fixed number of parameters* one suffers when r is small. To address this, we propose a strategy in-between these two extremes. Specifically, we determine the number of parameters retained in each layer by solving the following system of equations:

$$1 - r = \frac{\sum_{i=1}^{L} m_i}{\sum_{i=1}^{L} n_i}, \frac{m_1}{n_1^{\gamma}} = \frac{m_2}{n_2^{\gamma}} = ... = \frac{m_L}{n_L^{\gamma}}, \tag{5.11}$$

where $\gamma \in [0,1]$ is a hyper-parameter controlling the trade-off between the two extreme cases. When $\gamma = 0$, the strategy (5.11) is close to the *fixed number of parameters* one. When $\gamma = 1$, the strategy becomes the *fixed pruning rate* one. By setting $0 < \gamma < 1$, we can retain a higher proportion of parameters in the smaller layers without sacrificing the big layers too much. We call this strategy *adaptive pruning*.

As discussed above, the strategy obtained with $\gamma = 1$ tends to fail with a big r, while the strategy resulting from setting $\gamma = 0$ tends to fail with a small r. This indicates that we need to assign small values of γ given a big r and big values of γ otherwise. We validate this and study the influence of γ on the results of our approach in our experiments.

5.1.3 Binary Initialization Scheme

The empirical studies of [113] demonstrate the importance of the initialization scheme on the performance of the pruned network. To this end, they suggest the *Signed Kaiming Constant* initialization: the parameters in layer i are uniformly sampled from the set $\left\{-\sqrt{\frac{2}{l_{i-1}(1-r)}}, \sqrt{\frac{2}{l_{i-1}(1-r)}}\right\}$, where l_{i-1} represents the fan-out of the previous layer. Correspondingly, the scores $\mathbf{s}$ are initialized based on a uniform distribution $U\left[-\sqrt{\frac{1}{l_{i-1}}}, \sqrt{\frac{1}{l_{i-1}}}\right]$.

The magnitude of the *Signed Kaiming Constant* initialization is carefully calculated to keep the variance of the intermediate activations stable from the input to the output. In modern deep neural networks, the convolutional layers, potentially together with activation functions, are typically followed by a batch normalization layer. In [113] and our settings, these batch normalization layers only estimate the running statistics of their inputs, they do not have trainable parameters representing affine transformations. Because of these batch normalization layers, the magnitudes of the convolutional layers do not affect the outputs of the "convolution-batch norm" blocks. Furthermore, the fully-connected layers on top of the convolutional ones are homogeneous[1] because their bias terms are always initialized to zero. The activation functions we use, such as ReLU or leaky ReLU [97], are also homogeneous. Therefore, the magnitudes of parameters in these fully-connected layers do not change the predicted labels of the model.

Based on the analysis above, we can conclude that the magnitudes of the model parameters at initialization do not change the predicted labels and thus the expression power of the network. Therefore, we propose to scale the model parameters θ in all linear layers, i.e., convolutional and fully-connected ones, so that they are all sampled from $\{-1,+1\}$. Correspondingly, the scores $\mathbf{s}$ are initialized based on a uniform distribution $[-a,a]$ where a is a factor controlling the variance. *Binary initialization* is beneficial to model compression and acceleration, since there are no longer multiplication operations in linear layers.

Without the loss of generality, we consider r_{in}-channel input feature maps of size s, the size of the convolutional kernel is c and the convolutional layer outputs r_{out} channels. Table 5.1 demonstrate the complexity in FLOP operations in both full-precision and binary, both dense

[1] We call a function f homogeneous if it satisfies $\forall x \, \forall a \in \mathbb{R}^+, \, f(ax) = af(x)$.

(unpruned) and sparse (pruned) "Convolution-BatchNorm-ReLU" blocks. Since our model includes a batch normalization layer, we report the FLOP complexity in both the training mode and the evaluation mode. Theoretically, for the RN34 models we use in this paper, binary initialization can save approximately 45% and 32% FLOP operations compared with its full precision counterpart in the training time and inference time, respectively. Since we use irregular pruning in our method, fully take advantage of this improvement needs lower-level and hardware customization.

Network		Full Precision	Binary
Forward	Training	$2(1-r)c^2s^2r_{in}r_{out}+11s^2r_{out}$	$(1-r)c^2s^2r_{in}r_{out}+11s^2r_{out}$
	Evaluation	$2(1-r)c^2s^2r_{in}r_{out}+4s^2r_{out}$	$(1-r)c^2s^2r_{in}r_{out}+4s^2r_{out}$
Backward		$4(1-r)c^2s^2r_{in}r_{out}+4s^2r_{out}+c^2r_{in}r_{out}$	$3(1-r)c^2s^2r_{in}r_{out}+4s^2r_{out}$

Table 5.1 – The complexity in FLOP operations of the sparse "Convolution-BathNorm-ReLU" block in both full precision and binary case. For the forward pass, we consider both the training mode and the evaluation mode.

Although scaling the model parameters does not affect the expression power of the network, it does change the optimization landscape of the problem (5.1), because the softmax cross-entropy function $\mathcal{L}$ used to calculate the loss objective is not homogeneous. Compared with the *Signed Kaiming Constant* method, after multiplying the parameters initialized in the last layer by $\sqrt{\frac{l_{L-1}(1-r)}{2}}$, the output logits fed to the softmax cross-entropy function are also multiplied by the same factor. In practice, $\sqrt{\frac{l_{L-1}(1-r)}{2}} \gg 1$ greatly increases the magnitude of the output logits. Large logits which are fed to the softmax cross-entropy function $\mathcal{L}$ will cause numerical instability and thus greatly worsen the optimization performance.

To address this issue, we propose to add a 1-dimensional batch normalization layer at the end of model, just before the softmax cross-entropy function. We call this the *last batch normalization* (LBN). This normalization layer can cancel out the multiplication factor applied to the weights in the last layer and thus facilitates the optimization. We consider an L-layer neural network and each layer has l_1, l_2, ..., l_L neurons. Let $\mathbf{u} \in \mathbb{R}^{l_{L-1}}$, $\mathbf{W} \in \mathbb{R}^{l_L \times l_{L-1}}$, $\mathbf{o} \in \mathbb{R}^{l_L}$ be the output of the penultimate's output, the weight matrix of the last fully-connected layer and the last layer's output, respectively. By definition $\mathbf{o} = \mathbf{W}\mathbf{u}$. In addition, we use $y \in \{1, 2, ..., l_L\}$ to denote the label of the data and omit the bias term of the last layer since it is initialized as 0 and is not updated. For the 1-dimensional batch normalization layer, we use $\mathbf{b} \in \mathbb{R}^{l_L}$ and $\mathbf{v} \in \mathbb{R}^{l_L}$ to represent the running mean and running standard deviation, respectively.

Therefore, the loss objective $\mathcal{L}_{wo}$ and its gradient of the model without the 1-dimensional batch normalization layer is:

$$\mathcal{L}_{wo} = -log\frac{e^{\mathbf{o}_y}}{\sum_{i=1}^{l_L} e^{\mathbf{o}_i}}, \ \frac{\partial \mathcal{L}_{wo}}{\partial \mathbf{o}_j} = \frac{e^{\mathbf{o}_j}}{\sum_{i=1}^{l_L} e^{\mathbf{o}_i}} - \mathbf{1}(j=y) \tag{5.12}$$

Here $\mathbf{1}$ is the indicator function. Correspondingly, the loss objective $\mathcal{L}_{wi}$ and its gradient of

the model with the 1-dimensional batch normalization layer is:

$$\mathcal{L}_{wi} = -log\frac{e^{(\mathbf{o}_y-\mathbf{b}_y)/\mathbf{v}_y}}{\sum_{i=1}^{l_L} e^{(\mathbf{o}_i-\mathbf{b}_i)/\mathbf{v}_i}},\ \frac{\partial \mathcal{L}_{wi}}{\partial \mathbf{o}_j} = \frac{1}{\mathbf{v}_j}\left(\frac{e^{(\mathbf{o}_y-\mathbf{b}_y)/\mathbf{v}_y}}{\sum_{i=1}^{l_L} e^{(\mathbf{o}_i-\mathbf{b}_i)/\mathbf{v}_i}} - \mathbf{1}(j=y)\right) \tag{5.13}$$

Now we consider the case when the model parameter $\mathbf{W}$ is multiplied by a factor $\alpha \gg 1$: $\mathbf{W}' := \alpha\mathbf{W}$ and assume the output of the penultimate layer is unchanged. Based on this, the new output of the last layer is $\mathbf{o}' = \alpha\mathbf{o}$. For the model with the normalization layer, the new statistics are $\mathbf{b}' = \alpha\mathbf{b}$ and $\mathbf{v}' = \alpha\mathbf{v}$. In this regard, we can then recalculate the gradient of the loss objectives in both cases as follows:

$$\begin{aligned}
\frac{\partial \mathcal{L}'_{wo}}{\partial \mathbf{o}'_j} &= \frac{e^{\mathbf{o}'_j}}{\sum_{i=1}^{l_L} e^{\mathbf{o}'_i}} - \mathbf{1}(j=y) = \frac{e^{\alpha\mathbf{o}_j}}{\sum_{i=1}^{l_L} e^{\alpha\mathbf{o}_i}} - \mathbf{1}(j=y) \\
\frac{\partial \mathcal{L}'_{wi}}{\partial \mathbf{o}'_j} &= \frac{1}{\mathbf{v}'_j}\left(\frac{e^{(\mathbf{o}'_y-\mathbf{b}'_y)/\mathbf{v}'_y}}{\sum_{i=1}^{l_L} e^{(\mathbf{o}'_i-\mathbf{b}'_i)/\mathbf{v}'_i}} - \mathbf{1}(j=y)\right) = \frac{1}{\alpha\mathbf{v}_j}\left(\frac{e^{(\mathbf{o}_y-\mathbf{b}_y)/\mathbf{v}_y}}{\sum_{i=1}^{l_L} e^{(\mathbf{o}_i-\mathbf{b}_i)/\mathbf{v}_i}} - \mathbf{1}(j=y)\right)
\end{aligned} \tag{5.14}$$

We first study the case without the normalization layer. The first term $\frac{e^{\alpha\mathbf{o}_j}}{\sum_{i=1}^{l_L} e^{\alpha\mathbf{o}_i}}$ of the gradient $\frac{\partial \mathcal{L}'_{wo}}{\partial \mathbf{o}'_j}$ converges to $\mathbf{1}(j = \text{argmax}_i \mathbf{o}_i)$ as increases. For correctly classified inputs, $\frac{\partial \mathcal{L}'_{wo}}{\partial \mathbf{o}'_j}$ converges to 0 exponentially with α. In addition, the gradient $\frac{\partial \mathcal{L}'_{wo}}{\partial \mathbf{u}} = \mathbf{W}'^T\frac{\partial \mathcal{L}'_{wo}}{\partial \mathbf{o}'_j} = \alpha\mathbf{W}^T\frac{\partial \mathcal{L}'_{wo}}{\partial \mathbf{o}'_j}$ also vanish with α. Since $\frac{\partial \mathcal{L}'_{wo}}{\partial \mathbf{u}}$ is backward to previous layers, it leads to gradient vanishing. For incorrectly classified inputs, $\frac{\partial \mathcal{L}'_{wo}}{\partial \mathbf{o}'_j}$ converges to $\mathbf{1}(j = \text{argmax}_i \mathbf{o}_i) - \mathbf{1}(j = y)$, which is a vector with y-th element being -1, the element corresponding to the output label being $+1$ and the rest elements being 0. In this case, the gradient backward $\frac{\partial \mathcal{L}'_{wo}}{\partial \mathbf{u}} = \alpha\mathbf{W}^T\frac{\partial \mathcal{L}'_{wo}}{\partial \mathbf{o}'_j}$ will be approximately multiplied by α, causing gradient exploding.

By contrast, in the case of the model with the normalization layer, $\frac{\partial \mathcal{L}'_{wi}}{\partial \mathbf{o}'_j} = \frac{1}{\alpha}\frac{\partial \mathcal{L}_{wi}}{\partial \mathbf{o}_j}$. The factor $\frac{1}{\alpha}$ is canceled out when we calculate $\frac{\partial \mathcal{L}'_{wi}}{\partial \mathbf{u}} = \mathbf{W}'^T\frac{\partial \mathcal{L}'_{wi}}{\partial \mathbf{o}} = \mathbf{W}^T\frac{\partial \mathcal{L}_{wi}}{\partial \mathbf{o}}$. This means the gradient backward remains unchanged if we use the 1-dimensional batch normalization layer, which maintains the stability of training if we scale the model parameters.

In addition to the gradient of the intermediate activations, the gradient of the score $\mathbf{s}$ in (5.1) also remains unchanged on the network with the last batch normalization layer. By contrast, the gradient of the score $\mathbf{s}$ will vanish or explode without the last batch normalization layer. Since the score variable $\mathbf{s}$ is initialized based on a distribution of zero mean and a fixed predefined variance a, unstable gradient will make the performance sensitive to the choice of a and thus the hyper-parameter tuning more difficult.

In our experiments in the following section, we show that the last batch normalization layer can greatly improve the performance of both the *Signed Kaiming Constant* method and our

Binary Initialization one. Furthermore, it also makes the performance more robust to different score **s** initializations.

5.1.4 Experimental Results

In this section, we present extensive experimental results to validate our approach. First, we describe an ablation study and sensitivity analysis to evaluate the performance of our proposed methods. Then, we compare our performance with existing works, which achieve robustness and compression in either full-precision or binary cases. We also include adversarial training [98] as a baseline. Finally, we analyze the structure of the pruned networks that we obtain. We show some interesting patterns of these post-pruning networks, suggesting the potential of our approach for effective compression.

Unless explicitly stated otherwise, we use a 34-layer Residual Network (RN34) [63], it is the same as the one in [113, 128] and has 21265088 trainable parameters. [2] The bias terms of all linear layers are initialized 0, and are thus disabled. We also disable the learnable affine parameters in batch normalization layers, following the setup of [113].

We use the CIFAR10 dataset [83] in the ablation study; we also use the CIFAR100 dataset [83] and ImageNet100 [35] in the comparison with the baselines. All three datasets contain colored images, the resolution is 32×32 for CIFAR10, CIFAR100 and 224×224 for ImageNet100. For CIFAR10 and CIFAR100 datasets, we train the models for 400 epochs and use a cosine annealing learning rate scheduler with an initial value of 0.1 for the SGD optimizer with a weight decay factor equaling to 5×10^{-4}. For ImageNet100 dataset, we train the models for 100 epochs instead. We employ PGD attacks [98] to generate adversarial examples during training, but we use AutoAttack (AA) [33] for our robustness evaluation. While PGD is faster than AutoAttack and thus suitable for training, AutoAttack is the current state-of-the-art attack method, and we thus consider it a more reliable metric of robustness. We use an l_∞ norm-based adversarial budget, and the perturbation strength ϵ is $8/255$ for CIFAR10, $4/255$ for CIFAR100 and $2/255$ for ImageNet100. Finally, considering adversarial overfitting [118], we use a validation set consisting of 2% of the training data to select the best model during training.

Ablation Study and Sensitivity Analysis

Pruning Strategy and Pruning Rates We compare the performance of our method under different pruning rates r and *adaptive pruning strategies* with different values of γ. We focus on binary initialization and networks with the last batch normalization. The scores **s** are initialized from a uniform distribution $U[-0.01, 0.01]$.

Our results are summarized in Table 5.2, in which we include 7 different values of the pruning rate r and 7 different values of γ in the *adaptive pruning strategy*. First, we notice that the

[2] Note that the RN34 used in these papers and ours differs from the WideRN34-10 used in [98, 161], which is larger and has almost twice the number of trainable parameters.

best performance is achieved when $r = 0.99$ and $\gamma = 0.1$. $r = 0.8$ performs the best under the *fixed pruning rate* strategy ($\gamma = 1$). Compared with the vanilla (e.g., non-adversarial) case in [113], which uses the *fixed pruning rate* strategy and shows that $r = 0.5$ achieves the best clean accuracy, the best performance for robust accuracy is achieved at a much higher pruning rate. This interesting observation is consistent with the existing work [29], which shows that adversarial training implicitly encourages sparse convolutional kernels.

Prune Strategy	$r = 0.5$	$r = 0.8$	$r = 0.9$	$r = 0.95$	$r = 0.99$	$r = 0.995$	$r = 0.998$
$\gamma = 0.0$	2.16	6.86	23.01	41.61	44.60	40.70	**34.97**
$\gamma = 0.1$	4.35	15.03	28.12	42.65	**44.88**	**40.97**	33.09
$\gamma = 0.2$	8.01	19.21	27.99	43.72	42.92	40.52	32.99
$\gamma = 0.5$	9.21	32.70	42.84	43.62	42.45	40.55	30.08
$\gamma = 0.8$	28.90	41.51	**43.64**	**43.88**	39.12	33.61	28.07
$\gamma = 0.9$	39.09	41.71	43.07	42.28	38.68	33.89	17.43
$\gamma = 1.0$	**42.85**	**43.23**	42.13	41.12	34.57	26.67	20.56

Table 5.2 – Robust accuracy (in %) on the CIFAR10 test set under different pruning rates r and values of γ in *adaptive pruning*. The best result for each pruning rate is marked in bold.

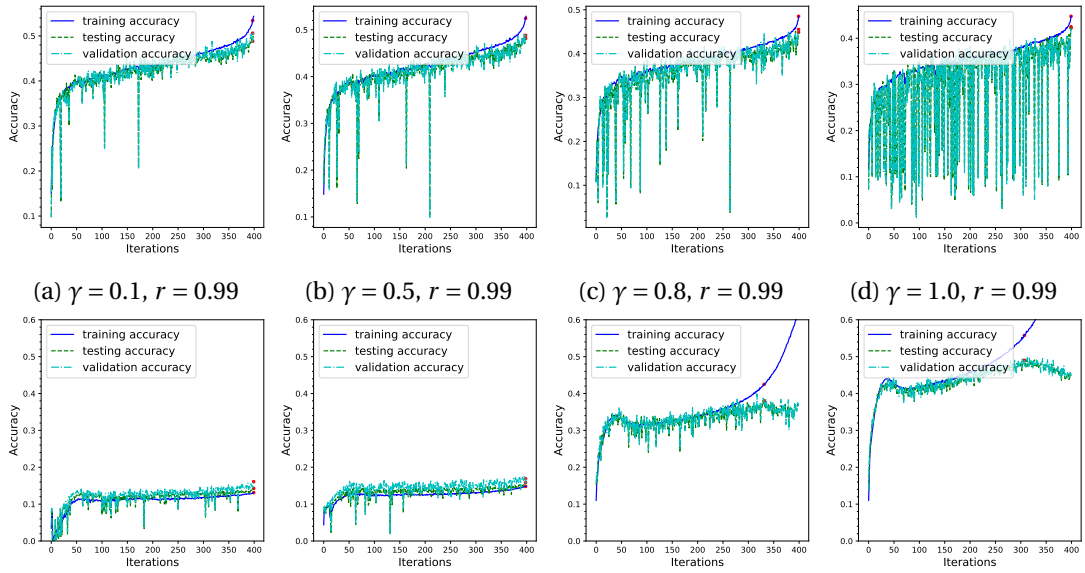
Table 5.2 further demonstrates the benefits of our proposed *adaptive pruning strategy*. For larger pruning rates r, a smaller value of γ prevails; for smaller pruning rates, a bigger value of γ prevails. This is consistent with our analysis in Section 5.1.2. In particular, compared with the best results for a fixed pruning rate strategy ($\gamma = 1.0$, $r = 0.8$), which is the pruning strategy in [113], our best adaptive pruning ($\gamma = 0.1$, $r = 0.99$) achieves not only better performance but also a higher pruning rate. That is to say, using our *adaptive pruning strategy* improves both robustness and compression rates.

