이학박사 학위논문

Robustness of deep neural networks to adversarial attack: from heuristic methods to certified methods

(심층신경망의 적대적 공격에 대한 강건성: 휴리스틱 방법론부터 검증가능한 방법론까지)

2021년 8월

서울대학교 대학원

수리과학부

이성윤

Robustness of deep neural networks to adversarial attack: from heuristic methods to certified methods

(심층신경망의 적대적 공격에 대한 강건성: 휴리스틱 방법론부터 검증가능한 방법론까지)

지도교수 이재욱

이 논문을 이학박사 학위논문으로 제출함

2021년 4월

서울대학교 대학원

수리과학부

이성윤

이성윤의 이학박사 학위논문을 인준함

2021년 6월

Robustness of deep neural networks to adversarial attack: from heuristic methods to certified methods

A dissertation
submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
to the faculty of the Graduate School of
Seoul National University

by

Sungyoon Lee

Dissertation Director : Professor Jaewook Lee

Department of Mathematical Sciences
Seoul National University

August 2021

Abstract

Deep learning has shown successful results in many applications. However, it has been demonstrated that deep neural networks are vulnerable to small but adversarially designed perturbations in the input which can fool the neural network. There have been many studies on such adversarial attacks and defenses against them. However, Athalye *et al.* [1] have shown that most defenses rely on specific predefined adversarial attacks and can be completely broken by stronger adaptive attacks. Thus, certified methods are proposed to guarantee stable prediction of input within a perturbation set. We present this transition from heuristic defense to certified defense, and investigate key features of certified defenses, *tightness and smoothness.*

Key words: Deep Learning, Adversarial Robustness, Certified Defense
Student Number: 2016-28750

Contents

Chapter 1

Introduction

Despite the success of deep learning in many applications, the existence of adversarial example, an imperceptibly modified input that is designed to fool the neural network [64, 8], hinders the application of deep learning to safety-critical domains [20, 63, 24]. There has been increasing interest in building a model that is robust to adversarial attacks [26, 58, 52, 80, 76, 28, 73]. However, most defense methods evaluate their robustness with adversarial accuracy against predefined attacks such as PGD attack [52], C&W attack [11] and AutoAttack [16]. Thus, these defenses can be broken by new attacks [1, 10, 66, 16, 15].

To this end, many training methods have been proposed to build a certifiably robust model that can be guaranteed to be robust to adversarial perturbations [30, 59, 71, 18, 54, 27, 78, 3]. They develop an upper bound on the worst-case loss over valid adversarial perturbations and minimize it to train a certifiably robust model. These certifiable training methods can be mainly categorized into two types: linear relaxation-based methods and bound propagation methods. Linear relaxation-based methods use relatively tighter bounds, but are slow, hard to scale to large models, and memory-inefficient [71, 72, 18]. On the other hand, bound propagation methods, represented by Interval Bound Propagation (IBP), are fast and scalable due to the use of simple but much looser bounds [54, 27].

Apart from these deterministic verification approaches, randomized smoothing is one promising approach to procure robust classification results. This can make any classifier to acquire a certified adversarial robustness by constructing a smoothed classifier [46, 42, 14, 61]. However, the randomized smoothing methods require a large number of samples to certify the classifier.

Chapter 2

Heuristic Defense

2.1 Heuristic Defense

2.1.1 Background

Notations

A randomized classifier can be represented as $F \sim q_{\boldsymbol{\theta}}(F)$ with a parametric distribution $q_{\boldsymbol{\theta}}(F)$, and thus the randomized classifier F can be learned via optimizing the parameter $\boldsymbol{\theta}$. A sample model from the randomized classifier F is denoted as $f : \mathcal{X} \to \mathcal{Y}$, where the input space is denoted as $\mathcal{X}$ and the output space as $\mathcal{Y} = \mathbb{R}^c$ with the number of classes c. The cross-entropy loss function is represented as $\mathcal{L}\left(f(X), Y\right)$, where (X, Y) is a pair of the input X and the corresponding label Y. To avoid confusion, we simplify the notation as $l_f(X) = \mathcal{L}\left(f(X), Y\right)$.

Adversarial attacks

Finding an adversarial example $X_0 + \boldsymbol{\epsilon}$ within a bound $\mathbb{B}(X_0)$ for an original image X_0 is often formulated as a maximization problem of the loss function as follows:

$$\max_{X_0+\boldsymbol{\epsilon}\in\mathbb{B}(X_0)} \mathcal{L}(f(X_0 + \boldsymbol{\epsilon}), Y). \tag{2.1.1}$$

Fast Gradient (Sign) Method (FG(S)M) The Fast Gradient Sign Method (FGSM) is an efficient one-step gradient-based adversarial attack. Starting from the original image X_0, it can be formulated as follows [26]:

$$X \leftarrow X_0 + \epsilon \cdot \operatorname{sign}(\nabla_X \mathcal{L}(f(X_0), Y)), \tag{2.1.2}$$

with the perturbation bound ϵ. If the adversary uses the l_2 normalization function instead of the sign function in (2.1.2), it is called the Fast Gradient Method (FGM). Both methods can be integrated into a single formulation

$$X \leftarrow X_0 + \epsilon \cdot \Pi_{\partial \mathbb{B}(0,1)} \nabla_X \mathcal{L}(f(X_0), Y) \tag{2.1.3}$$

where Π_S is the projection onto the set S, and $\partial \mathbb{B}$ represents the boundary of l_∞-ball or l_2-ball for the FGSM and the FGM, respectively. The perturbation in (2.1.3) provides the largest inner product with the gradient $\nabla_X \mathcal{L}(f(X_0), Y)$ within the ball $\mathbb{B}(X_0, \epsilon)$ around the image X_0. Therefore, (2.1.3) can be written as the following equivalent update:

$$X \leftarrow X_0 + \underset{g \in \mathbb{B}(0,\epsilon)}{\arg\max} \left(g^T \nabla_X \mathcal{L}(f(X_0), Y) \right). \tag{2.1.4}$$