In Figure 5.1, we provide the learning curves of the experiments in Table 5.2 when $r = 0.99$ and when $r = 0.5$. Regardless of the pruning rate r, these curves indicate the importance of the pruning strategy: a well chosen γ value not only improves the performance but also makes training more stable.

Initialization Scheme and Last Batch Normalization We now compare different initialization schemes and how the last batch normalization layer affects the performance. We focus on the *binary initialization* first and report the performance of models with and without the last normalization layer under different values of a, the hyper-parameter controlling the variance of the initial score $\mathbf{s}$. Based on the results of Table 5.2, we use the *adaptive pruning strategy* with $\gamma = 0.1$ and the prune rate $r = 0.99$.

The results are listed in Table 5.3 and clearly show that the last batch normalization layer (LBN) greatly improves the performance. Furthermore, LBN makes performance much less sensitive to the initialization of the scores, which in practice facilities the hyper-parameter selection.

We then compare the performance of the *binary initialization* with the *Signed Kaiming Constant*. We fix the pruning rate to $r = 0.99$ and employ an adaptive pruning strategy with different values of γ. Our results are summarized in Table 5.4. For binary initialization, we use



(a) $\gamma = 0.1$, $r = 0.99$ (b) $\gamma = 0.5$, $r = 0.99$ (c) $\gamma = 0.8$, $r = 0.99$ (d) $\gamma = 1.0$, $r = 0.99$

(e) $\gamma = 0.1$, $r = 0.5$ (f) $\gamma = 0.5$, $r = 0.5$ (g) $\gamma = 0.8$, $r = 0.5$ (h) $\gamma = 1.0$, $r = 0.5$

Figure 5.1 – Learning curves of our proposed method under adaptive pruning strategy with different values of pruning ratio r and pruning strategies represented by γ.

Model	Value of a in score initialization			
	0.001	0.01	0.1	1
no LBN	33.08	39.96	**41.01**	31.04
LBN	**45.06**	44.88	44.63	44.41

Table 5.3 – Robust accuracy (in %) on the CIFAR10 test set for models with and without the last batch normalization layer (LBN) under different values of a for score $\mathbf{s}$ initialization. The best result for each model architecture is marked in bold.

the optimal initialization scheme of the score $\mathbf{s}$ from Table 5.3; for *Signed Kaiming Constant* initialization, we use the optimal setting from [113] to initialize $\mathbf{s}$.

Prune Strategy	Signed Kaiming Constant		Binary Initialization	
	no LBN	LBN	no LBN	LBN
$\gamma = 0.0$	39.38	42.83	40.94	44.65
$\gamma = 0.1$	39.62	45.01	**41.01**	**45.06**
$\gamma = 0.2$	36.66	**45.04**	37.85	41.58
$\gamma = 0.5$	**39.98**	42.64	40.61	39.95
$\gamma = 0.8$	37.96	41.71	35.15	38.95
$\gamma = 0.9$	34.75	40.14	35.64	35.81
$\gamma = 1.0$	36.88	39.32	30.02	30.62

Table 5.4 – Robust accuracy (in %) on the CIFAR10 test set with the *Signed Kaiming Constant* and the binary initialization. We include models both with and without the last batch normalization layer (LBN). The best results are marked in bold.

Based on the results in Table 5.4, we can conclude that the *binary initialization* achieves a comparable performance with the *Signed Kaiming Constant*. In addition, the last batch normalization layer can also benefit performance when using the *Signed Kaiming Constant*. In Table 5.5 below, we demonstrate that these conclusions also hold in the non-adversarial cases, i.e., $\epsilon = 0$. This indicate the broad applicability of our methods.

Prune Strategy	Kaiming Constant Initialization		Binary Initialization	
	no LBN	LBN	no LBN	LBN
$\gamma = 0.0$	93.25	93.99	93.64	**94.05**
$\gamma = 0.1$	92.12	93.98	**93.84**	93.99
$\gamma = 0.2$	92.96	**94.35**	89.27	93.87
$\gamma = 0.5$	**93.44**	94.29	90.85	94.00
$\gamma = 0.8$	90.93	92.57	90.37	92.42
$\gamma = 0.9$	91.31	92.26	90.51	90.12
$\gamma = 1.0$	89.27	89.12	87.58	89.03

Table 5.5 – The accuracy (in %) of vanilla trained models on the CIFAR10 test set under various settings. The best result for each setting is marked in bold and the second best is underlined.

Comparison with Existing Methods

CIFAR10 and CIFAR100

In this section, we compare our approach with the state-of-the-art methods targeting model compression and robustness. Specifically, we include FlyingBird, FlyingBird+[22], Bayesian Connectivity Sampling (BCS) [107], Robust Scratch Ticket (RST) [46], HYDRA [128] and ATMC [58], as well as adversarial training (AT) [98] with early stopping [118]. Given our previous results, we fix the pruning rate to $r = 0.99$. For adversarial training, we use the full RN34 model and some smaller networks with approximately the same number of parameters as our pruned models. These smaller networks have the same architecture as the RN34 except that they have fewer channels. Table 5.6 demonstrates the architecture details of these networks. We follow the official implementations of all the baselines, and thus, unlike in our method, the normalization layers in all the baselines that update model parameters have an affine transformation with trainable parameters.

ATMC supports quantization but its parameterization introduces learnable quantized values. That is, the models obtained by ATMC's 1-bit quantization have only two parameter values in each layer; these values are different from layer to layer and are not necessarily -1 and $+1$. This means that, compared with the binary networks obtained with our method, those from ATMC have more flexibility. Nevertheless, we still include ATMC for comparison in the case of binary networks. Similarly to our method, RST does not update the model parameters. It initializes the model parameters with full-precision values, and we thus only provide full-precision results for RST. The other baselines and AT are not designed for quantization and do not inherently support binary networks. To address this, we use *BinaryConnect* [28] to

Layer Name	RN34	Small RN34-γ0.1	Small RN34-γ1.0
Conv1	3×3, 64	3×3, 23	3×3, 6
Block1	$\begin{bmatrix}3\times3,64\\3\times3,64\end{bmatrix}\times3$	$\begin{bmatrix}3\times3,23\\3\times3,23\end{bmatrix}\times3$	$\begin{bmatrix}3\times3,6\\3\times3,6\end{bmatrix}\times3$
Block2	$\begin{bmatrix}3\times3,128\\3\times3,128\end{bmatrix}\times4$	$\begin{bmatrix}3\times3,25\\3\times3,25\end{bmatrix}\times4$	$\begin{bmatrix}3\times3,13\\3\times3,13\end{bmatrix}\times4$
Block3	$\begin{bmatrix}3\times3,256\\3\times3,256\end{bmatrix}\times6$	$\begin{bmatrix}3\times3,27\\3\times3,27\end{bmatrix}\times6$	$\begin{bmatrix}3\times3,26\\3\times3,26\end{bmatrix}\times6$
Block4	$\begin{bmatrix}3\times3,512\\3\times3,512\end{bmatrix}\times3$	$\begin{bmatrix}3\times3,29\\3\times3,29\end{bmatrix}\times3$	$\begin{bmatrix}3\times3,51\\3\times3,51\end{bmatrix}\times3$
global average pool, fully connected layer to the output, softmax			
#Params	21265088	201078	216360

Table 5.6 – RN34 and its variants that have similar layer sizes as ones obtained by different γ values. $3\times3,23$ means the kernel size is 3×3 and there are 23 output channels.

replace the model's linear layers so that their parameters are binary. *BinaryConnect* generates binarized model parameters by taking the sign of the weights during the forward pass, and uses straight-through estimation [11] for gradient calculation.

Our method uses *binary initialization* and the last batch normalization layer, so the models we obtained are inherently binary. In addition to using PGD-based adversarial examples, we accelerate our method by using adversarial examples based on FGSM [**?**] with ATTA [182]. FGSM with ATTA generates adversarial examples by one-step attacks with accumulated perturbations across epochs. This is much cheaper than the 10-step PGD attacks.

Our main results on CIFAR10 and CIFAR100 are summarized in Table 5.7, where we report the robust accuracy under AutoAttack (AA). Our method using the *adaptive pruning* strategy ($p = 0.1$) achieves better performance than all baselines in case of binary models. We also achieves comparable performance to methods that aim to compress full-precision robust models. Furthermore, our method achieves results comparable with AT on the original unpruned models that has 100× more trainable parameters. In addition, our method based on accelerated adversarial training also achieves better performance than all baselines in the case of binary networks.

In Table 5.7, we report the results of all baselines using their default settings in architecture and pruning strategy based on publicly available codes. [3] The exceptions are that we also include adaptive pruning ($p = 0.1$) for HYDRA, ATMC, and the last batch normalization layer for RST, because we noticed such changes to improve their performance. For all baselines except RST, the last normalization layer does not improve the performance; it even hurts the performance in the full-precision cases. This is because these baselines (except RST) update

[3] Publicly available code on GitHub: FlyingBird/FlyingBird+: VITA-Group/Sparsity-Win-Robust-Generalization BCS: IGITUGraz/SparseAdversarialTraining RST: RICE-EIC/Robust-Scratch-Ticket HYDRA: inspire-group/hydra ATMC: VITA-Group/ATMC All the codes are free to use for non-commerical purposes.

Method	Architecture	Pruning Strategy	CIFAR10		CIFAR100	
			FP	Binary	FP	Binary
AT	RN34	Not Pruned	43.26	40.34	<u>36.63</u>	26.49
AT	RN34-LBN	Not Pruned	42.39	39.58	35.15	32.98
AT	Small RN34	Not Pruned	38.81	26.03	27.68	15.85
FlyingBird	RN34	Dynamic	<u>45.86</u>	34.37	35.91	23.32
FlyingBird+	RN34	Dynamic	44.57	33.33	34.30	22.64
BCS	RN34	Dynamic	43.51	-	31.85	-
RST	RN34	$p = 1.0$	34.95	-	21.96	-
RST	RN34-LBN	$p = 1.0$	37.23	-	23.14	-
HYDRA	RN34	$p = 0.1$	42.73	29.28	33.00	23.60
ATMC	RN34	Global	34.14	25.62	25.10	11.09
ATMC	RN34	$p = 0.1$	34.58	24.62	25.37	11.04
Ours	RN34-LBN	$p = 0.1$	-	**45.06**	-	**34.83**
Ours(fast)	RN34-LBN	$p = 0.1$	-	40.77	-	34.45

Table 5.7 – Robust accuracy (in %) on the CIFAR10 and CIFAR100 test sets for the baselines and our proposed method. "RN34-LBN" represents ResNet34 with the last batch normalization layer. "Small RN34" refers to Small RN34-p0.1 in Table 5.6 The pruning rate is set to 0.99 except for the not-pruned methods. The best results for the full-precision (FP) models are underlined; the best results for the binary models are marked in bold.

the model parameters $\mathbf{w}$. In the full precision cases, the magnitude of $\mathbf{w}$, and thus of the output logits, is automatically adjusted during training. The issue resulting from large output logits that we pointed out in Section 5.1.3 does thus not happen in these cases, so the last batch normalization layer is not necessary. In practice, we observed this layer to slow down the training convergence of these models. For the pruning strategy, the proposed *adaptive pruning* strategy ($p = 0.1$) consistently achieves better performance than the *fix pruning rate* strategy ($p = 1.0$) and than *global pruning*. FlyingBird, FlyingBird+ and BCS dynamically assign retrained parameters during training, which has similar benefits to adaptive pruning but at the cost of training efficiency [22].

To more comprehensively compare the baseline methods with our methods, we further study the baseline methods with additional settings such as adding the last batch normalization layer, changing the pruning strategy, using different AT methods, and provide a complete set of comparison results in Table 5.8. First, the last batch normalization layer (LBN) does not improve the baselines that update model parameters in the full-precision setting, because the magnitude of the output logits can be automatically adjusted in these cases. There is no need to insert another normalization layer. For FlyingBird(+), BCS and HYDRA, adding LBN to a binary network will most likely be beneficial to a better performance. This observation is consistent with our claim in Section 5.1.3. As for ATMC, it is actually not pruning a truly binary network since the value of model parameters are trainable and not necessarily $+1$ or -1, so adding LBN might not be useful in this case. For the pruning strategy, *adaptive pruning* strategy with $p = 0.1$ always has better performance than the *fixed pruning rate* strategy, i.e., $p = 1.0$. This is because the pruning rate here is very high $r = 0.99$, and we need a small value of p based

on the analysis in Section 5.1.2. Furthermore, we provide the performance of TRADES [178], which trades clean accuracy for adversarial accuracy. Compared with adversarial training (AT), TRADES achieves competitive performance in the full precision cases, but it performance degrades significantly in the binary cases.

Method	Architecture	Pruning Strategy	CIFAR10		CIFAR100	
			FP	Binary	FP	Binary
AT	RN34	Not Pruned	43.26	40.34	36.63	26.49
AT	RN34-LBN	Not Pruned	42.39	39.58	35.15	32.98
TRADES	RN34	Not Pruned	49.07	30.18	35.28	29.64
TRADES	RN34-LBN	Not Pruned	48.27	37.91	31.23	31.26
FlyingBird	RN34	Dynamic	<u>45.86</u>	34.37	<u>35.91</u>	22.49
FlyingBird+	RN34	Dynamic	44.57	33.33	34.30	22.64
FlyingBird	RN34-LBN	Dynamic	45.58	37.18	35.06	24.94
FlyingBird+	RN34-LBN	Dynamic	44.44	37.48	34.03	24.50
BCS	RN34	Dynamic	43.51	22.61	31.85	11.96
BCS	RN34-LBN	Dynamic	42.02	30.67	31.16	17.54
RST	RN34	$p = 1.0$	34.95	-	21.96	-
RST	RN34-LBN	$p = 1.0$	37.23	-	23.14	-
HYDRA	RN34	$p = 0.1$	42.73	29.28	33.00	23.60
HYDRA	RN34	$p = 1.0$	40.51	26.40	31.09	18.24
HYDRA	RN34-LBN	$p = 0.1$	40.55	33.99	13.63	25.53
HYDRA	RN34-LBN	$p = 1.0$	32.93	26.23	29.96	18.91
ATMC	RN34	Global	34.14	25.62	25.10	11.09
ATMC	RN34	$p = 0.1$	34.58	24.65	25.37	11.04
ATMC	RN34	$p = 1.0$	30.50	20.21	22.28	2.53
ATMC	RN34-LBN	Global	33.55	19.01	23.16	15.73
ATMC	RN34-LBN	$p = 0.1$	31.61	22.88	25.16	17.33
ATMC	RN34-LBN	$p = 1.0$	27.88	13.22	22.12	9.55
AT	Small RN34-p0.1	Not Pruned	42.01	32.54	28.46	16.18
AT	Small RN34-p1.0	Not Pruned	38.81	26.03	27.68	15.85
TRADES	Small RN34-p0.1	Not Pruned	42.60	29.92	28.44	15.25
TRADES	Small RN34-p1.0	Not Pruned	38.53	24.83	27.63	13.16
Ours	RN34-LBN	$p = 0.1$	-	**45.06**	-	**34.83**
Ours	RN34-LBN	$p = 1.0$	-	34.57	-	26.32
Ours (fast)	RN34-LBN	$p = 0.1$	-	40.77	-	34.45
Ours (fast)	RN34-LBN	$p = 1.0$	-	29.68	-	24.97

Table 5.8 – Robust accuracy (in %) on the CIFAR10 and CIFAR100 test sets for AT, HYDRA, ATMC and our proposed method. "RN34-LBN" represents RN34 with the last batch normalization layer. "Small RN34" here refers to Small RN34-p0.1 in Table 5.6. Among the compressed models, the best results for full precision (FP) models are underlined; the best results for binary models are marked in bold.

Finally, we report the vanilla accuracy of all methods in Table 5.9. We show that our proposed method also has competitive performance on clean inputs. Specifically, we achieve the best performance among all methods for binary networks. Combining the results in Table 5.7 and Table 5.9, we can conclude that our proposed method yields a better trade-off between vanilla accuracy and robust accuracy.

Method	Architecture	Pruning Strategy	CIFAR10		CIFAR100	
			FP	Binary	FP	Binary
AT	RN34	Not Pruned	80.99	74.37	61.48	47.87
AT	RN34-LBN	Not Pruned	80.96	74.17	57.73	60.08
AT	Small RN34	Not Pruned	74.76	58.69	52.77	28.81
FlyingBird	RN34	Dynamic	79.29	62.28	<u>62.12</u>	43.66
FlyingBird+	RN34	Dynamic	77.01	62.69	59.09	41.69
BCS	RN34	Dynamic	74.75	-	53.82	-
RST	RN34	$p = 1.0$	65.93	-	38.87	-
RST	RN34-LBN	$p = 1.0$	67.45	-	42.95	-
HYDRA	RN34	$p = 0.1$	75.31	62.09	55.92	45.96
ATMC	RN34	Global	<u>81.85</u>	72.97	57.15	36.39
ATMC	RN34	$p = 0.1$	81.37	73.34	59.99	32.68
Ours	RN34-LBN	$p = 0.1$	-	76.59	-	60.16
Ours(fast)	RN34-LBN	$p = 0.1$	-	**81.63**	-	**63.73**

Table 5.9 – The accuracy (in %) on the clean inputs of the models in Table 5.7. "RN34-LBN" represents RN34 with the last batch normalization layer. The best results in the full precision (FP) cases are underlined and the best results in the binary cases are marked in bold.

ImageNet100

Algorithm 5.2: Accelerated training for ImageNet100.

Input: training set $\mathcal{D}$, batch size B, FGSM step size α, adversarial budget $\mathcal{S}_\epsilon$, pruning rate r, mask function mask$(\cdot,\cdot)$, the optimizer.
Random initialize the model parameters θ and the scores $\mathbf{s}$.
Initialize the instance-to-perturbation dictionary $\mathcal{M} = \{\}$
for Sample a mini-batch $\{\mathbf{x}_i, y_i\}_{i=1}^B \sim \mathcal{D}$ **do**
  for $i = 1, 2, ..., n$ **do**
    Data augmentation $\mathbf{x}_i \leftarrow A(\mathbf{x}_i)$
    if $\mathbf{x}_i$ in $\mathcal{M}$ **then**
      Get the downsampled perturbation: $\delta_i' = A(\mathcal{M}(\mathbf{x}_i))$
      Upsample δ' to the original resolution and get δ_i.
    else
      Sample a random noise δ_i within the adversarial budget $\mathcal{S}_\epsilon$
    end if
    $\delta_i \leftarrow \delta_i + \alpha \nabla_{\delta_i} \mathcal{L}(f(\theta \odot \text{mask}(\mathbf{s}, r), \mathbf{x}_i + \delta_i), y_i)$
    $\delta_i \leftarrow \Pi_{\mathcal{S}_\epsilon} \delta_i$
    Update the dictionary by the downsampled perturbation δ_i': $\mathcal{M}(\mathbf{x}_i) = A^{-1}(\delta_i')$
  end for
end for
Calculate the gradient $\mathbf{g} = \frac{1}{B} \sum_{i=1}^B \nabla_{\mathbf{s}} \mathcal{L}(f(\theta \odot \text{mask}(\mathbf{s}, r), \mathbf{x}_i + \delta_i), y_i)$
Update the score $\mathbf{s}$ using the optimizer.
Output: the pruning mask mask$(\mathbf{s}, r)$.

Due to the high image resolution and large size of the ImageNet100 dataset, we use the accelerated training method (FGSM + ATTA) to generate adversarial examples for all baselines

and our proposed methods. In addition, due to the high resolution and large size of the ImageNet100 dataset, we need to compress the initial perturbation directory to reduce the overhead of memory consumption. Here, we choose to downsample the original perturbation to reduce its resolution for storage, and then upsample it back to the original resolution when using it as the initial perturbation. We provide the pseudo-code as Algorithm 5.2.

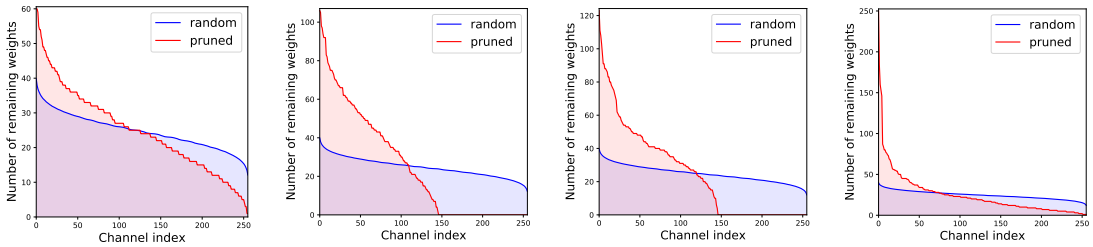
Similar to the settings in Table 5.7, we use RN34 models and the pruning rate is $r = 0.99$. The results in Table 5.10 show that our algorithm outperforms all baselines in the binary cases. Our method, which aims to train binary networks, also outperforms most full-precision networks trained by the baselines. BCS has almost trivial performance on ImageNet100 ($< 3\%$) and thus is not included in Table 5.10. This infers that BCS might not be able to converge under a high compression rate and a complicated dataset. Compared with adversarial training on the full network, which has 100 times the parameters than ours, we achieve comparable performance with the binary networks, but worse performance than the full-precision networks. It is extremely challenging to use 1% amount of the binary parameters to fit the high-dimensional ImageNet100 dataset under adversarial attacks. As we can see in Table 5.10, many baselines are only able to achieve low robust accuracy under this setting.

Method	Architecture	Pruning Strategy	ImageNet100	
			FP	Binary
AT	RN34	Not Pruned	53.92	34.20
AT	RN34-LBN	Not Pruned	<u>55.14</u>	<u>35.36</u>
AT	Small RN34	Not Pruned	25.40	10.44
FlyingBird	RN34	Dynamic	37.70	9.54
FlyingBird+	RN34	Dynamic	37.70	9.52
RST	RN34	$p = 1.0$	17.54	-
RST	RN34-LBN	$p = 1.0$	15.36	-
HYDRA	RN34	$p = 0.1$	<u>43.18</u>	18.22
ATMC	RN34	Global	22.18	5.78
ATMC	RN34	$p = 0.1$	23.52	4.58
Ours	RN34-LBN	$p = 0.1$	-	**33.04**

Table 5.10 – Robust accuracy (in %) on the ImageNet100 test sets for baselines and our proposed method. "RN34-LBN" represents ResNet34 with the last batch normalization layer. "Small RN34" here refers to Small RN34-p0.1 in Table 5.6. The prune rate is always 0.99 except for those not pruned. Among the compressed models, the best results for full precision (FP) models are underlined; the best results for binary models are marked in bold.

Analysis of the Subnetwork Pattern

In this work, we use irregular pruning. Compared with regular pruning, irregular pruning is more flexible but less structured, which means that it requires lower-level customization to fully take advantage of parameter sparsity for acceleration. However, visualizing the masks $\mathbf{m}$ of the convolutional layers in our pruned binary network with a pruning rate $r = 0.99$ allowed us to find that the mask is structured to some degree. For example, we visualize the mask of a



(a) In Channel, Layer1 (b) Out Channel, Layer1 (c) In Channel, Layer2 (d) Out Channel, Layer2

Figure 5.2 – Number of retained parameters in each input and output channel of layer1 and layer2 in a residual block. We sort the numbers and plot the curves from the largest on the left to the smallest on the right. The red curves represent the mask obtained by our method; the blue curves depict what happens when randomly pruning the corresponding layer.

convolutional layer with 256 input channels and 256 output channels in Figure 5.3 below. We notice that the retained parameters are quite concentrated: A large portion of the retained parameters concentrate on few input or output channels, while many other channels (40% of the total) are completely pruned.

Furthermore, we visualize two consecutive convolutional layers in the same residual block of the RN34 model. We call them *layer1* and *layer2* following the forward pass. In Figure 5.2, we plot the distribution of the retained parameters in each input channel and in each output channel, respectively. We find that many output channels of layer1 and the input channels of layer2, 40% of all channels in this case, are totally pruned. As a reference, we also plot the distribution of random pruning, based on the average of 500 simulations. In a randomly pruned network, it is almost impossible to have even one entirely pruned channel by probability theory. The comparison indicates that our mask $\mathbf{m}$ is structured to some degree.

Furthermore, in Figure 5.4, we visualize the pruned channels in both layers and find that they are aligned. That is, some neurons representing both the output channels of layer1 and the input channels of layer2 are entirely removed. The pattern of the structures learned by our method indicates the potential of regular pruning for a randomly-initialized neural network in the presence of adversarial attacks. We leave this as our future work.

5.2 Instance-Adaptive Fast Adversarial Training

Last section tackles the challenges of larger model capacity requirement in adversarial training. We now turn to the issue of training speed in this section: adversarial training based on PGD [98] needs much longer training time than the non-adversarial counterpart. This issue arises from the fact that PGD needs multiple iterations, i.e., multiple forward and backward passes, to generate high-quality adversarial examples and that adversarial training needs more model parameter updates to converge [93]. Therefore, to tackle this problem, we need to come up with algorithms which uses faster-generated adversarial examples and needs fewer model

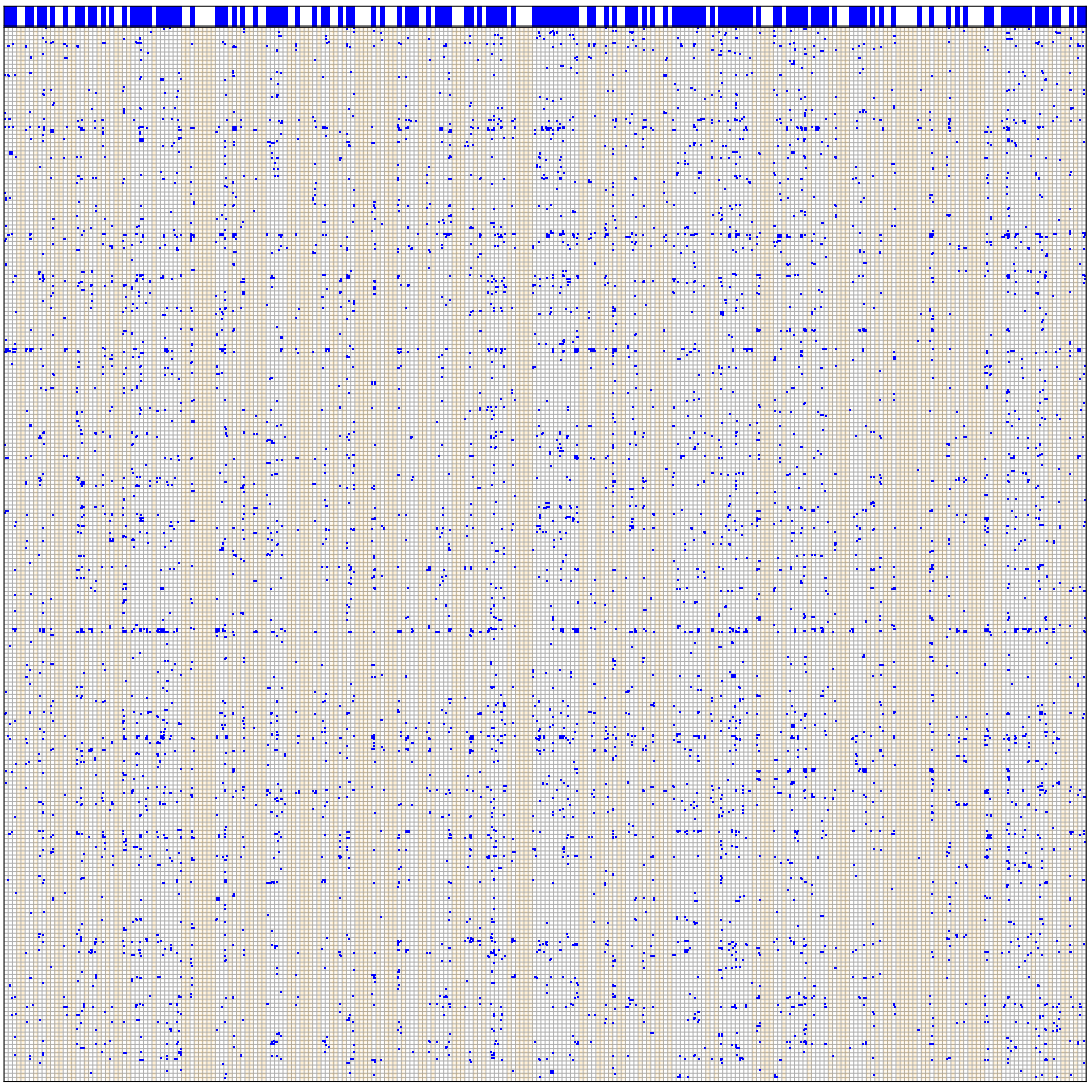


Figure 5.3 – Mask visualization of the weight of a random convolutional layer in our model. The parameters retained is highlighted as blue dots. The dimension of the convolutional kernel is $(r_{out}, r_{in}, 3, 3)$. We reshape this kernel in rectangle of shape $(r_{out} \times 3, r_{in} \times 3)$. Channels with no remaining weight are colored orange. The top bar indicates whether the channel is empty (white) or not (blue).

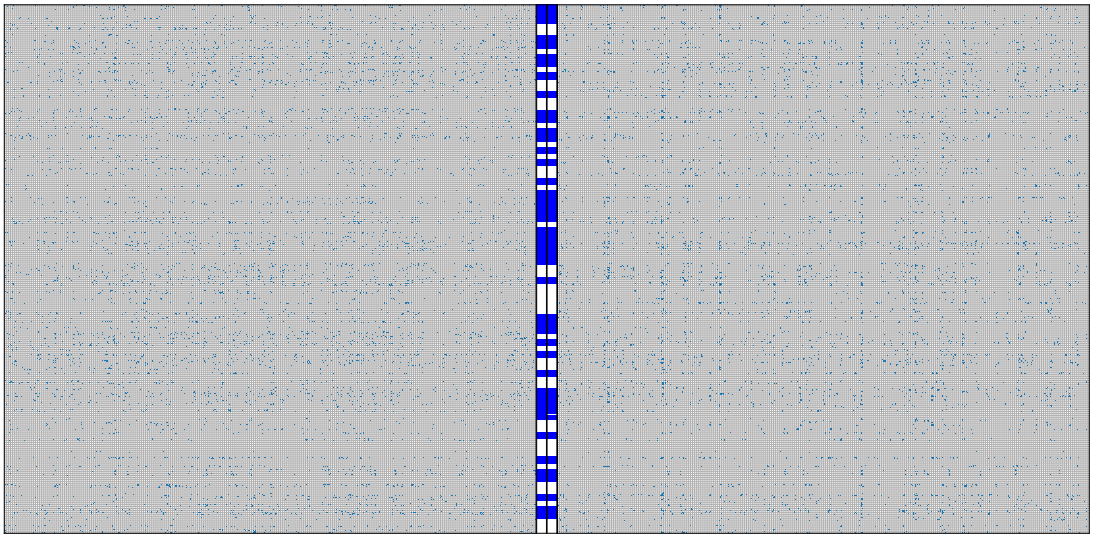


Figure 5.4 – Distribution of weights in two consecutive layers. In layer1 (left), the masks are reshaped into ($r_{out} \times 3$, $r_{in} \times 3$) while masks in layer2 (right) are reshaped into ($r_{in} \times 3$, $r_{out} \times 3$). The output channels totally pruned in layer1 and the input channels totally pruned in layer2 are highlighted as the white bars in the middle.

parameter updates to converge.

In this section, we will first review fast adversarial training algorithms, especially the problem of catastrophic overfitting. Then, we introduce the proposed method: adversarial training with adaptive step size. We theoretically justify it has faster convergence and empirically validate our claims by comprehensive experiments.

5.2.1 Fast Adversarial Training and Catastrophic Overfitting

[130] first proposes to accelerate adversarial training by *batch replaying*: to update model parameters in each iteration of PGD attack so that we can solve much fewer epochs for training. [158] introduces FGSM-RS to improve the performance. FGSM-RS uses FGSM [53] to generate adversarial examples, but from a random perturbation within the adversarial budget. Compared with FGSM, models trained by FGSM-RS can achieve robustness against stronger attackers such as PGD and AA. Compared with PGD, FGSM-RS is much cheaper and has the same complexity as FGSM or 1-step PGD.

Despite impressive performance, [158] also points out the phenomenon of *catastrophic overfitting*: for some model architectures or hyper-parameter settings, the model may overfit to the FGSM-RS and suddenly lose robustness against stronger attacks such as PGD. Figure 5.5 demonstrates the learning curves and the loss landscapes when the catastrophic overfitting happens. In addition, the model will not re-gain robustness once the catastrophic overfitting happens. Catastrophic overfitting greatly affects the stability of FGSM-RS adversarial training,

which we call *fast adversarial training* unless ambiguous.

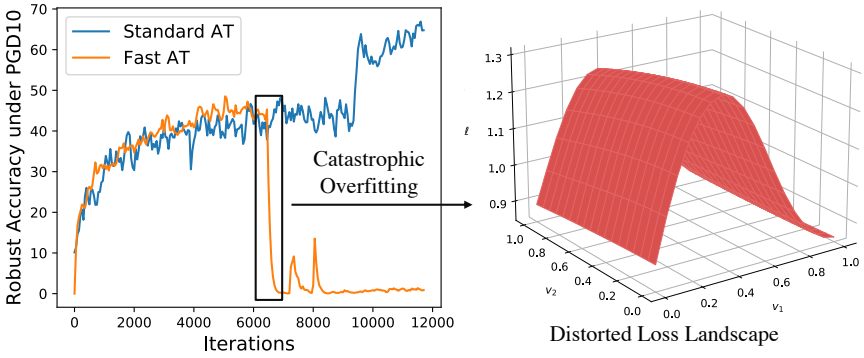


Figure 5.5 – When catastrophic overfitting occurs, the robust accuracy against PGD suddenly decreases to nearly 0%. And the loss landscape of the input becomes distorted.

To tackle the stability issue, [6] finds that it is the scattered gradients of the input that causes the catastrophic overfitting. Scattered gradients means curvy loss landscape in the input space, which makes FGSM-RS find sub-optimal perturbations. To avoid scatter gradients, [6] proposes gradient align regularization to punish scattered gradients w.r.t. the inputs. Mathematically, the regularization term is $\left\langle \frac{\partial \mathcal{L}(f(\theta,\mathbf{x}),y)}{\partial \mathbf{x}}, \frac{\partial \mathcal{L}(f(\theta,\mathbf{x}+\Delta),y)}{\partial \mathbf{x}} \right\rangle$ where Δ is a random perturbation within the adversarial budget. Although the regularization can successfully prevent the catastrophic overfitting, it considerably increases the computational complexity because of the need to calculate the second-order gradients. How to stabilize and improving accelerated adversarial training without introducing too much computational overhead is still an active problem to solve.

5.2.2 Attack by Adaptive Step Size

Similar to Section 4.3, we study the stability problem of accelerated adversarial training from the aspect of training instances. Different training instances have different loss landscapes and may result in different reasons causing catastrophic overfitting. In practice, FGSM-RS [158] uses large step size, such as $14/255$ in the case of $\epsilon = 8/255$. As a result, training against FGSM-RS perturbed training instances may result in only minimizing the classification loss for perturbed inputs near the boundary of the adversarial budget, while the loss of the inputs in the middle of the adversarial budget can be very large, which is exactly what the right half of Figure 5.5 demonstrates.

Distorted loss landscape shown in Figure 5.5 means the large magnitude of the input gradient $\frac{\partial \mathcal{L}}{\partial \mathbf{x}}$. For these training instances, we should use smaller step sizes when generating adversarial example. This will enable us to minimize the loss objective for perturbations in the middle of the adversarial budget, effectively preventing the catastrophic overfitting. However, for flat loss landscape, i.e., training instances with small magnitude of the input gradient $\frac{\partial \mathcal{L}}{\partial \mathbf{x}}$, we should use large step size instead to better approximate the optimality of the inner maximization

problem.

Based on the discussion above, we propose to use instance-adaptive step size in FGSM-RS to generate adversarial examples. Specifically, for each training instances $\mathbf{x}_i$, we maintain a pre-conditioner v_i to represent the moving average of the gradient norm:

$$v_i \longleftarrow \beta v_i + (1-\beta)\|\nabla_{\mathbf{x}}\mathcal{L}\|^2 \tag{5.15}$$

Here, β is the hyper-parameter representing the momentum. Based on this pre-conditioner, the instance-adaptive step size α_i is inversely proportional to the square root of v_i:

$$\alpha_i = \frac{\alpha}{\sqrt{v_i}+C} \tag{5.16}$$

Here, α is a unified step size before pre-conditioning, and C is a small positive constant to avoid zero-division. The step size α_i for the instance $\mathbf{x}_i$ is adjusted in each epoch.

Based on our adaptive step size method, training instance with large gradient norm will have smaller step size to avoid distorted loss landscape, which causes the catastrophic overfitting. However, using random start and small step size cannot fully explore the adversarial budget. To tackle this problem, we utilize the idea of adversarial training by transferable attacks (ATTA) [182] to initialize the perturbations by the one in the last epoch, which means we accumulatively calculate the perturbations across epochs. We call our proposed method *adversarial training with adaptive step-size* (ATAS), whose pseudo-code is provided as Algorithm 5.3 below. We use the l_∞ adversarial budget as an example, the algorithm is applicable for general l_p norm adversarial budgets.

Algorithm 5.3: ATAS algorithm in the case of l_∞ adversarial budget.