Projected Gradient Descent (PGD) The Projected Gradient Descent (PGD) is one of the strongest adversarial attacks [51]. While FG(S)M approximates the loss function as a linear function in the whole bound $\mathbb{B}(X_0)$, the PGD updates the perturbed input X_k iteratively by approximating the loss function as a linear function in a smaller local region around the current k th point X_k. Starting from the original image X_0 ($k = 0$), it performs one FGSM attack with step size $\alpha (\leq \epsilon)$ and the projection onto the l_∞-box around the original image X_0 for each iteration as follows:

$$X_k' \leftarrow X_k + \Pi_{\partial \mathbb{B}(0,\alpha)} \nabla_X \mathcal{L}(f(X_k), Y), \tag{2.1.5}$$

$$X_{k+1} \leftarrow \Pi_{\mathbb{B}(X_0,\epsilon)} X_k'. \tag{2.1.6}$$

As in (2.1.4), the projection in (2.1.5) can be replaced with the following update:

$$X_k' \leftarrow X_k + \underset{g \in \mathbb{B}(0,\alpha)}{\arg\max} \left(g^T \nabla_X \mathcal{L}(f(X_k), Y) \right). \tag{2.1.7}$$

2.2 Gradient diversity regularization

2.2.1 Randomized neural network

One such attempt to defend against adversarial attacks is to construct a randomized neural network which uses different networks for each inference [29, 75, 17]. Even in the white-box setting, where the adversary has full access to the network and the defense strategy implemented, it is infeasible to utilize the gradient of the inference model to construct adversarial examples since the adversary cannot specify the model used in the inference phase.

However, the attack algorithm Expectation over transformation (EOT) successfully circumvents the previous randomized defenses by using the expected gradient over the randomization as an alternative to the true gradient of the inference model [2, 1]. EOT approximates the expected gradient by the sample mean using Monte Carlo estimation with sample models from the randomized network. The EOT attack on a randomized network can be considered as a kind of substitute-based transfer attack in the sense that the adversary has no direct access to the chosen inference model and utilizes the averaged classifier of the randomized network as a surrogate model. Therefore, the success of EOT seems to be attributed to the high transferability among sample models of randomized networks.

There has been a line of work on understanding transferability and measuring the effectiveness of transfer attacks [57, 56, 67, 68, 50]. One explanation of transferability is that it comes from the similarity between the gradients of the target and surrogate model [56]. As shown in Figure 2.1 (left, black), sample models of a randomized network based on Adv-BNN [49] tend to have aligned gradients.

2.2.2 Expectation over Transformation (EOT)

For a randomized network, simple gradient-based methods like the PGD attack cannot obtain the true gradient of the inference model because of its test-time randomness. Therefore, an attack algorithm called Expectation over Transformation (EOT) [2, 1] has been proposed which uses the expected gradient over the randomness as an estimate of the true gradient. For each

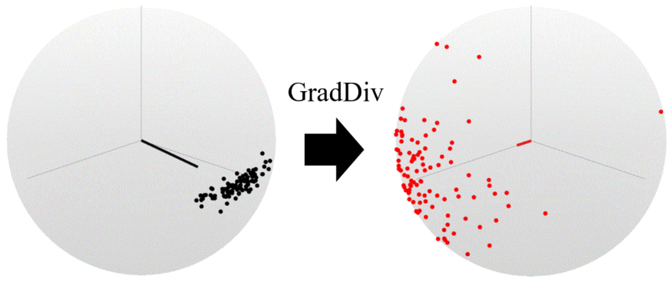


Figure 2.1: Illustration of the proposed Gradient Diversity (GradDiv) regularization. GradDiv encourages a randomized neural network to have dispersed loss gradients. The xy-plane is spanned by the sample mean vectors $\boldsymbol{u}$ (black line) and $\boldsymbol{u}_{reg}$ (red line) of the sample gradients for Adv-BNN (black dots) [49] and Adv-BNN+GradDiv (red dots), respectively. In addition, the z-axis is chosen to be orthogonal to both $\boldsymbol{u}$ and $\boldsymbol{u}_{reg}$.

step of the iterative attack, the update can be written as:

$$X_k' \leftarrow X_k + \underset{g \in \mathbb{B}(0,\alpha)}{\arg\max} \left(g^T \hat{\mathbb{E}}_{F \sim q_{\boldsymbol{\theta}}} \left[\nabla_X \mathcal{L}(F(X_k), Y) \right] \right) \tag{2.2.8}$$

in place of (2.1.5). It approximates the expected gradient by the sample mean of the sample gradient vectors from the randomized neural network. Note that standard gradient-based attacks are special instances of the EOT attack when $F \sim \delta(f)$ and δ is the Dirac delta function.

2.2.3 GradDiv

In this section, we first investigate the relationship between the gradient distribution and the robustness against the EOT attack, and then based on this analysis, we propose new regularization methods collectively called *GradDiv* to mitigate the effect of EOT on the randomized networks.

vMF Distribution and the Concentration Parameter

We model directional data of loss gradients of a randomized neural network with a p-dimensional von Mises-Fisher (vMF) distribution [53] which arises naturally in directional data. In this way, we could utilize estimated parameters of the vMF distribution from the gradient data to construct regularizers.

Definition 2.2.1. *The pdf of the vMF distribution is defined as follows:*

$$pdf(v; \mu, \kappa) = C_p(\kappa) \exp(\kappa \mu^T v), \tag{2.2.9}$$

where $v \in S^{p-1}$ is a variable vector on the $(p-1)$-dimensional hypersphere S^{p-1}, $\kappa \geq 0$ is the concentration parameter, $\mu \in S^{p-1}$ is the mean direction, $C_p(\kappa) = \frac{\kappa^{p/2-1}}{(2\pi)^{p/2} I_{p/2-1}(\kappa)}$ is the normalization constant and I_r is the r th modified Bessel function.