Input: Training set $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^N$, The model f parameterized by θ, loss function ℓ, adversarial budget $\mathcal{S}_\epsilon$, the unified step size α before pre-conditioning, learning rate η.
Initialize $v_i^{(0)} = 0$ for $i = 1, \cdots, N$
Initialize perturbation uniformly: $\mathbf{x}_i^{(0)} = \mathbf{x}_i + \mathrm{U}(-\epsilon, \epsilon)$ for $i = 1, \cdots, N$
for $j = 1$ to N **do**
  for $(\mathbf{x}_i, y_i) \in \mathcal{D}$ **do**
    Update the pre-conditioner: $v_i^{(j)} = \beta v_i^{(j-1)} + (1-\beta)\|\nabla_{x_i^{(j-1)}}\mathcal{L}(f(\theta, \mathbf{x}_i^{(j-1)}), y_i)\|_2^2$
    Get step size: $\alpha_i^{(j)} = \frac{\alpha}{\sqrt{v_i^{(j)}}+C}$
    Update perturbation: $\mathbf{x}_i^{(j)} = \Pi_{\mathcal{S}_\epsilon}\left(\mathbf{x}_i^{(j-1)} - \mathbf{x}_i + \alpha_i^{(j)} sign(\nabla_{\mathbf{x}_i^{(j-1)}}\mathcal{L}(f(\theta, \mathbf{x}_i^{(j-1)}), y))\right)$
    Update parameters: $\theta \longleftarrow \theta - \eta\nabla_\theta\mathcal{L}(f(\theta, \mathbf{x}_i^{(j)}), y)$.
  end for
end for
Output: Optimized model f with optimal parameter θ^*

5.2.3 Convergence Analysis

We conduct convergence analysis of the Algorithm 5.3 shown above. Specifically, we show the advantages of our method when the variance of input gradients' norm is large. We first recall the min-max problem solved by adversarial training, same as (1.2) in Chapter 1.

$$\min_{\theta} \frac{1}{N} \max_{\Delta_i \in \mathcal{S}_\epsilon} \sum_{i=1}^{N} \mathcal{L}(f(\theta, \mathbf{x}_i + \Delta_i), y_i)$$

Similar to Assumption 4.1 in Chapter 4, we assume the smoothness of the loss function $\mathcal{L}$. In addition, same as the convergence analysis of Adam [81], we study the convex case.

Assumption 5.1 *The loss function $\mathcal{L}$ has the following properties:*

1. $\mathcal{L}$ is convex and L_θ-smooth w.r.t. model parameters θ; θ and its gradient are bounded in l_2 balls:

$$\|\theta - \theta^*\|_2 \leq D_{\theta,2}, \ \ \frac{1}{N} \sum_{i=1}^{N} \|\nabla_\theta \mathcal{L}(f(\theta, \mathbf{x}_i + \Delta_i), y)\|_2^2 \leq G_{\theta,2}^2 \tag{5.17}$$

θ^* is the optimal model parameters: $\theta^* = \text{argmin}_\theta \max_{\Delta_i \in \mathcal{S}_\epsilon} \sum_{i=1}^{N} \mathcal{L}(f(\theta, \mathbf{x}_i + \Delta_i), y_i)$.

2. $\mathcal{L}$ is concave and $L_{\mathbf{x}}$-smooth w.r.t. each training instance $\mathbf{x}_i$; For any Δ_i satisfying $\|\Delta_i\|_\infty \leq D_{\mathbf{x},\infty}$ [4], we have the following bounds:

$$\|\nabla_{\Delta_i} \mathcal{L}(f(\theta, \mathbf{x}_i + \Delta_i), y_i)\|_2^2 \leq G_{\mathbf{x}_i,2}^2, \ \ \sum_{i=1}^{N} \|\nabla_{\Delta_i} \mathcal{L}(f(\theta, \mathbf{x}_i + \Delta_i), y_i)\|_2^2 \leq G_{\mathbf{x},2}^2 \tag{5.18}$$

For notation simplicity, we $\mathbf{X} \in \mathbb{R}^{N \times M}$ to represent the training data and $\Delta \in \mathbb{R}^{N \times M}$ the corresponding perturbation. In addition, we define the matrix form of the function $\mathcal{L}$ as follows, where the label y is omitted unless ambiguous:

$$\mathcal{L}(\mathbf{X}, \theta) := \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(f(\theta, \mathbf{x}_i), y_i), \ \ \max_{\Delta^* \in \mathcal{S}_\epsilon} \mathcal{L}(\mathbf{X} + \Delta^*, \theta) := \frac{1}{N} \sum_{i=1}^{N} \max_{\Delta_i \in \mathcal{S}_\epsilon} \mathcal{L}(f(\theta, \mathbf{x}_i + \Delta_i), y_i) \tag{5.19}$$

We average the trajectory of T-steps $\overline{\theta}^T = \frac{1}{T} \sum_{t=1}^{T} \theta^{(t)}$ and $\overline{\Delta}^T = \frac{1}{T} \sum_{t=1}^{T} \Delta^{(t)}$ to get the near-optimal points, where $\{\theta^{(t)}\}_{t=1}^{T}$ and $\{\Delta^{(t)}\}_{t=1}^{T}$ represent the model parameter and the adversarial perturbation trajectories, respectively. This is the standard technique used in the analysis of stochastic gradient method [42]. In this regard, the convergence gap $\max_{\Delta^* \in \mathcal{S}_\epsilon} \mathcal{L}(\mathbf{X} + \Delta^*, \overline{\theta}^T) - \max_{\Delta^* \in \mathcal{S}_\epsilon} \mathcal{L}(\mathbf{X} + \Delta^*, \theta^*)$ can be upper bounded by the regret $R(T)$ as follows:

$$R(T) = \sum_{t=1}^{T} \left[\max_{\Delta^* \in \mathcal{S}_\epsilon} \mathcal{L}(\mathbf{X} + \Delta^*, \theta^{(t)}) - \min_{\theta^*} \mathcal{L}(\mathbf{X} + \Delta^{(t)}, \theta^*) \right] \tag{5.20}$$

[4] By definition, $D_{\mathbf{x},\infty} = \epsilon$. Here, we use $D_{\mathbf{x},\infty}$ for the unified notation style.

The formal theoretical result is demonstrated as follows:

Lemma 5.1 *Consider the loss function $\mathcal{L}$ satisfying the Assumption 5.1, then:*

$$\max_{\Delta^* \in \mathcal{S}_\epsilon} \mathcal{L}(\mathbf{X}+\Delta^*, \overline{\theta}^T) - \min_{\theta^*} \max_{\Delta^* \in \mathcal{S}_\epsilon} \mathcal{L}(\overline{\theta}^*, \mathbf{X}+\Delta^*) \leq \frac{R(T)}{T} \tag{5.21}$$

Proof:

$$\begin{aligned}
&\max_{\Delta^* \in \mathcal{S}_\epsilon} \mathcal{L}(\mathbf{X}+\Delta^*, \overline{\theta}^T) - \min_{\theta^*} \max_{\Delta^* \in \mathcal{S}_\epsilon} \mathcal{L}(\mathbf{X}+\Delta^*, \theta^*) \\
\leq &\max_{\Delta^* \in \mathcal{S}_\epsilon} \mathcal{L}(\mathbf{X}+\Delta^*, \overline{\theta}^T) - \min_{\theta^*} \mathcal{L}(\mathbf{X}+\overline{\Delta}^T, \theta^*) \\
= &\max_{\Delta^* \in \mathcal{S}_\epsilon} \mathcal{L}\left(\mathbf{X}+\Delta^*, \frac{1}{T}\sum_{t=1}^{T}\theta^{(t)}\right) - \min_{\theta^*} \mathcal{L}\left(\mathbf{X}+\frac{1}{T}\sum_{t=1}^{T}\Delta^{(t)}, \theta^*\right) \\
\leq &\frac{1}{T}\left(\max_{\Delta^* \in \mathcal{S}_\epsilon} \sum_{t=1}^{T} \mathcal{L}(\mathbf{X}+\Delta^*, \theta^{(t)}) - \min_{\theta^*} \sum_{t=1}^{T} \mathcal{L}(\mathbf{X}+\Delta^{(t)}, \theta^*)\right) \\
\leq &\frac{1}{T}\sum_{t=1}^{T}\left(\max_{\Delta^* \in \mathcal{S}_\epsilon} \mathcal{L}(\mathbf{X}+\Delta^*, \theta^{(t)}) - \min_{\theta^*} \mathcal{L}(\mathbf{X}+\Delta^{(t)}, \theta^*)\right) = \frac{R(T)}{T}.
\end{aligned} \tag{5.22}$$

The first and the third inequality follows the optimality condition and the second inequality uses the Jensen inequality. □

Convergence of ATAS

ATAS can be formulated as *Adaptive Stochastic Gradient Descent Block Coordinate Ascent* (ASGDBCA), which randomly picks an instance $\mathbf{x}_k$ at the step t, applying stochastic gradient descent to the parameter θ and adaptive block coordinate ascent to the input perturbation matrix Δ. Unlike stochastic gradient descent ascent (SGDA) [92], where all dimensions of the input perturbation matrix Δ get updated in each iteration, ASGDBCA only updates some rows of Δ, corresponding to the training instances sampled in the current mini-batch. Mathematically, ASGDBCA first calculates the pre-conditioner $v_i^{(}t)$ for the training instance $\mathbf{x}_k$ as:

$$\begin{aligned}
v_i^{(t+1)} &= \begin{cases} \beta v_i^{(t)} + (1-\beta)\|\nabla_{\Delta_i} \mathcal{L}(f(\theta, \mathbf{x}_i+\Delta_i), y_i)\|_2^2 & i = k \\ v_i^{(t)} & i \neq k \end{cases} \\
\widehat{v}_i^{(t+1)} &= \max\left(\hat{v}_i^{(t)}, v_i^{(t+1)}\right).
\end{aligned} \tag{5.23}$$

Then the adversarial perturbations $\{\Delta_i\}_{i=1}^N$ and the model parameters θ are optimized by the following formula:

$$\begin{aligned}
\Delta_i^{(t+1)} &= \begin{cases} \Pi_{\mathbb{S}_\epsilon}\left[\Delta_i^{(t)} + \frac{\eta_{\mathbf{x}}}{\sqrt{\widehat{v}_i^{(t+1)}}} \nabla_{\Delta_i^{(t)}} \mathcal{L}(f(\theta^{(t)}, \mathbf{x}_i+\Delta_i^{(t)}), y_i)\right] & i = k \\ \Delta_i^{(t)} & i \neq k \end{cases} \\
\theta^{(t+1)} &= \theta^{(t)} - \eta_\theta \nabla_\theta \mathcal{L}(f(\theta^{(t)}, \mathbf{x}_k+\Delta_k^{(t+1)}), y_k).
\end{aligned}$$

The only difference of the update rule above and Algorithm 5.3 is $\widehat{v}_k^{(t)}$. This is because the pre-conditioner needs to be non-decreasing for ASGDBCA to convergence, similar to Adam in [116]. In practice, the non-convergence version works better, so ATAS still uses $v_k^{(t)}$ as the pre-conditioner.

Now, we provide the convergence analysis of ASGDBCA as the theorem below. Our theorem shows, using properly set learning rate $\eta_{\mathbf{x}}$ and η_θ for the perturbation update and model parameter update, the converge gap, which is bounded by $\frac{R(T)}{T}$, converges in the speed of $\mathcal{O}(\frac{1}{\sqrt{T}})$. This is asymptotically the same speed as SGD for the convex function.

Theorem 5.3 *Under the assumption 5.1 and the update rules (5.23), (5.2.3), if* $\eta_\theta = \frac{D_{\theta,2}}{G_{\theta,2}\sqrt{T}}$ *and* $\eta_{\mathbf{x}} = \frac{\sqrt{d}D_{\mathbf{x},\infty}}{(1-\beta)^{-1/4}\sqrt{T}}$*, the regret of ASGDBCA is bounded by the following inequality:*

$$R^{ASGDBCA}(T) \leq G_{\theta,2}D_{\theta,2}\sqrt{T} + \frac{D_{\mathbf{x},\infty}\sum_{i=1}^N G_{\mathbf{x}_i,2}\sqrt{MT}}{N(1-\beta)^{1/4}} + \frac{ML_{\mathbf{x}}D_{\mathbf{x},\infty}^2}{2N^2\sqrt{1-\beta}} \tag{5.24}$$

Without adaptive step size, the Stochastic Gradient Descent Block Coordinate Ascent (SGDBCA can be formulated as:

$$\begin{aligned}\Delta_i^{(t+1)} &= \begin{cases}\Pi_{\mathbb{S}_\epsilon}[\Delta_i^{(t)} + \eta_{\mathbf{x}}\nabla_{\Delta_i^{(t)}}\mathcal{L}(f(\theta^{(t)}, \mathbf{x}_i + \Delta_i^{(t)}), y_i)] & i = k \\ \Delta_i^{(t)} & i \neq k\end{cases} \\ \theta^{(t+1)} &= \theta^{(t)} - \eta_\theta \nabla_\theta \mathcal{L}(f(\theta^{(t)}, \mathbf{x}_k + \Delta^{(t+1)}), y_k)\,,\end{aligned} \tag{5.25}$$

This update rule represents ATTA [182] in practice. Similar to ASGDCBA, the following theorem provides the bound of the regret for SGDBCA.

Theorem 5.4 *Under the assumption 5.1 and the update rules (5.25), if* $\eta_\theta = \frac{D_{\theta,2}}{G_{\theta,2}\sqrt{T}}$ *and* $\eta_{\mathbf{x}} = \frac{\sqrt{NM}D_{\mathbf{x},\infty}}{G_{\mathbf{x},2}\sqrt{T}}$*, the regret of SGDBCA is bounded by the following inequality:*

$$R^{SGDBCA}(T) \leq G_{\theta,2}D_{\theta,2}\sqrt{T} + G_{\mathbf{x},2}D_{\mathbf{x},\infty}\sqrt{\frac{MT}{N}} + \frac{ML_{\mathbf{x}}D_{\mathbf{x},\infty}^2}{2N} \tag{5.26}$$

We compare the bounds in Theorem 5.3 and Theorem 5.4. When T is large, the third term of the regret bound in both SGDBCA and ASGDBCA is negligible. Considering that their first terms are the same, the main difference is the second term: $\frac{D_{\mathbf{x},\infty}\sum_{i=1}^N G_{\mathbf{x}_i,2}\sqrt{MT}}{N(1-\beta)^{1/4}}$ in ASGDBCA and $G_{\mathbf{x},2}D_{\mathbf{x},\infty}\sqrt{\frac{MT}{N}}$ in SGDBCA. The difference between them is:

$$\text{Ratio} = (1-\beta)^{1/4}\sqrt{\left(\frac{\sum_{i=1}^N G_{\mathbf{x}_i,2}}{N}\right)^2 \Big/ \frac{\sum_{i=1}^N G_{\mathbf{x}_i,2}^2}{N}} \tag{5.27}$$

The Cauchy-Schwarz inequality indicates the value inside the squared root is always no bigger than 1 and gets smaller when $G_{\mathbf{x}_i,2}$ has long-tailed distribution, which indicates the faster convergence of ATAS in this case. In the experimental part, we demonstrate the long-tailed distribution for common datasets.

We provide the detailed proofs for Theorem 5.3 and Theorem 5.4 below.

Proofs of Theorem 5.3 and Theorem 5.4

To facilitate the proof, we first introduce the several notations:

$$\begin{aligned} \widehat{g}_\theta(\theta,\mathbf{x}_i) &:= \nabla_\theta \mathcal{L}(f(\theta,\mathbf{x}_i),y_i), \\ g_\theta(\theta,\mathbf{X}) &:= \frac{1}{N}\sum_{i=1}^{N}\widehat{g}_\theta(\theta,\mathbf{x}_i), \\ g_\Delta(\theta,\Delta_i) &:= -\nabla_{\Delta_i}\mathcal{L}(f(\theta,\mathbf{x}_i+\Delta_i),y_i), \\ g_\Delta(\theta,\Delta) &:= [g_\Delta(\theta,\mathbf{x}_1),g_\Delta(\theta,\mathbf{x}_2),\cdots,g_\Delta(\theta,\mathbf{x}_N)] = -N\nabla_\Delta\mathcal{L}(\theta,\mathbf{X}). \end{aligned} \tag{5.28}$$

We first prove Theorem 5.4 below

Proof: Let $h_{\mathbf{x}} = \frac{\eta_{\mathbf{x}}}{\eta_\theta}$. At step t, SGDBCA picks a random instance indexed by k from $\{1,2,\cdots,N\}$, then we have the following inequality:

$$\begin{aligned} \|\Delta_k^{(t+1)} - \Delta_k^*\|_2^2 &= \|\Pi_{\mathbb{S}_\epsilon}(\Delta_k^{(t+1)} - h_{\mathbf{x}}\eta_\theta g_\Delta(\theta^{(t)},\mathbf{x}_k+\Delta_k^{(t)})) - \Delta_k^*\|_2^2 \\ &\le \|\Delta_k^{(t+1)} - h_{\mathbf{x}}\eta_\theta g_\Delta(\theta^{(t)},\mathbf{x}_k+\Delta_k^{(t)}) - \Delta_k^*\|_2^2. \end{aligned} \tag{5.29}$$

Rearrange this inequality, we obtain the following:

$$2\eta_\theta g_\Delta(\theta^{(t)},\mathbf{x}_k+\Delta_k^{(t)})^T(\Delta_k^{(t)}-\Delta_k^*) \le \frac{\|\Delta_k^{(t)}-\Delta_k^*\|_2^2 - \|\Delta_k^{(t+1)}-\Delta_k^*\|_2^2}{h_{\mathbf{x}}} + h_{\mathbf{x}}\eta_\theta^2\|g_\Delta(\theta^{(t)},\mathbf{x}_k+\Delta_k^{(t)})\|_2^2. \tag{5.30}$$

We apply the similar inequality on the model parameter updates, then have:

$$2\eta_\theta g_\Delta(\theta^{(t)},\mathbf{x}_k+\Delta_k^{(t+1)})^T(\theta^{(t)}-\theta^*) \le \|\theta^{(t)}-\theta^*\|_2^2 - \|\theta^{(t+1)}-\theta^*\|_2^2 + \eta_\theta^2\|\widehat{g}_\theta(\theta^{(t)},\mathbf{x}_k+\Delta_k^{(t+1)})\|_2^2. \tag{5.31}$$

We take the expectation over k, which is uniformly sampled from $\{1,2,\cdots,N\}$, on the left hand side of (5.30) and (5.31), we get:

$$\begin{aligned} \mathbb{E}_k\left[g_\Delta(\theta^{(t)},\mathbf{x}_k+\Delta_k^{(t)})^T(\Delta_k^{(t)}-\Delta_k^*)\right] &= \frac{1}{N}\left[g_\Delta(\theta^{(t)},\mathbf{X}+\Delta)^T(\Delta-\Delta^*)\right] \\ \mathbb{E}_k\left[g_\Delta(\theta^{(t)},\mathbf{x}_k+\Delta_k^{(t+1)})^T(\theta^{(t)}-\theta^*)\right] &= g_\theta(\theta^{(t)},\mathbf{X}+\Delta^{(t+1)})^T(\theta^{(t)}-\theta^*) \end{aligned} \tag{5.32}$$

Then, we consider the expectation of the right hand side of (5.30), we have the following equations:

$$\begin{aligned}\mathbb{E}_k\left[\|\Delta_k^{(t)}-\Delta_k^*\|_2^2-\|\Delta_k^{(t+1)}-\Delta_k^*\|_2^2\right]&=\frac{1}{N}\left[\|\Delta^{(t)}-\Delta^*\|_2^2-\|\Delta^{(t+1)}-\Delta^*\|_2^2\right]\\ \mathbb{E}_k\left[\|g_\Delta(\theta^{(t)},\mathbf{x}_k+\Delta_k^{(t)})\|_2^2\right]&=\frac{1}{N}\|g_\Delta(\theta^{(t)},\mathbf{X}+\Delta^{(t)})\|_2^2\end{aligned} \tag{5.33}$$