The (population) mean resultant length (MRL) ρ is defined as $\rho = \|\mathbb{E}[v]\|_2 \leq 1$ and it satisfies the equation $\rho = A_p(\kappa)$ with the concentration parameter κ where $A_p(\cdot) = \frac{I_{p/2}(\cdot)}{I_{p/2-1}(\cdot)}$ is a monotonically increasing function. In addition, the sample MRL $\hat{\rho}$ is defined as $\hat{\rho} = \|\bar{v}\|_2$ where the sample mean $\bar{v} = \frac{\sum_{i=1}^n v_i}{n}$ of the samples v_i. The concentration parameter κ is used as a measure of the concentration of the directional distribution. To estimate the parameter κ based on a maximum likelihood (ML) estimation, the approximation

$$\hat{\kappa} \approx \frac{\hat{\rho}(p - \hat{\rho})}{1 - \hat{\rho}^2} \tag{2.2.10}$$

is often used since the exact estimation is intractable [4]. Throughout the chapter, we use n to denote the number of directional data samples, especially the gradient samples.

Relationship between the Gradient Distribution and the EOT Attack

When the gradient of the target inference model f is not accessible, it is crucial to make a proper estimate of the actual gradient to construct adversarial

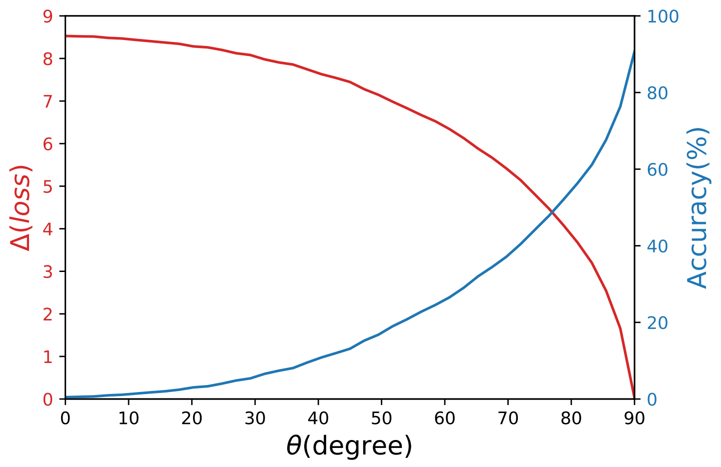


Figure 2.2: The effectiveness of the PGD attack using proxy gradients instead of the actual gradients. The proxy gradients are obtained by rotating the true gradients by θ with a random axis at every iterations of the attack. The loss difference caused by the proxy gradients (red) decreases, and the accuracy under the PGD attack (blue) increases as θ increases to 90°. Note that the graph of the loss difference (red) appears similar to the cosine curve, empirically demonstrating (2.2.12).

examples via optimization-based attacks. In other words, it requires solving the following problem in (2.1.7) in each iteration:

$$\arg\max_{g \in \mathbb{B}(0,\alpha)} \left(g^T \nabla_X \mathcal{L}(f(X), Y) \right) \tag{2.2.11}$$

without direct access to the true gradient $\nabla_X \mathcal{L}(f(X), Y)$. However, the perturbation needs not to be optimal to fool the network. As shown in Figure 2.2, a proxy gradient with a sufficiently large inner product with the true gradient is sufficient to cause misclassification.

The above discussion can be extended to our case, the white-box attack against a randomized network. In this case, it is infeasible to directly utilize the gradient of the inference model since the adversary cannot specify the model used in the inference phase. Therefore, we further investigate the effect of a proxy gradient g to the randomized network F in terms of the expected loss increase along the direction g. Using the first-order approximation, we can prove that the increase is proportional to the inner product value with the expectation $\hat{g} = \mathbb{E}_{q_{\boldsymbol{\theta}}}[\nabla_X l_F(X)]$ of the gradients as follows:

$$\Delta(g) \equiv \mathbb{E}_{q_{\boldsymbol{\theta}}}\left[l_F(X + \alpha g) - l_F(X)\right] = \alpha \hat{g}^T g + O(\alpha^2). \tag{2.2.12}$$

This explains why the graph of the loss difference (red) in Figure 2.2 looks similar to the cosine curve. Moreover, if we plug the optimal direction to maximize the loss difference, i.e., the expected gradient $\hat{g}$ used in the EOT attack, into (2.2.12), then we can upper bound the loss difference as in the following theorem:

Theorem 2.2.1. *For a randomized neural network F with bounded gradients at a given point X, the increase of the expected loss function at the point X along the direction of $\hat{g}$ with the step size α is upper bounded as follows:*

$$\Delta(\hat{g}/\|\hat{g}\|_2) \leq \alpha M_X \rho + O(\alpha^2) \tag{2.2.13}$$

for some $M_X \in \mathbb{R}$, where ρ is the MRL of the gradient direction $v = \frac{\nabla_X l_F(X)}{\|\nabla_X l_F(X)\|_2}$.