Consider the convexity and concavity of the loss function $\mathcal{L}$ in Assumption 5.1:

$$\begin{aligned}\nabla_\theta\mathcal{L}(\theta^{(t),\mathbf{X}+\Delta^{(t+1)}})^T(\theta^*-\theta^{(t)})&\leq\mathcal{L}(\theta^*,\mathbf{X}+\Delta^{(t+1)})-\mathcal{L}(\theta^{(t)},\mathbf{X}+\Delta^{(t+1)})\\ \mathcal{L}(\theta^{(t)},\mathbf{X}+\Delta^*)-\mathcal{L}(\theta^{(t)},\mathbf{X}+\Delta^{(t)})&\leq\nabla_\Delta\mathcal{L}(\theta^{(t)},\mathbf{X}+\Delta^{(t)})^T(\Delta^*-\Delta^{(t)})\end{aligned} \tag{5.34}$$

Re-arrange the following inequalities and combine them with (5.28), we get:

$$\begin{aligned}&g_\theta(\theta^{(t),\mathbf{X}+\Delta^{(t+1)}})^T(\theta^{(t)}-\theta^*)+\frac{1}{N}g_\Delta(\theta^{(t)},\mathbf{X}+\Delta^{(t)})^T(\Delta^*-\Delta^{(t)})\\ \geq&\mathcal{L}(\theta^{(t)},\mathbf{X}+\Delta^*)-\mathcal{L}(\theta^{(t)},\mathbf{X}+\Delta^{(t+1)})+\mathcal{L}(\theta^{(t)},\mathbf{X}+\Delta^{(t+1)})-\mathcal{L}(\theta^{(t)},\mathbf{X}+\Delta^{(t)}).\end{aligned} \tag{5.35}$$

Combining (5.30) to (5.35), we obtain the following inequality:

$$\begin{aligned}&2\eta_\theta\Big(\mathcal{L}(\theta^{(t)},\mathbf{X}+\Delta^*)-\mathcal{L}(\theta^*,\mathbf{X}+\Delta^{(t+1)})\Big)\\ &\leq\mathbb{E}_k\Big[\|\theta^{(t)}-\theta^*\|_2^2-\|\theta^{(t+1)}-\theta^*\|_2^2+\eta_\theta^2\|g_\theta(\theta^{(t)},\mathbf{x}_k+\Delta_k^{(t+1)})\|_2^2+\\ &\frac{1}{Nh_\mathbf{x}}\left(\|\Delta^{(t)}-\Delta^*\|_2^2-\|\Delta^{(t+1)}-\Delta^*\|_2^2\right)+\frac{1}{N}h_\mathbf{x}\eta_\theta^2\|g_\Delta(\theta^{(t)},\mathbf{X}+\Delta^{(t)})\|_2^2+\\ &2\eta_\theta\left(\mathcal{L}(\theta^{(t)},\mathbf{X}+\Delta^{(t)})-\mathcal{L}(\theta^{(t)},\mathbf{X}+\Delta^{(t+1)})\right)\Big]\end{aligned} \tag{5.36}$$

Consider the update rule of $\mathbf{x}_k$, and collectively $\mathbf{X}$, we have:

$$\begin{aligned}&\mathbb{E}_k[\mathcal{L}(\theta^{(t)},\mathbf{X}+\Delta^{(t)})-\mathcal{L}(\theta^{(t)},\mathbf{X}+\Delta^{(t+1)})]\\ \leq&\mathbb{E}_k\left[\frac{L_\mathbf{x}}{2N}\|\Delta_k^{(t)}-\Delta_k^{(t+1)}\|_2^2+\frac{1}{N}g_\Delta(\theta^{(t)},\mathbf{x}_k+\Delta_k^{(t)})^T(\Delta_k^{(t+1)}-\Delta_k^{(t)})\right]\\ =&\mathbb{E}_k\left[\frac{L_\mathbf{x}(h_\mathbf{x}\eta_\theta)^2}{2N}\|g_\Delta(\theta^{(t)},\mathbf{x}_k+\Delta_k^{(t)})\|_2^2-\frac{h_\mathbf{x}\eta_\theta}{N}\|g_\Delta(\theta^{(t)},\mathbf{X}+\Delta^{(t)})\|_2^2\right]\\ =&\frac{L_\mathbf{x}(h_\mathbf{x}\eta_\theta)^2}{2N^2}\|g_\Delta(\theta^{(t)},\mathbf{X}+\Delta^{(t)})\|_2^2-\frac{h_\mathbf{x}\eta_\theta}{N^2}\|g_\Delta(\theta^{(t)},\mathbf{X}+\Delta^{(t)})\|_2^2.\end{aligned} \tag{5.37}$$

The above inequality can be re-arranged as:

$$\begin{aligned}
&2\Big(\mathcal{L}(\theta^{(t)},\mathbf{X}+\Delta^*)-\mathcal{L}(\theta^*,\mathbf{X}+\Delta^{(t+1)})\Big)\\
&\qquad\leq \mathbb{E}_k\Bigg[\frac{1}{\eta_\theta}\left(\|\theta^{(t)}-\theta^*\|_2^2-\|\theta^{(t+1)}-\theta^*\|_2^2\right)+\eta_\theta\|g_\theta(\theta^{(t)},\mathbf{X}+\Delta^{(t+1)})\|_2^2+\\
&\qquad\frac{1}{Nh_{\mathbf{x}}\eta_\theta}\left(\|\Delta^{(t)}-\Delta^*\|_2^2-\|\Delta^{(t+1)}-\Delta^*\|_2^2\right)+\\
&\qquad\frac{N-2}{N^2}h_{\mathbf{x}}\eta_\theta\|g_\Delta(\theta^{(t)},\mathbf{X}+\Delta^{(t)})\|_2^2+\frac{L_{\mathbf{x}}(h_{\mathbf{x}}\eta_\theta)^2}{N^2}\|g_\Delta(\theta^{(t)},\mathbf{X}+\Delta^{(t)})\|_2^2\Bigg]
\end{aligned}\tag{5.38}$$

Summing the bound over t, the regret is then bounded by:

$$\begin{aligned}
R(T)\leq\ &\frac{1}{2\eta_\theta}\|\theta^{(1)}-\theta^*\|_2^2+\frac{1}{2N\eta_{\mathbf{x}}}\|\Delta^{(1)}-\Delta^*\|_2^2+\\
&\frac{1}{2}\mathbb{E}\left[\sum_{t=1}^{T}\eta_\theta\|g_\theta(\theta^{(t)},\mathbf{X}+\Delta^{(t+1)})\|_2^2\right]+\frac{1}{2}\mathbb{E}\left[\sum_{t=1}^{T}\frac{[(N-2)\eta_{\mathbf{x}}+L_{\mathbf{x}}\eta_{\mathbf{x}}^2]}{N^2}\|g_\Delta(\theta^{(t)},\mathbf{X}+\Delta^{(t)})\|\right]
\end{aligned}\tag{5.39}$$

Plug Assumption 5.1 and $h_{\mathbf{x}}=\frac{\eta_{\mathbf{x}}}{\eta_\theta}$ into the inequality above, we obtain:

$$R(T)\leq\frac{D_{\theta,2}^2}{2\eta_\theta}+\frac{MD_{\mathbf{x},\infty}^2}{2\eta_{\mathbf{x}}}+\frac{T\eta_\theta G_{\theta,2}^2}{2}+\frac{T\theta_{\mathbf{x}}G_{\mathbf{x},2}^2}{2N}+\frac{TL_{\mathbf{x}}\eta_{\mathbf{x}}^2G_{\mathbf{x},2}^2}{2N^2}\tag{5.40}$$

Using the inequality of arithmetic and geometric means, the optimal choice is $\eta_\theta=\frac{D_{\theta,2}}{G_{\theta,2}\sqrt{T}}$ and $\eta_{\mathbf{x}}=\frac{\sqrt{NM}D_{\mathbf{x},\infty}}{G_{\mathbf{x},2}\sqrt{T}}$, then:

$$R(T)\leq G_{\theta,2}D_{\theta,2}\sqrt{T}+G_{\mathbf{x},2}D_{\mathbf{x},\infty}\sqrt{\frac{MT}{N}}+\frac{ML_{\mathbf{x}}D_{\mathbf{x},\infty}^2}{2N}\tag{5.41}$$

□

Now we are ready to prove Theorem 5.3 as shown below. We can directly utilize some technique from the proof above.

Proof: Let $h_{\mathbf{x}}=\frac{\eta_{\mathbf{x}}}{\eta_\theta}$. At step t, ASGDABCA picks a random instance indexed by k from $\{1,2,\cdots,N\}$, then we have the following inequality:

$$\begin{aligned}
\|\Delta_k^{(t+1)}-\Delta_k^*\|_2^2&=\|\Pi_{\mathbb{S}_\epsilon}\left(\Delta_k^{(t)}-h_{\mathbf{x}}\frac{\eta_\theta}{\sqrt{\widehat{v}^{(t+1)}}}g_\Delta(\theta,\mathbf{x}_k)\right)-\Delta_k^*\|_2^2\\
&\leq\|\Delta_k^{(t)}-h_{\mathbf{x}}\frac{\eta_\theta}{\sqrt{\widehat{v}^{(t+1)}}}g_\Delta(\theta,\mathbf{x}_k)-\Delta_k^*\|_2^2
\end{aligned}\tag{5.42}$$

Re-arrange this inequality, we then obtain:

$$2g_{\Delta}(\theta^{(t)},\Delta_k^{(t)})^T(\Delta_k^{(t)}-\Delta_k^*) \leq \frac{\left\|\Delta_k^{(t)}-\Delta_k^*\right\|_2^2-\left\|\Delta_k^{(t+1)}-\Delta_k^*\right\|_2^2}{h_{\mathbf{x}}\eta_\theta}\sqrt{\widehat{v}_k^{(t+1)}}+h_{\mathbf{x}}\eta_\theta\frac{\left\|g_{\Delta}(\theta^{(t)},\Delta_k^{(t)})\right\|_2^2}{\sqrt{\widehat{v}_k^{(t+1)}}} \tag{5.43}$$

We apply the same inequality for the update rule of model parameters.

$$2g_{\theta}(\theta^{(t)},\mathbf{x}_k^{(t+1)})^T(\theta^{(t)}-\theta^*) \leq \frac{\left\|\theta^{(t)}-\theta^*\right\|_2^2-\left\|\theta^{(t+1)}-\theta^*\right\|_2^2}{\eta_\theta}+\eta_\theta\left\|g_{\theta}(\theta^{(t)},\mathbf{x}_k^{(t+1)})\right\|_2^2. \tag{5.44}$$

Let

$$\widehat{V}^{(t)} = diag([\underbrace{\widehat{v}_1^{(t)},\cdots,\widehat{v}_1^{(t)}}_{d},\underbrace{\widehat{v}_2^{(t)},\cdots,\widehat{v}_2^{(t)}}_{d},\cdots,\underbrace{\widehat{v}_N^{(t)},\cdots,\widehat{v}_N^{(t)}}_{d}]),$$

denote the pre-conditioner of all the coordinates of $\mathbf{X}$. Take the expectation over k on the right hand side, then we have the following equality:

$$\mathbb{E}_k\left[\frac{\left\|\Delta_k^{(t)}-\Delta_k^*\right\|_2^2-\left\|\Delta_k^{(t+1)}-\Delta_k^*\right\|_2^2}{h_{\mathbf{x}}}\sqrt{\widehat{v}_k^{(t+1)}}\right] = \frac{1}{N}\left[\left\|\Delta^{(t)}-\Delta^*\right\|^2_{\frac{\sqrt{\widehat{V}^{(t+1)}}}{h_{\mathbf{x}}}}-\left\|\Delta^{(t+1)}-\Delta^*\right\|^2_{\frac{\sqrt{\widehat{V}^{(t+1)}}}{h_{\mathbf{x}}}}\right], \tag{5.45}$$

and

$$\mathbb{E}_k\left[h_{\mathbf{x}}\eta_\theta^2\frac{\left\|g_{\Delta}(\theta^{(t)},\Delta_k^{(t)})\right\|_2^2}{\sqrt{\widehat{v}_k^{(t+1)}}}\right] = \frac{\eta_\theta^2}{N}\left\|g_{\Delta}(\theta^{(t)},\Delta^{(t)})\right\|^2_{\frac{h_{\mathbf{x}}}{\sqrt{\widehat{V}^{(t+1)}}}}. \tag{5.46}$$

Similar to the convergence proof for SGDBCA, we have:

$$\begin{aligned}
&\mathcal{L}(\theta^{(t)},\mathbf{X}+\Delta^*)-\mathcal{L}(\theta^*,\mathbf{X}+\Delta^{(t+1)}) \\
&\quad\leq \mathbb{E}_k\Bigg[\frac{1}{2\eta_\theta}\left(\|\Delta^{(t)}-\Delta^*\|_2^2-\|\Delta^{(t+1)}-\Delta^*\|_2^2\right)+\frac{\eta_\theta}{2}\|g_{\theta}(\theta^{(t)},\mathbf{X}+\Delta^{(t+1)})\|_2^2+ \\
&\quad\frac{1}{2Nh_{\mathbf{x}}\eta_\theta}\|\Delta^{(t)}-\Delta^*\|^2_{\sqrt{\widehat{V}^{(t+1)}}}+\frac{1}{2Nh_{\mathbf{x}}\eta_\theta}\|\Delta^{(t+1)}-\Delta^*\|^2_{\sqrt{\widehat{V}^{(t+1)}}}+ \\
&\quad\frac{N-2}{2N^2}h_{\mathbf{x}}\eta_\theta\|g_{\Delta}(\theta^{(t)},\mathbf{X}+\Delta^{(t)})\|^2_{(\widehat{V}^{(t+1)})^{-1/2}}+\frac{L_{\mathbf{x}}(h_{\mathbf{x}}\eta_\theta)^2}{2N^2}\|g_{\Delta}(\theta^{(t)},\mathbf{X}+\Delta^{(t)})\|^2_{(\widehat{V}^{(t+1)})^{-1}}\Bigg].
\end{aligned} \tag{5.47}$$

Summing the inequality from 1 to T, the regret $R(T)=R_\theta(T)+R_{\mathbf{x}}(T)$ is then upper bounded by the following inequalities where $R_\theta(T)$ represents the first two terms in the right hand side

of (5.47), $R_{\mathbf{x}}(T)$ represents the rest terms.

$$\begin{aligned} R_\theta(T) &\leq \frac{1}{2\eta_\theta}\|\theta^{(1)}-\theta^*\|_2^2+\frac{\eta_\theta}{2}\mathbb{E}\sum_{t=1}^{T}\|\widehat{g}_\theta(\theta^{(t)},\mathbf{x}_k+\Delta_k^{(t+1)})\|_2^2 \\ &\leq \frac{D_{\theta,2}^2}{2\eta_\theta}+\frac{\eta_\theta T G_{\theta,2}^2}{2} \end{aligned} \tag{5.48}$$

$$\begin{aligned} R_{\mathbf{x}}(T) \leq \sum_{t=1}^{T}\Bigg[&\frac{1}{2N\eta_{\mathbf{x}}}\left(\|\Delta^{(t)}-\Delta^*\|^2_{\sqrt{\widehat{V}^{(t+1)}}}-\|\Delta^{(t+1)}-\Delta^*\|^2_{\sqrt{\widehat{V}^{(t+1)}}}\right)+ \\ &\frac{N-2}{2N^2}\eta_{\mathbf{x}}\|g_\Delta(\theta^{(t)},\mathbf{X}+\Delta^{(t)})\|^2_{(\widehat{V}^{(t+1)})^{-1/2}}+\frac{L_{\mathbf{x}}\eta_{\mathbf{x}}^2}{2N^2}\|g_\Delta(\theta^{(t)},\mathbf{X}+\Delta^{(t)})\|^2_{(\widehat{V}^{(t+1)})^{-1}}\Bigg] \end{aligned} \tag{5.49}$$

$R_\theta(T)$ is the same as SGDBCA. Similarly, we set $\eta_\theta=\frac{D_{\theta,2}}{G_{\theta,2}\sqrt{T}}$ to minimize the right hand side of (5.48), we then have:

$$R_\theta(T)\leq G_{\theta,2}D_{\theta,2}\sqrt{T}. \tag{5.50}$$

We then focus on the upper bound of $R_{\mathbf{x}}(T)$, we the first term on the right hand side of (5.49):

$$\begin{aligned} &\sum_{t=1}^{T}\left(\|\Delta^{(t)}-\Delta^*\|^2_{\sqrt{\widehat{V}^{(t+1)}}}-\|\Delta^{(t+1)}-\Delta^*\|^2_{\sqrt{\widehat{V}^{(t+1)}}}\right) \\ &=\sum_{i=1}^{N}\left(\sum_{t=2}^{T}(\sqrt{\widehat{v}_i^{(t+1)}}-\sqrt{\widehat{v}_i^{(t)}})\|\Delta_i^{(t)}-\Delta_i^*\|_2^2+\sqrt{\widehat{v}_i^{(2)}}\|\Delta_i^{(1)}-\Delta_i^*\|_2^2\right) \\ &=\sum_{i=1}^{N}\sum_{j=1}^{M}\left(\sum_{t=2}^{T}(\sqrt{\widehat{v}_i^{(t+1)}}-\sqrt{\widehat{v}_i^{(t)}})(\Delta_{i,j}^{(t)}-\Delta_{i,j}^{(t)})^2+\sqrt{\widehat{v}_i^{(2)}}(\Delta_{i,j}^{(1)}-\Delta_{i,j}^*)^2\right). \end{aligned} \tag{5.51}$$

Here, $\Delta_{i,j}$ represents the j-th element in the vector Δ_i. Based on the Assumption 5.1, $\forall i,j,t$, $|\Delta_{i,j}^{(t)}-\Delta_{i,j}^*|<D_{\mathbf{x},\infty}$, then we have:

$$\begin{aligned} &\sum_{t=1}^{T}\left(\|\Delta^{(t)}-\Delta^*\|^2_{\sqrt{\widehat{V}^{(t+1)}}}-\|\Delta^{(t+1)}-\Delta^*\|^2_{\sqrt{\widehat{V}^{(t+1)}}}\right) \\ &\leq\sum_{i=1}^{N}\sum_{j=1}^{M}\left(\sqrt{\widehat{v}_i^{(2)}}D_{\mathbf{x},\infty}^2+\sum_{t=2}^{T}(\sqrt{\widehat{v}_i^{(t+1)}}-\sqrt{\widehat{v}_i^{(t)}})D_{\mathbf{x},\infty}^2\right) \\ &=\sum_{i=1}^{N}\sum_{j=1}^{M}\left(\sqrt{\widehat{v}^{(T+1)}}D_{\mathbf{x},\infty}^2\right) \\ &\leq\sum_{i=1}^{N}\left(MD_{\mathbf{x},\infty}^2\sqrt{\widehat{v}_i^{(T+1)}}\right) \\ &\leq MD_{\mathbf{x},\infty}^2\sum_{i=1}^{N}G_{\mathbf{x}_i,2} \end{aligned} \tag{5.52}$$

We now turn to the second term on the right hand side of (5.49).