Proof. From the boundedness of the gradient of the loss function, we can derive

$$\|\hat{g}\|_2 = \|\mathbb{E}_{q_{\boldsymbol{\theta}}}[\nabla_X l_F(X)]\|_2 \leq \|M_X \mathbb{E}_{q_{\boldsymbol{\theta}}}[\frac{\nabla_X l_F(X)}{\|\nabla_X l_F(X)\|_2}]\|_2, \tag{2.2.14}$$

for some $M_X \in \mathbb{R}$. Therefore, if we choose to attack with the direction $\hat{g}$, then the following inequality for the expected difference is obtained from (2.2.12):

$$\Delta(\hat{g}/\|\hat{g}\|_2) = \alpha\|\hat{g}\|_2 + O(\alpha^2) \leq \alpha M_X \rho + O(\alpha^2). \tag{2.2.15}$$

$\square$

Therefore, we penalize MRL ρ of the gradient directions to lower the upper bound in (2.2.13), and thus build a robust model against the EOT attack. To this end, we propose the following regularizers:

$$R_\kappa(X; \boldsymbol{\theta}) = \frac{1}{p}\hat{\kappa} = \frac{1}{p}\frac{\hat{\rho}(p - \hat{\rho})}{1 - \hat{\rho}^2}, \tag{2.2.16}$$

$$R_{mean}(X;\boldsymbol{\theta}) = \frac{1}{n(n-1)}\sum_{i\neq j}\cos(g_i, g_j), \tag{2.2.17}$$

where $\{g_i\}_{i=1,\cdots,n}$ is a set of sample gradients at the point X, p is the dimension of the gradients, $\hat{\rho}$ is the sample MRL of the sample gradients, and $\cos(u, v)$ is the cosine of the angle between two directions u and v. Note that the regularizer R_{mean} in (2.2.17) is the average value of the cosine between gradient samples which directly penalizes the left-hand side of (2.2.12). We also tested variants of the regularizer (2.2.17) (see the Appendix for the details).

Reducing Transferability using DPP

A Determinantal Point Process (DPP) is a stochastic point process with a probability distribution over subsets of a given ground set $\mathcal{G}$ and the sampling of each element in the subsets being negatively correlated [37]. Therefore, it assigns higher probabilities to subsets which are diverse, and thus the probability relies on a measure used to evaluate the diversity among items in a subset. DPPs are often applied to select a subset configuration with high diversity, but we focus on learning the parameter of the distribution of the randomized neural network that would give diverse sample models.

We consider the population Ω of the randomized neural network F as a ground set $\mathcal{G} = \Omega$, and subsets $S_n = \{f_1, \cdots, f_n\}$ of sample models as point configurations of the ground set $\mathcal{G}$. We suppose that the sampling follows $F \sim q_{\boldsymbol{\theta}}(F)$, and aim to learn the parameter $\boldsymbol{\theta}$ to make the point configurations diversified, i.e., to be less transferable to each other. In this setting, we focus on a specific class of DPPs called L-ensembles [9]. An L-ensemble models the probabilities for subset $S = \{s_i\}$ as

$$\mathcal{P}_L(S) = \frac{\det(L_S)}{\sum_{T\subset\mathcal{G}}\det(L_T)} = \frac{\det(L_S)}{\det(L+I)} \propto \det(L_S) \tag{2.2.18}$$

with a positive semi-definite matrix L, where $L_S \in \mathcal{R}^{|S|\times|S|}$ is the restriction of L to the entries indexed by elements of S, where $(L_S)_{i,j} = k(s_i, s_j)$ and $k(\cdot,\cdot)$ is a kernel function that measures the similarity between two inputs. As mentioned, the measure is crucial to determine the characteristics of a DPP.

For our purposes, we define it by the inner product between the corresponding gradients, i.e., $k(f_i, f_j) = \langle g_i, g_j \rangle$, where $g_i = \nabla_X \mathcal{L}(f_i(X), Y)/\|\nabla_X \mathcal{L}(f_i(X), Y)\|_2$. Therefore, we model the probability for the sample models S_n as $\mathcal{P}_L(S_n) \propto \det(G_n^T G_n)$ where $G_n = [g_1, \cdots, g_n]$ is a $p \times n$ matrix.

Therefore, to maximize the likelihood, we use the negative log-likelihood as a regularizer:

$$\begin{aligned} R_{DPP}(X;\boldsymbol{\theta}) &= -\log \mathcal{P}_L(S_n|X;\boldsymbol{\theta}) \\ &= -\log \det(G_n^T G_n). \end{aligned} \tag{2.2.19}$$

Intuitively, $\mathcal{P}_L(S_n) \propto \det(G_n^T G_n) = Vol^2(G_n)$, where $Vol(G_n)$ is the volume of the n-dimensional parallelotope spanned by the columns of G_n, i.e., $\{g_1, \cdots, g_n\}$.

Finally, we consider the total objective of the randomized neural network with the parameter $\boldsymbol{\theta}$ formulated as:

$$L(X;\boldsymbol{\theta}) + \lambda R_{GradDiv}(X;\boldsymbol{\theta}), \tag{2.2.20}$$

where $L(X;\boldsymbol{\theta})$ is the original objective for the randomized neural network, $R_{GradDiv}$ is one of the following GradDiv regularizers $\{R_\kappa, R_{mean}, R_{DPP}\}$, and λ is the regularization parameter.

2.2.4 Experiments

Datasets and Setup. To evaluate our methods, we perform experiments on the MNIST [41], CIFAR10 [36], and STL10 [13] datasets. We extract the validation set from the training set to cross-validate the hyperparameters of the regularization methods. We use the last 5,000 training examples as the validation set for the MNIST datasets, and subsample the last 10% of the training examples for the validation set for the CIFAR10 and STL10 datasets.

During the training, we generate adversarial examples on the fly using PGD attack [51] with $\epsilon_{train} = 0.3, \frac{8}{255}, 0.03$ for MNIST, CIFAR10, and STL10, respectively, the step size $\alpha_{train} = \epsilon_{train}/4$ and the attack iterations $m = 10$.

We refer to the GradDiv regularizations in (2.2.16), (2.2.17), and (2.2.19) as GradDiv-κ, GradDiv-mean, and GradDiv-DPP, respectively. To compute

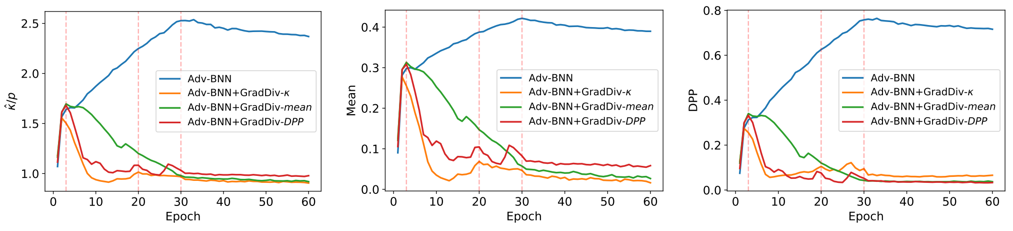


Figure 2.3: The change in the concentration measures (2.2.16) (left), (2.2.17) (middle), and (2.2.19) (right) during training on MNIST. The two leftmost vertical lines in each graph indicate the end of the warm-up and ramp-up periods, and the third vertical line indicates when the learning rate has decayed.

the GradDiv regularization terms, we have to sample the loss gradients. We empirically found that three gradient samples for each input are sufficient to improve performance.

To stabilize the training with the objective (2.2.20), we start with the regularization parameter $\lambda = 0$ during a warm-up period and linearly ramp it up to a target value during a ramp-up period. For more details about the training parameters, we refer the readers to Table A.1 in the Appendix.

Baseline. We use Adv-BNN [49] as a baseline randomized neural network model, and we test the effect of GradDiv on the baseline. In addition, we set the hyperparameters in the Adv-BNN objective as the KL regularization factor $\alpha_{KL} = 0.02$ and the prior standard deviation $\sigma_0 = 0.05, 0.1, 0.15$ for MNIST, CIFAR10, and STL10, respectively. The hyperparameters were chosen to achieve similar performance on the Adv-BNN model as in [49].

Note that GradDiv can be applied to any other randomized defenses, but we apply it to Adv-BNN because by doing so it can directly optimize the parameter on the randomness, while other methods such as [75, 29, 48] introduce randomness using manual configuration which can not be learned via stochastic gradient descent.

Effects of GradDiv during Training. Figure 2.3 shows the effects of GradDiv on the concentration measures we used, i.e., the normalized esti-

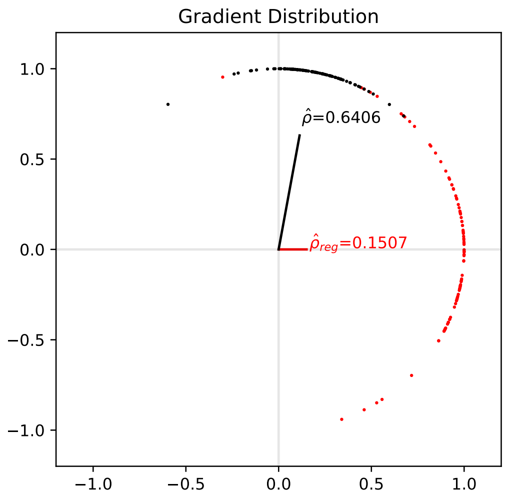


Figure 2.4: The scatter plot of the gradient samples on the unit circle. The dots indicate the gradient samples from the randomized neural network and the lines indicate the sample mean vectors of the gradients with the length of the sample MRL $\hat{\rho}$ (black: Adv-BNN, red: Adv-BNN+GradDiv).

mate $\hat{\kappa}/p$ of the concentration parameter (2.2.16), the mean of the cosine similarity (2.2.17), and the DPP loss (2.2.19) during training on MNIST. The values are averaged over each epoch. The concentration measures are successfully reduced by introducing the GradDiv regularizations after the warm-up period, while the measures tend to increase without GradDiv, especially for the first 30 epochs with large learning rates. Even though each regularizer has different objectives, they have similar regularization effects. We also observed similar results for CIFAR10 and STL10 (see the Appendix for the details). Moreover, the graphs for the networks trained with GradDiv approached optimal values. For example, the right graph of the DPP loss in Figure 2.3 shows that the DPP loss reached near to 0, which is optimal, when the n-dimensional parallelotope has the largest volume, i.e., when every gradient sample is orthogonal to each other.

Gradient Distribution and Loss Increase. We first demonstrate the ef-

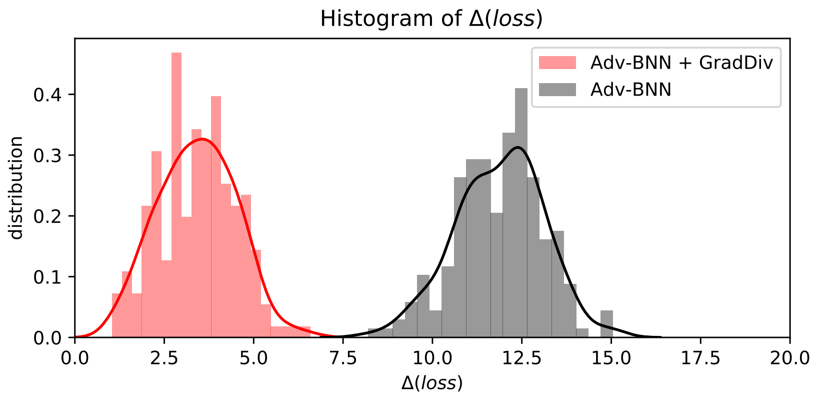


Figure 2.5: The density plots of the loss increase under the EOT attack.

fects on the directional distribution of the gradients when the regularizations are applied on MNIST as in Figure 2.4. To visualize the high dimensional gradient samples, we project them onto a 2D plane. The projection plane is spanned by two vectors $\bar{g}$ (black line) and $\bar{g}_{reg}$ (red line), where $\bar{g}$ is the sample mean vector of the gradient samples for the model that is trained without GradDiv and $\bar{g}_{reg}$ is the counterpart for the model trained with GradDiv. We sample 100 gradients at the first test image. We also noted the sample MRLs $\hat{\rho}$ and $\hat{\rho}_{reg}$ near the mean vectors $\bar{g}$ and $\bar{g}_{reg}$, respectively. The mean vectors are on the projection plane, so their lengths can be compared directly. The gradients appear to be more dispersed when the regularizations are applied (red dots in Figure 2.4). The dispersion is evaluated by the sample MRL, and it is shortened about 75% from $\hat{\rho} = 0.6406$ to $\hat{\rho}_{reg} = 0.1507$ after applying GradDiv.