$$\begin{aligned}\|g_\Delta(\theta^{(t)},\mathbf{X}+\Delta^{(t)})\|^2_{(\widehat{V}^{(t+1)})^{-1/2}} &= \sum_{i=1}^{N}\sum_{j=1}^{M}\frac{(g_\Delta(\theta^{(t)},\mathbf{x}_i+\Delta_i^{(t)})_j)^2}{\sqrt{\widehat{v}_i^{(t+1)}}}\\ &\le \sum_{i=1}^{N}\sum_{j=1}^{M}\frac{(g_\Delta(\theta^{(t)},\mathbf{x}_i+\Delta_i^{(t)})_j)^2}{\sqrt{1-\beta}\sqrt{\sum_{j=1}^{d}((g_\Delta(\theta^{(t)},\mathbf{x}_i+\Delta_i^{(t)}))_j)^2}}\\ &= \frac{1}{\sqrt{1-\beta}}\sum_{i=1}^{N}\|g_\Delta\|_2\\ &\le \frac{1}{\sqrt{1-\beta}}\sum_{i=1}^{N}G_{\mathbf{x}_i,2}.\end{aligned} \tag{5.53}$$

where $(g_\Delta(\theta^{(t)},\mathbf{x}_i+\Delta_i^{(t)}))_j$ represents the j-th coordinate of $g_\Delta(\theta^{(t)},\mathbf{x}_i+\Delta_i^{(t)})$. Summing over t, we can then bound this term by:

$$\sum_{t=1}^{T}\|g_\Delta(\theta^{(t)},\mathbf{X}+\Delta^{(t)})\|^2_{(\widehat{V}^{(t+1)})^{-1/2}} \le \frac{T}{\sqrt{1-\beta}}\sum_{i=1}^{N}G_{\mathbf{x}_i,2} \tag{5.54}$$

Finally, we bound the third term on the right hand side of (5.49) by:

$$\begin{aligned}\|g_\Delta(\theta^{(t)},\mathbf{X}+\Delta^{(t)})\|^2_{(\widehat{V}^{(t+1)})^{-1}} &= \sum_{i=1}^{N}\sum_{j=1}^{M}\frac{(g_\Delta(\theta^{(t)},\mathbf{x}_i+\Delta_i^{(t)})_j)^2}{\widehat{v}_i^{(t+1)}}\\ &\le \sum_{i=1}^{N}\sum_{j=1}^{M}\frac{(g_\Delta(\theta^{(t)},\mathbf{x}_i+\Delta_i^{(t)})_j)^2}{(1-\beta)\sum_{j=1}^{M}(g_\Delta(\theta^{(t)},\mathbf{x}_i+\Delta_i^{(t)}))_j)^2}\\ &= \frac{1}{1-\beta}\end{aligned} \tag{5.55}$$

Combining the bound of three components of $R_{\mathbf{x}}(T)$, we obtain:

$$R_{\mathbf{x}}(T) \le \frac{\eta_{\mathbf{x}}T}{2N\sqrt{1-\beta}}\sum_{i=1}^{N}G_{\mathbf{x}_i,2} + \frac{D_{\mathbf{x},\infty}^2 M}{2N\eta_{\mathbf{x}}}\sum_{i=1}^{N}G_{\mathbf{x}_i,2} + \frac{\eta_{\mathbf{x}}^2 L_{\mathbf{x}}T}{2N^2(1-\beta)} \tag{5.56}$$

Similar to $R_\theta(T)$, the right hand side of (5.56) is achieved when $\eta_{\mathbf{x}} = \frac{D_{\mathbf{x},\infty}\sqrt{M}(1-\beta)^{1/4}}{\sqrt{T}}$:

$$R_{\mathbf{x}}(T) \le \frac{D_{\mathbf{x},\infty}\sum_{i=1}^{N}G_{\mathbf{x}_i,2}\sqrt{TM}}{N(1-\beta)^{1/4}} + \frac{L_\infty M D_{\mathbf{x},\infty}^2}{2N^2\sqrt{1-\beta}}. \tag{5.57}$$

Combining $R_\theta(T)$ and $R_{\mathbf{x}}(T)$, the regret of ASGDBCA is bounded by:

$$R(T) \leq G_{\theta,2} D_{\theta,2} \sqrt{T} + \frac{D_{\mathbf{x},\infty} \sum_{i=1}^{N} G_{\mathbf{x}_i,2} \sqrt{TM}}{N(1-\beta)^{1/4}} + \frac{L_{\mathbf{x}} M D_{\mathbf{x},\infty}^2}{2N^2 \sqrt{1-\beta}} \tag{5.58}$$

□

5.2.4 Experimental Results

In this section, we compare our proposed ATAS (Algorithm 5.3) with the state-of-the-art accelerated adversarial training algorithms, including FreeAT [130], YOPO [177], FGSM-RS [158], FGSM-GA [6] and ATTA [182]. In addition, we also use standard adversarial training [98] using 10-iteration PGD as the reference.

For evaluation, we consider three attacks: 10-iteration PGD attack (PGD10), 50-iteration PGD attack (PGD50) and AutoAttack (AA) [33]. PGD is a white box attacks, while black-box attack is also included in AA, where Square Attack [5] is used to eliminate the effect of gradient masking.

Our proposed ATAS is based on ATTA: the adversarial perturbations are accumulated across epochs. Following previous works [158, 182], we consider the l_∞ adversarial budget $\mathbb{S}_\epsilon$. Our experiments are based on CIFAR10 and CIFAR100 [83] with the widely-used WideResNet-28-10 (WRN28) [176] and ResNet-18 (RN18) and on ImageNet [35] with ResNet-18 (RN18) and ResNet-50 (RN50).

While early stopping [118] is widely used to improve the performance of standard adversarial training [98], it is rarely used in the accelerated algorithms. This is because using PGD to evaluate the model's performance on a separate validation set is large. Besides, considering the small budget of training time in fast AT, even if early stopping is applied to terminate the training before catastrophic overfitting occurs, the training is far from convergence, resulting in poor performance [6]. As a result, we do not use early stopping here and report the performance of the checkpoints when the training stops.

For ImageNet dataset, it contains more than one million images whose resolution is relatively high (224×224). As a result, storing every details of the perturbations in ATTA and ATAS is not feasible due to the limited GPU memory. To solve this problem, we utilize the local property of the adversarial examples [75, 73] and only store the interpolated perturbations in the memory. Specifically, we compress the perturbation into 32×32 for storage in the memory and up-sample it back when using it as the initialization for the next epoch.

We train 30 epochs for CIFAR10 and CIFAR100 datasets and 90 epochs for ImageNet. For methods utilizing *batch replaying* such as FreeAT and YOPO, we keep the number of forward and backward passes the same as the other methods so that the total training time is comparable. For learning rate scheduling, we use both the piece-wise decay [182] and the cyclic

Methods	ResNet-18					WideResNet-28-10				
	Clean	PGD10	PGD50	AA	Time	Clean	PGD10	PGD50	AA	Time
PGD10	*80.13*	*50.59*	*48.94*	*45.97*	*1.23*	*85.00*	*55.51*	*53.53*	*51.27*	*8.49*
FreeAT	78.37	40.90	39.02	36.00	0.33	84.54	46.09	43.80	41.19	2.31
YOPO	74.72	37.51	35.79	33.21	0.28	82.92	44.62	42.14	40.23	1.90
FGSM-RS	**83.99**	48.99	46.36	42.95	**0.22**	80.21	0.01	0.00	0.00	1.67
FGSM-GA	80.10	49.14	47.21	43.44	0.57	75.84	45.57	43.28	39.44	3.82
ATTA	82.16	47.47	45.32	42.51	0.30	85.90	51.52	48.94	46.84	1.70
ATAS	81.22	**50.03**	**48.18**	**45.38**	0.30	**85.96**	**53.43**	**51.03**	**48.72**	**1.63**

(a) CIFAR10 with adversarial budget $\varepsilon = 8/255$.

Methods	ResNet-18					WideResNet-28-10				
	Clean	PGD10	PGD50	AA	Time	Clean	PGD10	PGD50	AA	Time
PGD10	*54.08*	*28.03*	*27.23*	*23.04*	*1.32*	*60.04*	*31.70*	*30.67*	*27.11*	*8.53*
FreeAT	50.56	19.57	18.58	15.09	0.33	59.38	24.41	23.00	19.60	2.30
YOPO	51.55	20.65	19.17	16.05	0.29	50.35	19.44	18.36	15.43	1.92
FGSM-RS	**59.35**	26.40	24.29	19.73	**0.21**	51.83	0.00	0.00	0.00	**1.60**
FGSM-GA	50.61	24.48	24.07	19.42	0.57	54.29	25.86	24.56	20.74	3.80
ATTA	57.21	25.76	24.90	21.03	0.28	**63.04**	28.93	27.18	24.42	1.63
ATAS	55.49	**27.68**	**26.60**	**22.62**	0.31	62.34	**29.89**	**28.35**	**25.03**	1.61

(b) CIFAR100 with adversarial budget $\varepsilon = 8/255$.

Methods	ResNet-18					ResNet-50				
	Clean	PGD10	PGD50	AA	Time	Clean	PGD10	PGD50	AA	Time
FreeAT	58.80	35.56	34.78	31.77	**40.01**	65.81	44.12	43.34	40.80	**108.3**
YOPO	47.69	28.50	28.10	25.22	48.22	55.68	33.46	32.19	29.56	111.8
FGSM-RS	55.26	37.33	36.98	33.28	43.46	67.83	46.12	45.56	43.58	115.0
FGSM-GA	37.01	24.15	24.05	19.98	182.7	/	/	/	/	/
ATTA	58.32	39.62	38.32	36.08	45.83	66.62	48.27	47.65	45.00	111.7
ATAS	**61.20**	**40.84**	**39.86**	**37.25**	45.70	**69.10**	**49.05**	**48.05**	**46.01**	120.4

(c) ImageNet with adversarial budget $\varepsilon = 2/255$.

Table 5.11 – Accuracy (in %) and training time (in hours) of different fast AT methods on CIFAR10, CIFAR100 and ImageNet. ATAS improves the robust accuracy under various attacks including PGD10, PGD50 and AutoAttack (AA). The method "*PGD10*" refers to the standard AT using PGD10 for the inner maximization. Note that, we do not have enough computational resources to perform standard AT on ImageNet because of computational complexity. Besides, we are unable to train the ResNet-50 on ImageNet with FGSM-GA as its memory requirement exceeds the maximum GPU memory of our devices (NVIDIA Tesla V100). For CIFAR10 and CIFAR100, the training time is evaluated on a single GPU. And we use two GPUs to train the models for ImageNet. We use default step size from the original papers for the baselines so that catastrophic overfitting seldom happens in these methods.

learning rate [158], and report the best performance. The initial learning rate is always 0.1. The piece-wise decay scheduler divides the learning rate by 10 in the 24th, 28th epoch for CIFAR10, CIFAR100, and 50th, 75th epochs for ImageNet. The weight decay is 5×10^{-4} for

CIFAR10, CIFAR100 models, and 1×10^{-4} for ImageNet models.

For FreeAT, we replays the mini-batch for 8 times on CIFAR10, CIFAR100 and 4 times for ImageNet. This means we train the model for 10 epochs on CIFAR10, CIFAR100 and 45 epochs for ImageNet. For YOPO, we use the hyper-parameter settings in YOPO-5-3 in [177] as it achieves the best performance. YOPO-5-3 trains CIFAR10 and CIFAR100 models for 12 epochs and ImageNet models for 36 epochs. We the publicly available codes [5] to reproduce the results for FGSM-RS and FGSM-GA. For different sizes of the adversarial budgets $\mathcal{S}_\epsilon$, we always set the step size α as $\alpha = 1.25\epsilon$. For the coefficient of the gradient-align regularization in FGSM-GA, we use the same as in [6] for CIFAR10. On CIFAR100 and ImageNet, we search for the optimal choices and set the coefficient as 0.5, 0.005, respectively. For ATTA, we follow the hyper-parameter settings for ATTA-1 as in [182] and set the step size $\alpha = 4/255$.

For our proposed ATAS, the hyper-parameter γ and C jointly determine the effective step size. When the gradient norm is small, we can approximate the step size by γ/C. In this regard, we set $\gamma/C = 16/255$ in all our experiments. Considering the ImageNet inputs have higher dimensionality than CIFAR10 and CIFAR100, we set higher value of C (0.1) for ImageNet models than CIFAR10, CIFAR100 models (0.01). The momentum factor β is always 0.5 for all experiments.

Our main experimental results are demonstrated in Table 5.11. The robust accuracy of FreeAT and YOPO is much lower than the other methods. While FGSM-RS maintains non-trivial robust accuracy when using RN18 and RN50, it suffers from catastrophic overfitting when using large networks such as WRN28. The regularizer in FGSM-GA prevents catastrophic overfitting. However, it may over-regularize the network so that the clean accuracy and the robust accuracy decrease on WRN28. In addition, the regularizer also brings computational overhead: FGSM-GA needs nearly double training time compared with other methods. ATAS achieves the best robust accuracy among all accelerated adversarial training algorithms while keeping the training time nearly the same. Furthermore, for small networks like RN18, the performance of ATAS is on par with standard AT (PGD10) but needs only one fifth of the training time.

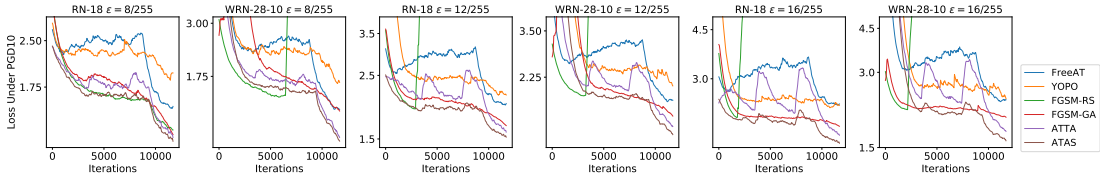


Figure 5.6 – Robust loss under PGD10 on the CIFAR10 training set under different network architectures and the adversarial budgets. The loss function is the softmax cross-entropy function. ATAS shows faster convergence in all cases.

Figure 5.6 shows the curve of the training loss function on CIFAR10, where we use PGD-

[5] FGSM-RS: https://github.com/locuslab/fast_adversarial. FGSM-GA: https://github.com/tml-epfl/understanding-fast-adv-training.

generated adversarial examples to approximate the optimal perturbed inputs and mini-batch to approximate the loss on the whole training set. It is clear that ATAS achieves the smaller robust training loss under different settings, which demonstrate the faster convergence of ATAS and is consistent with the theoretical analyses in Section 5.2.3.

5.3 Summary and Broader Impact

In this chapter, we have discussed the efficiency of robust learning, especially in the framework of adversarial training. In Section 5.1, we have extended the Strong Lottery Ticket hypothesis in the context of adversarial training. We have also proposed adaptive pruning and a binary initialization scheme for improving the performance of the sub-networks inside the randomly-initialized large network under adversarial attacks. In Section 5.2, we have proposed ATAS algorithm for stabilizing and improving the accelerated adversarial training.

Although our proposed methods are still far from ideal, they introduce additional hyper-parameters for tuning and cannot jointly solve several efficiency challenges, and they inspire some feasible directions for the improvement of the efficiency of robust learning. We can use encode adversarial robustness to the methods that intrinsically improve the efficiency in order to achieve robustness and efficiency at the same time. Our proposed methods, in Section 5.1, use pruning as a way of adversarial training to improve robustness. Note that it is important to figure out the fundamental reasons of the efficiency issue in robust learning. In Section 5.2, we have noted that training instances with large input gradients causes catastrophic overfitting, which directly inspires our proposed method. We believe that other efficiency challenges in robust learning, including those mentioned in Section 1.3, can be solved or mitigated in similar ways.

6 Conclusion

6.1 Summary

In this thesis, we have focused on the robust learning problem in the context of deep neural networks; this can be formulated as a min-max optimization problem in (1.2). Due to the non-convexity of deep neural networks, it is NP-hard to perfectly solve the inner maximization problem in (1.2). We choose to minimize alternatively either the upper bound or the lower bound of the inner maximization problem.

By minimizing the upper bound of the inner maximum, we can obtain models that are provably robust. The corresponding verified robust accuracy is the lower bound of the true robust accuracy. The key challenge here is to efficiently estimate the upper bound of the loss objective for a deep neural network, which is highly non-convex, under the given adversarial budget. In Chapter 3, we have studied this problem from the aspect of geometry. We bound, specifically, the decision boundaries, which is untrackable in general, by an envelope represented by a hyper-cube or a polyhedron, both consisting of several hyper-planes. By maximizing the envelope's volume, we developed algorithms to study the geometric properties of the decision boundary. The envelope is represented by several linear constraints. We can efficiently calculate the distance between the input instance and the envelope's boundary as the lower bound of the distance between the input instance and the decision boundary. As this distance is differentiable with respect to model parameters, we can then train the model to enlarge this distance so as to obtain provably robust deep neural networks.

In addition to verified robustness, we can obtain empirically robust models by minimizing the lower bound of the inner maximum. The corresponding empirical robust accuracy is the upper bound of the true robust accuracy. Adversarial training, i.e., training the model against adversarially perturbed examples, is the most popular framework to obtain empirically robust models. In Chapter 4, we have investigated the convergence and generalization properties of adversarial training, in comparison to classic empirical risk minimization. For convergence, we are the first, to the best of our knowledge, to point out the non-smooth nature of the adversarial loss landscape in the model parameter space; as this loss landscape leads to

scattered gradients and slow down the convergence. We have also numerically shown the sharper and more isolated local minima in adversarial training. To overcome these challenges, we have proposed a warm-up strategy in the adversarial budget and periodical scheduler to ensemble intermediate model checkpoints, which correspond to different local minima, to boost the performance. For the much more severe overfitting observed in adversarial training, we find that it is a result of the model's fitting hard adversarial training instances, i.e., instances with large loss values. Our findings are confirmed theoretically and empirically in various settings. We discovered that existing methods successfully mitigating adversarial overfitting implicitly avoid fitting hard adversarial training instances.

To achieve either verified or empirical robustness presents certain challenges, including but not limited to larger model capacity needs, larger training data needs, much larger computational complexity, and degradation in clean accuracy. In order to improve the efficiency of robust learning, in Chapter 5, we have introduced two methods, in different aspects. To mitigate the large model capacity needs for adversarial training, we extend the strong Lottery Ticket Hypothesis to the adversarial cases. We use, specifically, pruning as a means of training in order to obtain robust sub-networks inside a randomly-initialized large network, without actually updating its model parameters. Compared with the non-adversarial case, to obtain random-initialized robust sub-networks is more challenging. In this regard, we have proposed adaptive pruning, a binary initialization scheme, and have added an additional batch normalization layer on top of the network to obtain competitive performance. To accelerate adversarial training and avoid stability issues, such as catastrophic overfitting, we have proposed to use instance-adaptive step size when perturbing the input instances for training. We find that catastrophic overfitting results from using a too-large step size to perturb training instances, and this step size's input gradients have a large magnitude, which leads to sub-optimal input perturbation for training. We have introduced an Adam-style pre-conditioner to adaptively assign different step sizes for perturbing the input. Our method has been shown to eliminate catastrophic overfitting, converge faster, and to obtain better performance.

Despite many efforts devoted to robust learning, the current state-of-the-art are still far from ideal. The challenges discussed in Section 1.3 are still not satisfactorily solved. In the final section, we will discuss the key unsolved challenges and our future directions.