In Theorem 2.2.1, we argued that the loss increase under the EOT attack is upper bounded by the MRL ρ of the gradients, and this is empirically proven in Figure 2.5. In detail, GradDiv successfully reduces the lose increase by EOT and the lose increase is about 75% smaller after applying GradDiv, which is similar to the ratio of the sample MRLs, $\hat{\rho}_{reg}/\hat{\rho}$. In the experiment, we use the EOT attack with the number of sample gradients $n = 20$, $\epsilon = 0.4$, $\alpha = 0.1$, and the attack iterations $m = 20$ on MNIST.

Table 2.1: The test accuracy against adversarial attacks on the CIFAR10 dataset. Each column indicates the results according to the different perturbation bounds ϵ. We test the randomized networks against the EOT attack with the gradient sample sizes $n = 5, 10, 20$. Note that the performances of deterministic neural networks (None and Adv.train) do not depend on the gradient sample size n.

Dataset	Attack		Method	0	0.01	0.02	0.03	0.04	0.05	0.06	0.07
CIFAR10	EOT-FGSM		None	**92.39**	26.57	12.06	8.37	7.18	6.83	6.81	6.96
			Adv.train [51]	78.83	68.12	57.65	49.63	42.72	37.38	33.14	29.26
			RSE [48]	84.06	68.49	51.83	35.67	21.6	12.9	7.08	4.12
			Adv-BNN [49]	75.97	67.94	59.28	49.9	41.59	33.91	27.47	22.1
			Adv-BNN [49]+GradDiv	76.45	**71.37**	**65.65**	**59.02**	**53.69**	**47.71**	**42.72**	**37.69**
	EOT -PGD	-	None	-	3.16	0	0	0	0	0	0
			Adv.train [51]	-	66.75	52.84	39.49	28.32	18.26	11.47	7.23
		$n = 5$	RSE [48]	-	66.39	44.5	23.86	10.48	3.98	1.4	0.44
			Adv-BNN [49]	-	67.57	58.06	46.81	36.13	25.15	16.24	9.18
			Adv-BNN [49]+GradDiv	-	**69.81**	**61.88**	**53.63**	**44.52**	**36.4**	**28.52**	**20.91**
		$n = 10$	RSE [48]	-	65.95	43.1	22.4	9.43	3.24	1.19	0.37
			Adv-BNN [49]	-	67.41	56.85	45.46	33.73	23.04	13.81	7.31
			Adv-BNN [49]+GradDiv	-	**68.59**	**59.31**	**49.17**	**38.96**	**29.53**	**20.74**	**13.56**
		$n = 20$	RSE [48]	-	65.48	42.12	21.65	9.02	3.18	1.01	0.3
			Adv-BNN [49]	-	**67.17**	56.36	44.49	32.75	21.5	12.27	6.29
			Adv-BNN [49]+GradDiv	-	66.39	**56.73**	**44.88**	**33.81**	**22.95**	**14.52**	**8.6**

The Estimated Concentration Parameter $\hat{\kappa}$. In Figure 2.6, we draw density plots of the estimated concentration parameters $\hat{\kappa}$ of the sample gradients for every test image. While Figure 2.4 shows the distribution of gradients at a single test image, the density plots can represent whole test images. We sample 100 gradients for the estimation of the concentration parameter κ using (2.2.10). As desired, the density plots show that the regularized models have lower estimated concentration parameters $\hat{\kappa}$ compared to the baseline, Adv-BNN. Note that the estimated concentration parameters $\hat{\kappa}$ for the STL10 dataset have higher values because the STL10 images are embedded in a higher-dimensional ($3 \times 96 \times 96$) space than the other datasets.

Robustness against Adversarial Attacks. For comparison, we use five models on each dataset: (1) a standard deterministic neural network trained with the clean training images referred as *None*, (2) a neural network trained with the PGD adversarial images referred as *Adv-train* [51], (3) *RSE* [48], (4) *Adv-BNN* [49], and (5) *Adv-BNN + GradDiv*, an Adv-BNN model trained

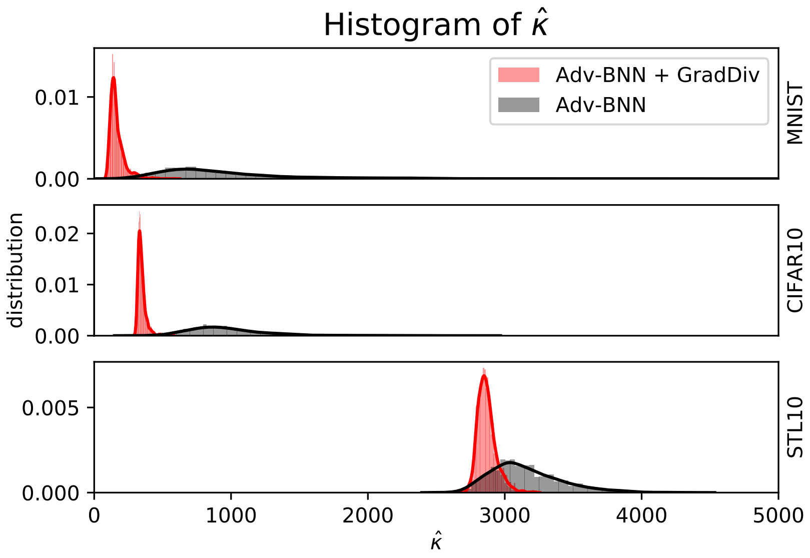


Figure 2.6: The density plots of the estimated concentration parameters $\hat{\kappa}$ for the test examples of MNIST (Top), CIFAR10 (Middle), and STL10 (Bottom).

with GradDiv. In RSE, we set the init-noise level as $\sigma_{init} = 0.2$ for the first noise layer and the inner-noise level as $\sigma_{inner} = 0.1$ for the other noise layers, the same as in [48]. Among many GradDiv regularization models, we cross-validated over the range of the hyperparameter λ, $[0.05, 10]$, with the validation set to select the best model for each dataset. The best regularizers and the corresponding regularization weights for each dataset are as follows: MNIST (GradDiv-κ, 1), CIFAR10 (GradDiv-DPP, 1), and STL10 (GradDiv-mean, 0.7). We note that there is no significant difference between the regularization methods as implied in Figure 2.3.

We perform the white-box attacks on the defense models to evaluate the robustness of the proposed methods. We emphasize that we follow [1] and evaluate the robustness against the EOT attacks, i.e., every attack is based on the expected gradients. We test the models against (1) EOT-FGSM, a one-step attack with the expected gradient, and (2) EOT-PGD, an iterative attack with the expected gradients. For EOT-FGSM, we use the number of sample gradients $n = 20$, the step-size $\alpha = \epsilon$, and the attack iteration $m = 1$.

For EOT-PGD, we use $n = 5$, 10, and 20, $\alpha = \epsilon/4$, and $m = 20$. For all randomized networks, we use 20 sample models for the ensemble. We note that the number of the ensemble is not sensitive above 20. The comparison results of the models are shown in Figure 2.7 and Table 2.1.

Figure 2.7 shows the test accuracy against EOT-PGD with the perturbation bound ϵ on MNIST and STL10. It shows that our models outperform the other models in robustness under the EOT attack. Especially, it shows high accuracy for large ϵ on MNIST. As shown in Figure 2.7 (top), using GradDiv, Adv-BNN can improve the performance by 37.52%p, 37.72%p and 15.55%p on $\epsilon = 0.36$, 0.38 and 0.4, respectively. Moreover, Adv-BNN+GradDiv shows higher accuracy than Adv-BNN for every ϵ on STL10 as shown in Figure 2.7 (bottom). We observed no significant difference when using $n > 20$ in this setting.

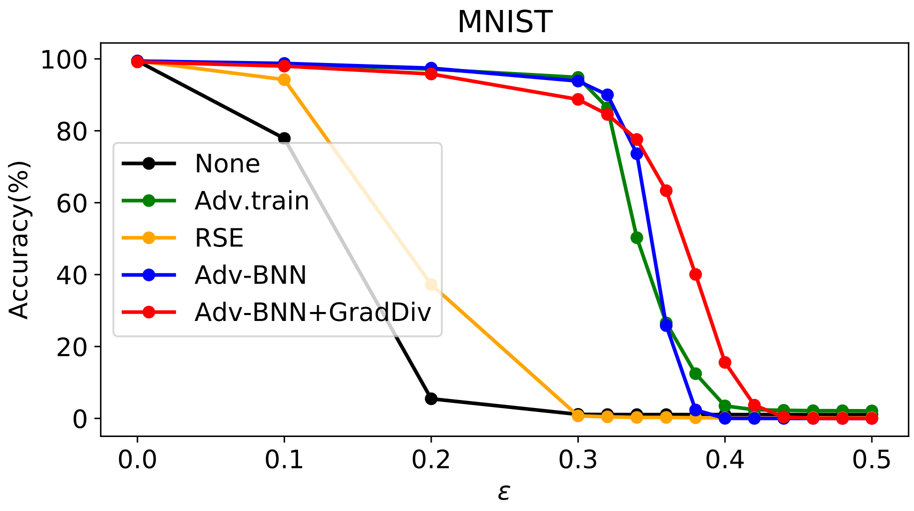
Table 2.1 shows the test accuracy against several white-box attacks on CIFAR10, including EOT-FGSM and EOT-PGD with various sample sizes n. Adv-BNN+GradDiv outperforms the other methods in most of the cases with a cost of standard accuracy drop similar to that of Adv.train and Adv-BNN. While the improvement caused by GradDiv in the case $n = 20$ is marginal, there are significant performance gaps between Adv-BNN+GradDiv and the others in other cases, especially for the smaller n and smaller attack iterations (EOT-FGSM). For the results on MNIST and STL10, we refer the readers to the Appendix.

Even though we focus on the white-box setting, we emphasize that randomized neural networks also have advantages over deterministic models in robustness under query-based black-box attacks such as [12, 7, 32, 33] since it is much harder for the adversary to estimate the actual gradient when the oracle is stochastic. Furthermore, with limited queries, the GradDiv regularized model is more effective than the other methods as shown in the experiments on the limited sample size n in Table 2.1.

Transferability between Sample Models. To further study the effect of the GradDiv regularizations, we evaluate the transferability among sample models of Adv-BNN and Adv-BNN+GradDiv-DPP. We sample 50 sample models for each randomized network and test PGD attacks with $\epsilon = 0.4$ and the attack iterations $m = 20$ on MNIST. Note that we do not have to use the EOT attack because the sample models are deterministic.