6.2 Unsolved Challenges and Future Work

Figure 6.1 demonstrates some key challenges of robust learning, as well as some related fields and directions for future explorations. Among these points, the algorithm complexity and model capacity needs for robustness are the most extensively studied fields. Although there are still some detailed open problems, such as structured pruning while maintaining robustness (mentioned in Section 5.1) , we mainly discuss the other relatively underexplored points in Figure 6.1.

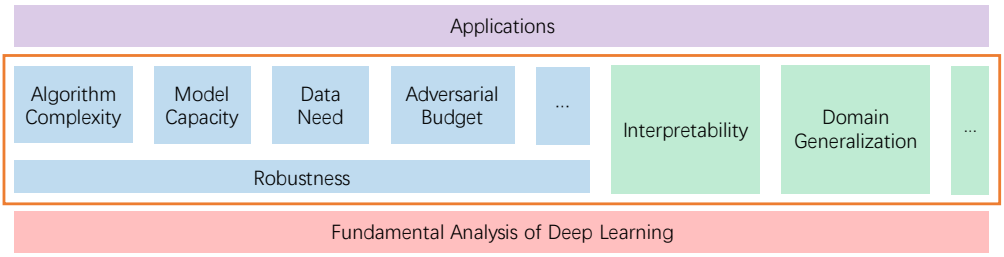


Figure 6.1 – The diagrams showing the related fields and directions for exploration in the future, from theoretical understandings to practical applications. The points inside the box are closely related to reliable machine learning.

Data Needs The sample complexity required by adversarial robustness is much higher than that of classic empirical risk minimization [126]. This means that using more training instances might improve the robustness performance. As of today, there are mainly three ways of obtaining more training instances, but they all have limitations or disadvantages. First, we can use external data, even unlabeled [18, 56], to boost adversarial training. However, it is usually difficult or expensive to obtain sufficient data that follows the same distribution as the training set. External data with a too-large distributional shift can even undermine adversarial training. Second, to generate more training data, we can train a generative model by using the training set. For example, to improve the performance, [57] generates, based on CIFAR10 training set, one million synthetic data. This category of methods can be applied only when we have a relatively large "original" training set to train a good generator. Training a generator also introduces considerable computational overhead. Third, we can use traditional data augmentation techniques to diversify the training data. [114] uses spatial transformation, especially CutMix [175], to achieve the state-of-the-art performance on CIFAR10. This technique, however, works only with model weight averaging and needs careful tuning. It is not clear if the method in [114] can be generalized to other kinds of datasets. In summary, there are still some open problems unsolved in this field, such as how to boost, without introducing too much computational overhead, adversarial training by more data, and how to improve, based on a very small training set, the model robustness.

Adversarial Budget The methods we have introduced in this thesis are focused on the l_p norm-based adversarial budget $\mathbb{S}_\epsilon^{(p)}$, which is a probably an oversimplified formulation for facilitating mathematical analyses. Ideally, the adversarial budget should include all the perturbations that do not change the semantic meaning of the input. However, it is very difficult to have a unified mathematical formulation of "semantically unchanged". Nevertheless, there are several works that explore robustness beyond l_p norm. For example, [64] studies 15 different types of common corruptions such as zoom blur, Gaussian noise, and defocus blur. [117] bridges the gap between the worst-case robustness and average robustness against random perturbations. Furthermore, [156] uses the conditional variational autoencoder [139] to learn a parameterized adversarial budget, based on a measure of its quality to capture the semantic meaning of the input instances. Although recent work [82] has shown the effectiveness of l_p

norm-based adversarial training against other common corruptions, the current notion of the adversarial budget and the corresponding training methods are still far from the needs of some applications in the real life.

Jointly Solving Multiple Challenges In Section 1.3, we list several challenges of robust learning. We notice that solving or mitigating one challenge can result in worsening the issue of a different aspect. For example, to decrease the training data needs, we can generate synthetic data to improve robustness. This, however, introduces significant computational overhead. Another example is the learning-based adversarial budget [156] described above, it enriches the definition of the adversarial budget and solves a more practical robustness problem. Nevertheless, we need to optimize the adversarial budget in each mini-batch update; this algorithm converges more slowly hence needs more training time than the l_p counterpart. In summary, jointly tackling several issues of robust learning would be even more challenging, although there are not any theorems demonstrating the strict trade-offs between them.

Interpretability and Domain Generalization To better understand deep neural networks and to build reliable machine learning models, there are also problems other than adversarial robustness. Here, we discuss two examples: model interpretability and domain generalization.

Figure 3.5 in Section 3.2 indicates that robust models tend to use input features that are more aligned with human perception hence that are more easily interpretable. [145] visualize the gradient of the loss objective with respect to the input. The gradient maps of the non-robust models are mostly noise, whereas the gradient maps of robust models clearly reveals the edges and shapes of the object in the input images. Both phenomena indicate that robustness implicitly improves model interpretability. We believe methods that explicitly improve the interpretability of the model also have other benefits, such as robustness.

In all the problems studied in this thesis, we focus on robustness on the instance level. In other words, the adversarial perturbation is based on a specific instances, and perturbations for different instances are independent of each other. Compared with instance-wise perturbations, a domain shift is more common in real-life applications. The domain shift can arise from a systematic sampling error, such as samples from different sources or errors introduced by different sampling equipment. Domain generalization has been studied from different perspectives [184]. Similarly to distributional robustness optimization (DRO) [138], it would be interesting to study how to extend our methods against instance-wise perturbations to a domain shift.

Fundamental Analysis of Deep Learning & Related Applications To rigorously understand the robustness of deep neural networks, we need theoretical guarantees. The network properties related to robustness includes the Lipschitz constant [16], input neighborhood [129] and decision boundaries. Further investigation about how to derive tighter bounds under a more general settings would be worthwhile.

Finally, integrating adversarial robustness into some specific applications is highly non-trivial,

because different applications define different training objectives and different adversarial budgets. Pioneering explorations include reinforcement learning [51] and medical imaging [109].

Bibliography

[1] Wieland Brendel *, Jonas Rauber *, and Matthias Bethge. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. In International Conference on Learning Representations, 2018.

[2] Abdullah Al-Dujaili and Una-May O'Reilly. Sign bits are all you need for black-box attacks. In International Conference on Learning Representations, 2020.

[3] Jean-Baptiste Alayrac, Jonathan Uesato, Po-Sen Huang, Alhussein Fawzi, Robert Stanforth, and Pushmeet Kohli. Are labels required for improving adversarial robustness? In Advances in Neural Information Processing Systems, pages 12192–12202, 2019.

[4] Moustafa Alzantot, Yash Sharma, Ahmed Elgohary, Bo-Jhang Ho, Mani B. Srivastava, and Kai-Wei Chang. Generating natural language adversarial examples. In EMNLP, pages 2890–2896, 2018.

[5] Maksym Andriushchenko, Francesco Croce, Nicolas Flammarion, and Matthias Hein. Square attack: a query-efficient black-box adversarial attack via random search. In European Conference on Computer Vision, pages 484–501. Springer, 2020.

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

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

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

[9] Mislav Balunovic and Martin Vechev. Adversarial training and provable defenses: Bridging the gap. In International Conference on Learning Representations, 2020.

[10] Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. A neural probabilistic language model. Advances in Neural Information Processing Systems, 13, 2000.

[11] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. arXiv preprint arXiv:1308.3432, 2013.

[12] Arjun Nitin Bhagoji, Warren He, Bo Li, and Dawn Song. Practical black-box attacks on deep neural networks using efficient query mechanisms. In Proceedings of the European Conference on Computer Vision (ECCV), pages 154–169, 2018.

[13] George Boole. The mathematical analysis of logic. Philosophical Library, 1847.

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

[15] Thomas Brunner, Frederik Diehl, Michael Truong Le, and Alois Knoll. Guessing smart: Biased sampling for efficient black-box adversarial attacks. In Proceedings of the IEEE/CVF International Conference on Computer Vision, pages 4958–4966, 2019.

[16] Sébastien Bubeck and Mark Sellke. A universal law of robustness via isoperimetry. Advances in Neural Information Processing Systems, 34, 2021.

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

[18] Yair Carmon, Aditi Raghunathan, Ludwig Schmidt, John C Duchi, and Percy S Liang. Unlabeled data improves adversarial robustness. In Advances in Neural Information Processing Systems, pages 11190–11201, 2019.

[19] Alvin Chan, Yi Tay, Yew Soon Ong, and Jie Fu. Jacobian adversarially regularized networks for robustness. In International Conference on Learning Representations, 2020.

[20] Jinghui Chen, Yu Cheng, Zhe Gan, Quanquan Gu, and Jingjing Liu. Efficient robust training via backward smoothing, 2021.

[21] Tianlong Chen, Zhenyu Zhang, Sijia Liu, Shiyu Chang, and Zhangyang Wang. Robust overfitting may be mitigated by properly learned smoothening. In International Conference on Learning Representations, 2021.

[22] Tianlong Chen, Zhenyu Zhang, pengjun wang, Santosh Balachandra, Haoyu Ma, Zehao Wang, and Zhangyang Wang. Sparsity winning twice: Better robust generalization from more efficient training. In International Conference on Learning Representations, 2022.

[23] Zhuotong Chen, Qianxiao Li, and Zheng Zhang. Towards robust neural networks via close-loop control. In International Conference on Learning Representations, 2021.

[24] 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.

[25] Shuyu Cheng, Yinpeng Dong, Tianyu Pang, Hang Su, and Jun Zhu. Improving black-box adversarial attacks with a transfer-based prior. Advances in neural information processing systems, 32, 2019.

[26] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). In International Conference on Learning Representations, 2016.

[27] Jeremy Cohen, Elan Rosenfeld, and Zico Kolter. Certified adversarial robustness via randomized smoothing. In International Conference on Machine Learning, pages 1310–1320. PMLR, 2019.

[28] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In Advances in neural information processing systems, pages 3123–3131, 2015.

[29] Francesco Croce, Maksym Andriushchenko, and Matthias Hein. Provable robustness of relu networks via maximization of linear regions. In the 22nd International Conference on Artificial Intelligence and Statistics, pages 2057–2066. PMLR, 2019.

[30] Francesco Croce, Maksym Andriushchenko, Vikash Sehwag, Edoardo Debenedetti, Nicolas Flammarion, Mung Chiang, Prateek Mittal, and Matthias Hein. Robustbench: a standardized adversarial robustness benchmark. In Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2), 2021.

[31] Francesco Croce, Sven Gowal, Thomas Brunner, Evan Shelhamer, Matthias Hein, and Taylan Cemgil. Evaluating the adversarial robustness of adaptive test-time defenses. arXiv preprint arXiv:2202.13711, 2022.

[32] Francesco Croce and Matthias Hein. Minimally distorted adversarial examples with a fast adaptive boundary attack. In International Conference on Machine Learning, pages 2196–2205. PMLR, 2020.

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

[34] Francesco Croce and Matthias Hein. Mind the box: l_1-apgd for sparse adversarial attacks on image classifiers. In International Conference on Machine Learning, pages 2201–2211. PMLR, 2021.

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

[36] Guneet S. Dhillon, Kamyar Azizzadenesheli, Jeremy D. Bernstein, Jean Kossaifi, Aran Khanna, Zachary C. Lipton, and Animashree Anandkumar. Stochastic activation pruning for robust adversarial defense. In International Conference on Learning Representations, 2018.

[37] Laurent Dinh, Razvan Pascanu, Samy Bengio, and Yoshua Bengio. Sharp minima can generalize for deep nets. In Proceedings of the 34th International Conference on Machine Learning-Volume 70, pages 1019–1028. JMLR. org, 2017.

[38] Yinpeng Dong, Ke Xu, Xiao Yang, Tianyu Pang, Zhijie Deng, Hang Su, and Jun Zhu. Exploring memorization in adversarial training. In International Conference on Learning Representations, 2022.

[39] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In International Conference on Learning Representations, 2021.

[40] Felix Draxler, Kambis Veschgini, Manfred Salmhofer, and Fred Hamprecht. Essentially no barriers in neural network energy landscape. In International conference on machine learning, pages 1309–1318. PMLR, 2018.

[41] Jiawei Du, Hu Zhang, Joey Tianyi Zhou, Yi Yang, and Jiashi Feng. Query-efficient meta attack to deep neural networks. In International Conference on Learning Representations, 2020.

[42] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. Journal of machine learning research, 12(7), 2011.

[43] Ecenaz Erdemir, Jeffrey Bickford, Luca Melis, and Sergul Aydore. Adversarial robustness with non-uniform perturbations. Advances in Neural Information Processing Systems, 34, 2021.

[44] Stanislav Fort and Stanislaw Jastrzebski. Large scale structure of neural network loss landscapes. Advances in Neural Information Processing Systems, 32, 2019.

[45] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In International Conference on Learning Representations, 2019.

[46] Yonggan Fu, Qixuan Yu, Yang Zhang, Shang Wu, Xu Ouyang, David Daniel Cox, and Yingyan Lin. Drawing robust scratch tickets: Subnetworks with inborn robustness are found within randomly initialized networks. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, Advances in Neural Information Processing Systems, 2021.

[47] Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry P Vetrov, and Andrew G Wilson. Loss surfaces, mode connectivity, and fast ensembling of dnns. In Advances in Neural Information Processing Systems, pages 8789–8798, 2018.

[48] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In 2018 IEEE Symposium on Security and Privacy (SP), pages 3–18. IEEE, 2018.

[49] Mario Geiger, Stefano Spigler, Stéphane d'Ascoli, Levent Sagun, Marco Baity-Jesi, Giulio Biroli, and Matthieu Wyart. Jamming transition as a paradigm to understand the loss landscape of deep neural networks. Physical Review E, 100(1):012115, 2019.

[50] Behrooz Ghorbani, Shankar Krishnan, and Ying Xiao. An investigation into neural net optimization via hessian eigenvalue density. In International Conference on Machine Learning, pages 2232–2241. PMLR, 2019.

[51] Adam Gleave, Michael Dennis, Cody Wild, Neel Kant, Sergey Levine, and Stuart Russell. Adversarial policies: Attacking deep reinforcement learning. In International Conference on Learning Representations, 2020.

[52] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the thirteenth international conference on artificial intelligence and statistics, pages 249–256, 2010.

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

[54] Akhilesh Gotmare, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. A closer look at deep learning heuristics: Learning rate restarts, warmup and distillation. In International Conference on Learning Representations, 2019.

[55] Sven Gowal, Krishnamurthy Dj Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Relja Arandjelovic, Timothy Mann, and Pushmeet Kohli. Scalable verified training for provably robust image classification. In Proceedings of the IEEE/CVF International Conference on Computer Vision, pages 4842–4851, 2019.

[56] Sven Gowal, Chongli Qin, Jonathan Uesato, Timothy Mann, and Pushmeet Kohli. Uncovering the limits of adversarial training against norm-bounded adversarial examples. arXiv preprint arXiv:2010.03593, 2020.

[57] Sven Gowal, Sylvestre-Alvise Rebuffi, Olivia Wiles, Florian Stimberg, Dan Andrei Calian, and Timothy A Mann. Improving robustness using generated data. Advances in Neural Information Processing Systems, 34, 2021.

[58] Shupeng Gui, Haotao Wang, Haichuan Yang, Chen Yu, Zhangyang Wang, and Ji Liu. Model compression with adversarial robustness: A unified optimization framework. Advances in Neural Information Processing Systems, 32, 2019.

[59] Chuan Guo, Jared S Frank, and Kilian Q Weinberger. Low frequency adversarial perturbation. In Uncertainty in Artificial Intelligence, pages 1127–1137. PMLR, 2020.

[60] Chuan Guo, Jacob Gardner, Yurong You, Andrew Gordon Wilson, and Kilian Weinberger. Simple black-box adversarial attacks. In International Conference on Machine Learning, pages 2484–2493. PMLR, 2019.

[61] Chuan Guo, Mayank Rana, Moustapha Cisse, and Laurens van der Maaten. Countering adversarial images using input transformations. In International Conference on Learning Representations, 2018.

[62] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In Proceedings of the IEEE international conference on computer vision, pages 1026–1034, 2015.

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

[64] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. In International Conference on Learning Representations, 2019.

[65] Dan Hendrycks, Kimin Lee, and Mantas Mazeika. Using pre-training can improve model robustness and uncertainty. In International Conference on Machine Learning, pages 2712–2721, 2019.

[66] Magnus R Hestenes. Multiplier and gradient methods. Journal of optimization theory and applications, 4(5):303–320, 1969.

[67] Dorjan Hitaj, Giulio Pagnotta, Iacopo Masi, and Luigi V Mancini. Evaluating the robustness of geometry-aware instance-reweighted adversarial training. arXiv preprint arXiv:2103.01914, 2021.

[68] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. In The collected works of Wassily Hoeffding, pages 409–426. Springer, 1994.

[69] Roger A Horn and Charles R Johnson. Matrix analysis. Cambridge university press, 2012.

[70] Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E Hopcroft, and Kilian Q Weinberger. Snapshot ensembles: Train 1, get m for free. In International Conference on Learning Representations, 2017.

[71] Lang Huang, Chao Zhang, and Hongyang Zhang. Self-adaptive training: beyond empirical risk minimization. Advances in neural information processing systems, 33:19365–19376, 2020.

[72] Qian Huang, Isay Katsman, Horace He, Zeqi Gu, Serge Belongie, and Ser-Nam Lim. Enhancing adversarial example transferability with an intermediate level attack. In Proceedings of the IEEE/CVF international conference on computer vision, pages 4733–4742, 2019.

[73] Zhichao Huang, Yaowei Huang, and Tong Zhang. Corrattack: Black-box adversarial attack with structured search, 2021.

[74] Andrew Ilyas, Logan Engstrom, Anish Athalye, and Jessy Lin. Black-box adversarial attacks with limited queries and information. In International Conference on Machine Learning, pages 2137–2146. PMLR, 2018.

[75] Andrew Ilyas, Logan Engstrom, and Aleksander Madry. Prior convictions: Black-box adversarial attacks with bandits and priors. In International Conference on Learning Representations, 2019.

[76] 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, pages 125–136, 2019.

[77] Ziwei Ji and Matus Telgarsky. The implicit bias of gradient descent on nonseparable data. In Conference on Learning Theory, pages 1772–1798. PMLR, 2019.

[78] Matt Jordan and Alexandros G Dimakis. Exactly computing the local lipschitz constant of relu networks. Advances in Neural Information Processing Systems, 33:7344–7353, 2020.

[79] Sekitoshi Kanai, Masanori Yamada, Hiroshi Takahashi, Yuki Yamanaka, and Yasutoshi Ida. Smoothness analysis of loss functions of adversarial training. CoRR, abs/2103.01400, 2021.

[80] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In International conference on computer aided verification, pages 97–117. Springer, 2017.

[81] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In ICLR (Poster), 2015.

[82] Klim Kireev, Maksym Andriushchenko, and Nicolas Flammarion. On the effectiveness of adversarial training against common corruptions. CoRR, abs/2103.02325, 2021.

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

[84] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems, 25, 2012.

[85] Nupur Kumari, Mayank Singh, Abhishek Sinha, Harshitha Machiraju, Balaji Krishnamurthy, and Vineeth N Balasubramanian. Harnessing the vulnerability of latent layers in adversarially trained models. In Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19, pages 2779–2785. International Joint Conferences on Artificial Intelligence Organization, 7 2019.

[86] Alexey kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. In International Conference on Learning Representations, 2017.

[87] Alexey Kurakin, Ian Goodfellow, Samy Bengio, et al. Adversarial examples in the physical world, 2016.

[88] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. Neural computation, 1(4):541–551, 1989.

[89] 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.

[90] Sungyoon Lee, Woojin Lee, Jinseong Park, and Jaewook Lee. Towards better understanding of training certifiably robust models against adversarial examples. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, Advances in Neural Information Processing Systems, 2021.

[91] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. In Advances in Neural Information Processing Systems, pages 6389–6399, 2018.

[92] Tianyi Lin, Chi Jin, and Michael Jordan. On gradient descent ascent for nonconvex-concave minimax problems. In International Conference on Machine Learning, pages 6083–6093. PMLR, 2020.

[93] Chen Liu, Mathieu Salzmann, Tao Lin, Ryota Tomioka, and Sabine Süsstrunk. On the loss landscape of adversarial training: Identifying challenges and how to overcome them. Advances in Neural Information Processing Systems, 33:21476–21487, 2020.

[94] Chen Liu, Mathieu Salzmann, and Sabine Süsstrunk. Training provably robust models by polyhedral envelope regularization. IEEE Transactions on Neural Networks and Learning Systems, 2021.

[95] Chen Liu, Ryota Tomioka, and Volkan Cevher. On certifying non-uniform bounds against adversarial attacks. In International Conference on Machine Learning, pages 4072–4081. PMLR, 2019.

[96] Chen Liu, Ziqi Zhao, Sabine Süsstrunk, and Mathieu Salzmann. Robust binary models by pruning randomly-initialized networks. arXiv preprint arXiv:2202.01341, 2022.

[97] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. Citeseer.

[98] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In International Conference on Learning Representations, 2018.

[99] Apostolos Modas, Seyed-Mohsen Moosavi-Dezfooli, and Pascal Frossard. Sparsefool: a few pixels make a big difference. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 9087–9096, 2019.

[100] Seungyong Moon, Gaon An, and Hyun Oh Song. Parsimonious black-box adversarial attacks via efficient combinatorial optimization. In International Conference on Machine Learning, pages 4636–4645. PMLR, 2019.

[101] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 1765–1773, 2017.

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

[103] Aamir Mustafa, Salman Khan, Munawar Hayat, Roland Goecke, Jianbing Shen, and Ling Shao. Adversarial defense by restricting the hidden space of deep neural networks. In Proceedings of the IEEE/CVF International Conference on Computer Vision, pages 3385–3394, 2019.

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

[105] Behnam Neyshabur, Srinadh Bhojanapalli, David McAllester, and Nati Srebro. Exploring generalization in deep learning. In Advances in Neural Information Processing Systems, pages 5947–5956, 2017.

[106] Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. In search of the real inductive bias: On the role of implicit regularization in deep learning. In ICLR (Workshop), 2015.

[107] Ozan Özdenizci and Robert Legenstein. Training adversarially robust sparse networks via bayesian connectivity sampling. In International Conference on Machine Learning, pages 8314–8324. PMLR, 2021.

[108] Tianyu Pang, Kun Xu, Yinpeng Dong, Chao Du, Ning Chen, and Jun Zhu. Rethinking softmax cross-entropy loss for adversarial robustness. In International Conference on Learning Representations, 2020.

[109] Magdalini Paschali, Sailesh Conjeti, Fernando Navarro, and Nassir Navab. Generalizability vs. robustness: investigating medical imaging networks using adversarial examples. In International Conference on Medical Image Computing and Computer-Assisted Intervention, pages 493–501. Springer, 2018.

[110] Michael JD Powell. A method for nonlinear constraints in minimization problems. Optimization, pages 283–298, 1969.

[111] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Certified defenses against adversarial examples. In International Conference on Learning Representations, 2018.

[112] Aditi Raghunathan, Jacob Steinhardt, and Percy S Liang. Semidefinite relaxations for certifying robustness to adversarial examples. Advances in Neural Information Processing Systems, 31, 2018.

[113] Vivek Ramanujan, Mitchell Wortsman, Aniruddha Kembhavi, Ali Farhadi, and Mohammad Rastegari. What's hidden in a randomly weighted neural network? In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 11893–11902, 2020.

[114] Sylvestre-Alvise Rebuffi, Sven Gowal, Dan A. Calian, Florian Stimberg, Olivia Wiles, and Timothy A. Mann. Fixing data augmentation to improve adversarial robustness. CoRR, abs/2103.01946, 2021.

[115] Sylvestre-Alvise Rebuffi, Sven Gowal, Dan Andrei Calian, Florian Stimberg, Olivia Wiles, and Timothy Mann. Data augmentation can improve robustness. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, Advances in Neural Information Processing Systems, 2021.

[116] Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. In International Conference on Learning Representations, 2018.

[117] Leslie Rice, Anna Bair, Huan Zhang, and J. Zico Kolter. Robustness between the worst and average case. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, Advances in Neural Information Processing Systems, volume 34, pages 27840–27851. Curran Associates, Inc., 2021.

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

[119] Volker Roth. Kernel fisher discriminants for outlier detection. Neural computation, 18(4):942–960, 2006.

[120] Wenjie Ruan, Xiaowei Huang, and Marta Kwiatkowska. Reachability analysis of deep neural networks with provable guarantees. In IJCAI, pages 2651–2659, 2018.

[121] Hadi Salman, Andrew Ilyas, Logan Engstrom, Ashish Kapoor, and Aleksander Madry. Do adversarially robust imagenet models transfer better? Advances in Neural Information Processing Systems, 33:3533–3545, 2020.

[122] Hadi Salman, Jerry Li, Ilya Razenshteyn, Pengchuan Zhang, Huan Zhang, Sebastien Bubeck, and Greg Yang. Provably robust deep learning via adversarially trained smoothed classifiers. Advances in Neural Information Processing Systems, 32, 2019.

[123] Hadi Salman, Greg Yang, Huan Zhang, Cho-Jui Hsieh, and Pengchuan Zhang. A convex relaxation barrier to tight robustness verification of neural networks. Advances in Neural Information Processing Systems, 32, 2019.

[124] Pouya Samangouei, Maya Kabkab, and Rama Chellappa. Defense-GAN: Protecting classifiers against adversarial attacks using generative models. In International Conference on Learning Representations, 2018.

[125] Kevin Scaman and Aladin Virmaux. Lipschitz regularity of deep neural networks: analysis and efficient estimation. In Proceedings of the 32nd International Conference on Neural Information Processing Systems, pages 3839–3848, 2018.

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

[127] Vikash Sehwag, Shiqi Wang, Prateek Mittal, and Suman Jana. Hydra: Pruning adversarially robust neural networks. Advances in Neural Information Processing Systems, 33:19655–19666, 2020.

[128] Vikash Sehwag, Shiqi Wang, Prateek Mittal, and Suman Jana. Hydra: Pruning adversarially robust neural networks. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, Advances in Neural Information Processing Systems, volume 33, pages 19655–19666. Curran Associates, Inc., 2020.

[129] Ali Shafahi, W. Ronny Huang, Christoph Studer, Soheil Feizi, and Tom Goldstein. Are adversarial examples inevitable? In International Conference on Learning Representations, 2019.

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

[131] Changhao Shi, Chester Holtz, and Gal Mishne. Online adversarial purification based on self-supervised learning. In International Conference on Learning Representations, 2021.

[132] Zhouxing Shi, Yihan Wang, Huan Zhang, Jinfeng Yi, and Cho-Jui Hsieh. Fast certified robust training with short warmup. Advances in Neural Information Processing Systems, 34, 2021.

[133] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. nature, 529(7587):484–489, 2016.

[134] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. nature, 550(7676):354–359, 2017.

[135] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014.

[136] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. Fast and effective robustness certification. Advances in neural information processing systems, 31, 2018.

[137] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. Proceedings of the ACM on Programming Languages, 3(POPL):1–30, 2019.

[138] Aman Sinha, Hongseok Namkoong, and John Duchi. Certifiable distributional robustness with principled adversarial training. In International Conference on Learning Representations, 2018.

[139] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. Advances in neural information processing systems, 28, 2015.

[140] Yang Song, Taesup Kim, Sebastian Nowozin, Stefano Ermon, and Nate Kushman. Pixeldefend: Leveraging generative models to understand and defend against adversarial examples. In International Conference on Learning Representations, 2018.

[141] Daniel Soudry, Elad Hoffer, Mor Shpigel Nacson, Suriya Gunasekar, and Nathan Srebro. The implicit bias of gradient descent on separable data. The Journal of Machine Learning Research, 19(1):2822–2878, 2018.

[142] 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.

[143] Vincent Tjeng, Kai Y. Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. In International Conference on Learning Representations, 2019.

[144] Antonio Torralba, Rob Fergus, and William T Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. IEEE transactions on pattern analysis and machine intelligence, 30(11):1958–1970, 2008.

[145] 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.

[146] Ramon Van Handel. Probability in high dimension. Technical report, PRINCETON UNIV NJ, 2014.

[147] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. Advances in neural information processing systems, 30, 2017.

[148] Roman Vershynin. High-dimensional probability: An introduction with applications in data science, volume 47. Cambridge university press, 2018.

[149] Ke Wang and Christos Thrampoulidis. Benign overfitting in binary classification of gaussian mixtures. In ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 4030–4034. IEEE, 2021.

[150] Shiqi Wang, Yizheng Chen, Ahmed Abdou, and Suman Jana. Mixtrain: Scalable training of formally robust neural networks. arXiv preprint arXiv:1811.02625, 2018.

[151] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Efficient formal safety analysis of neural networks. Advances in Neural Information Processing Systems, 31, 2018.

[152] Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. Beta-CROWN: Efficient bound propagation with per-neuron split constraints for complete and incomplete neural network verification. Advances in Neural Information Processing Systems, 34, 2021.

[153] 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, 2020.

[154] Lily Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Luca Daniel, Duane Boning, and Inderjit Dhillon. Towards fast computation of certified robustness for relu networks. In International Conference on Machine Learning, pages 5276–5285. PMLR, 2018.

[155] Tsui-Wei Weng, Huan Zhang, Pin-Yu Chen, Jinfeng Yi, Dong Su, Yupeng Gao, Cho-Jui Hsieh, and Luca Daniel. Evaluating the robustness of neural networks: An extreme value theory approach. In International Conference on Learning Representations, 2018.

[156] Eric Wong and J Zico Kolter. Learning perturbation sets for robust machine learning. In International Conference on Learning Representations, 2021.

[157] Eric Wong and Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In International Conference on Machine Learning, pages 5286–5295. PMLR, 2018.

[158] Eric Wong, Leslie Rice, and J. Zico Kolter. Fast is better than free: Revisiting adversarial training. In International Conference on Learning Representations, 2020.

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

[160] Eric Wong, Frank Schmidt, Jan Hendrik Metzen, and J Zico Kolter. Scaling provable adversarial defenses. Advances in Neural Information Processing Systems, 31, 2018.

[161] Dongxian Wu, Shu-Tao Xia, and Yisen Wang. Adversarial weight perturbation helps robust generalization. Advances in Neural Information Processing Systems, 33, 2020.

[162] Chang Xiao, Peilin Zhong, and Changxi Zheng. Enhancing adversarial defense by k-winners-take-all. In International Conference on Learning Representations, 2020.

[163] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

[164] Kai Y. Xiao, Vincent Tjeng, Nur Muhammad (Mahi) Shafiullah, and Aleksander Madry. Training for faster adversarial robustness verification via inducing reLU stability. In International Conference on Learning Representations, 2019.

[165] Cihang Xie, Jianyu Wang, Zhishuai Zhang, Zhou Ren, and Alan Yuille. Mitigating adversarial effects through randomization. In International Conference on Learning Representations, 2018.

[166] Cihang Xie and Alan Yuille. Intriguing properties of adversarial training at scale. In International Conference on Learning Representations, 2020.

[167] Kaidi Xu, Zhouxing Shi, Huan Zhang, Yihan Wang, Kai-Wei Chang, Minlie Huang, Bhavya Kailkhura, Xue Lin, and Cho-Jui Hsieh. Automatic perturbation analysis for scalable certified robustness and beyond. Advances in Neural Information Processing Systems, 33:1129–1141, 2020.

[168] Linli Xu, Koby Crammer, Dale Schuurmans, et al. Robust support vector machine training via convex outlier ablation. In AAAI, volume 6, pages 536–542, 2006.

[169] Greg Yang, Tony Duan, J Edward Hu, Hadi Salman, Ilya Razenshteyn, and Jerry Li. Randomized smoothing of all shapes and sizes. In International Conference on Machine Learning, pages 10693–10705. PMLR, 2020.

[170] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. Advances in neural information processing systems, 32, 2019.

[171] Zhewei Yao, Amir Gholami, Qi Lei, Kurt Keutzer, and Michael W Mahoney. Hessian-based analysis of large batch training and robustness to adversaries. In Advances in Neural Information Processing Systems, pages 4949–4959, 2018.

[172] Shaokai Ye, Kaidi Xu, Sijia Liu, Hao Cheng, Jan-Henrik Lambrechts, Huan Zhang, Aojun Zhou, Kaisheng Ma, Yanzhi Wang, and Xue Lin. Adversarial robustness vs. model compression, or both? In Proceedings of the IEEE/CVF International Conference on Computer Vision, pages 111–120, 2019.

[173] Jongmin Yoon, Sung Ju Hwang, and Juho Lee. Adversarial purification with score-based generative models. In International Conference on Machine Learning, pages 12062–12072. PMLR, 2021.

[174] Xiaoyong Yuan, Pan He, Qile Zhu, and Xiaolin Li. Adversarial examples: Attacks and defenses for deep learning. IEEE transactions on neural networks and learning systems, 30(9):2805–2824, 2019.

[175] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In Proceedings of the IEEE/CVF international conference on computer vision, pages 6023–6032, 2019.

[176] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In Edwin R. Hancock Richard C. Wilson and William A. P. Smith, editors, Proceedings of the British Machine Vision Conference (BMVC), pages 87.1–87.12. BMVA Press, September 2016.

[177] Dinghuai Zhang, Tianyuan Zhang, Yiping Lu, Zhanxing Zhu, and Bin Dong. You only propagate once: Accelerating adversarial training via maximal principle. In Advances in Neural Information Processing Systems, pages 227–238, 2019.

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

[179] Huan Zhang, Hongge Chen, Chaowei Xiao, Sven Gowal, Robert Stanforth, Bo Li, Duane Boning, and Cho-Jui Hsieh. Towards stable and efficient training of verifiably robust neural networks. In International Conference on Learning Representations, 2020.

[180] Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. Advances in neural information processing systems, 31, 2018.

[181] Jingfeng Zhang, Jianing Zhu, Gang Niu, Bo Han, Masashi Sugiyama, and Mohan Kankanhalli. Geometry-aware instance-reweighted adversarial training. In International Conference on Learning Representations, 2021.

[182] Haizhong Zheng, Ziqi Zhang, Juncheng Gu, Honglak Lee, and Atul Prakash. Efficient adversarial training with transferable adversarial examples. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 1181–1190, 2020.

[183] Hattie Zhou, Janice Lan, Rosanne Liu, and Jason Yosinski. Deconstructing lottery tickets: Zeros, signs, and the supermask. Advances in Neural Information Processing Systems, 32:3597–3607, 2019.

[184] Kaiyang Zhou, Ziwei Liu, Yu Qiao, Tao Xiang, and Chen Change Loy. Domain generalization in vision: A survey. arXiv preprint arXiv:2103.02503, 2021.

Curriculum Vitae

Education

École Polytechnique Fédérale de Lausanne(EPFL) — Lausanne, Switzerland
Ph.D in Computer Science — 2017 - 2022
Supervisors: Prof. Sabine Süsstrunk, Dr. Mathieu Salzmann

École Polytechnique Fédérale de Lausanne(EPFL) — Lausanne, Switzerland
M.S in Computer Science — 2015 - 2017
GPA: 5.73/6.00 Transcript

Tsinghua University — Beijing, P.R.China
B.ENG in Computer Science and Technology — 2011 - 2015
GPA: 91.34/100.00 Rank 9/123 Transcript

Publications

In reverse chronological order, * indicates equal contributions.

Refereed Papers & Patent

Chen Liu, Mathieu Salzmann, Sabine Süsstrunk. "Training Provably Robust Models by Polyhedral Envelope Regularization". *IEEE Transactions on Neural Networks and Learning Systems 2021*.

Chen Liu, Mathieu Salzmann, Tao Lin, Ryota Tomioka, Sabine Süsstrunk. "On the Loss Landscape of Adversarial Training: Identifying Challenges and How to Overcome Them". *Neural Information Processing Systems (NeurIPS) 2020*.

Chen Liu, Ryota Tomioka, Volkan Cevher. "On Certifying Non-uniform Bounds against Adversarial Attacks". *International Conference on Machine Learning (ICML) 2019*.

Ya-Ping Hsieh, Chen Liu, Volkan Cevher. "Finding the Mixed Nash Equilibria of Generative Adversarial Networks". *International Conference on Machine Learning (ICML) 2019*. **Oral** in *Smooth Games Optimization and Machine Learning Workshop in NeurIPS 2018*.

Chen Liu, Shun Miao, Kaloian Petkov, Sandra Sudarsky, Daphne Yu, Tommaso Mansi. "Consistent 3D Rendering in Medical Imaging". *European Patent No. 18160956.1 - 1208*.

Preprint

Chen Liu*, Ziqi Zhao*, Sabine Sösstrunk, Mathieu Salzmann. "Robust Binary Models by Pruning Randomly-initialized Networks". Preprint.

Chen Liu, Zhichao Huang, Mathieu Salzmann, Tong Zhang, Sabine Süsstrunk. "On the Impact of Hard Adversarial Instances on Overfitting in Adversarial Training". Preprint.

Zhichao Huang, Chen Liu, Tong Zhang. "Adversarial Examples are By-products of Overparametrization". Preprint.

Zhichao Huang, Yanbo Fan, Chen Liu, Weizhong Zhang, Yong Zhang, Mathieu Salzmann, Sabine Süsstrunk, Jue Wang. "Fast Adversarial Training with Adaptive Steps". Preprint.

Work Experience

Swisscom Digital Lab — Lausanne, Switzerland
Internship — 02/2017 - 08/2017
Master's Thesis Project: Automatic Document Summarization.

Siemens Research (USA) — Priceton, NJ, USA
Research Intern — 07/2016 - 02/2017
Automatic parameter tuning for a 3D medical-imaging renderer

Awards & Honors

Qualcomm Innovation Fellowship Europe 2020 Finalist (15 candidates in Europe)

ICML Travel Award (2019)

Microsoft Research Scholarship.(MSR sponsored student 2017 - 2019)

Outstanding Undergraduate Students in Department of Computer Science and Technology in Tsinghua University. (Top 10%, 2015)

Scholarship of Academic Excellence in Tsinghua University. (2014)

Scholarship of Social Work in Tsinghua University. (2013)

Scholarship of Academic Excellence in Tsinghua University. (2013)

Teaching

Teaching Assistant at EPFL

MATH-111(e) Linear Algebra. 2019-Fall, 2020-Fall.

CS-413 Computational Photography. 2020-Spring, 2021-Spring.

EE-618 Theory and Methods for Reinforcement Learning. 2019-Spring.

EE-556 Mathematics of Data: from Theory to Computation. 2018-Fall.