We report the results of the transfer attacks from the source models (rows) to the target models (columns) in 50×50 matrices as in Figure 2.8. The sample models of Adv-BNN+GradDiv are less transferable to each other, achieving an average test accuracy (off-diagonal) of 75% (min: 29%, max: 94%), while the samples from Adv-BNN show higher transferability with an average test accuracy of 24% (min: 4.5%, max: 67.5%). This result implies that GradDiv significantly lowers the transferability among the sample models as desired.

Cosine Similarities. To compare the cosine similarities among the sample gradients, we sample 100 sample gradients for Adv-BNN and Adv-BNN+GradDiv trained on MNIST, and plot the cosine similarity matrix as in Figure 2.9. The sample gradients from Adv-BNN+GradDiv are less aligned than those from Adv-BNN.



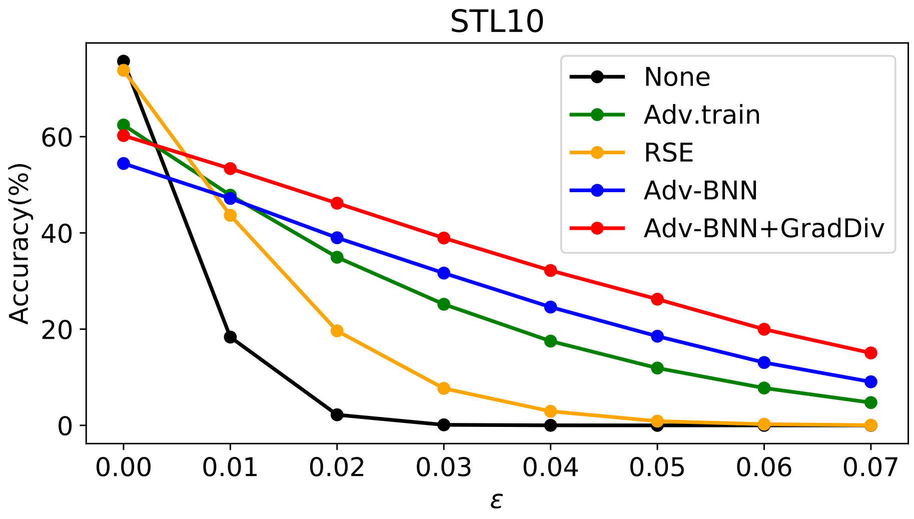


Figure 2.7: The test accuracy against the EOT attack on MNIST (top) and STL10 (bottom).

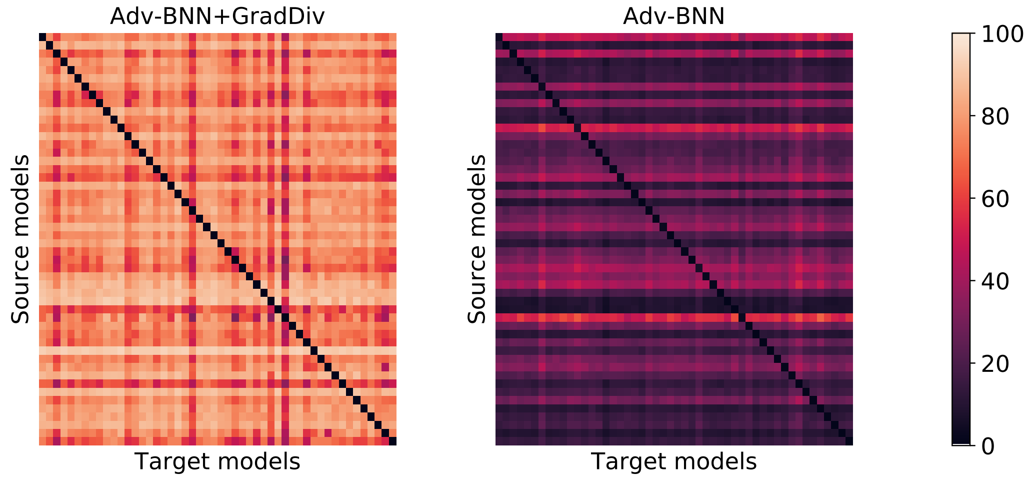


Figure 2.8: Transferability among sample models of Adv-BNN+GradDiv (left) and Adv-BNN (right). Darker colors indicate lower test accuracy, i.e., higher transferability.

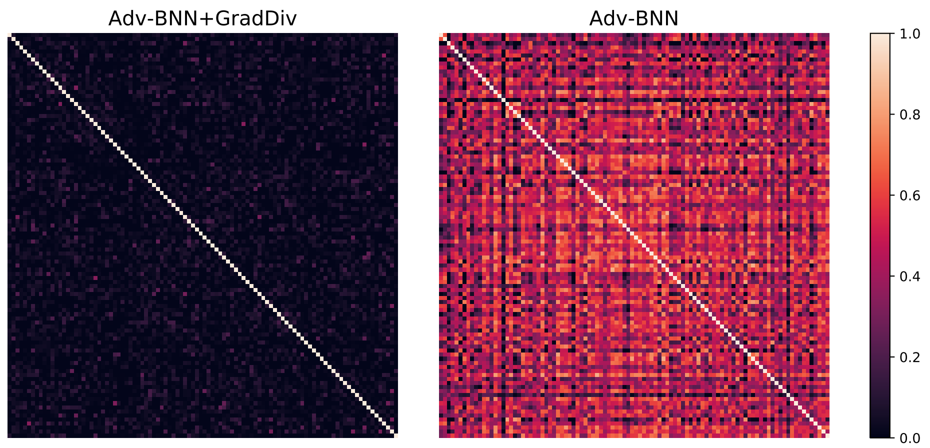


Figure 2.9: Cosine similarities among gradient samples of Adv-BNN [49]+GradDiv (left) and Adv-BNN [49] (right). Darker colors indicate lower cosine similarity.

Chapter 3

Certified Defense

3.1 Certified Defense

3.1.1 Background

In this section, we introduce the robust training problem for multi-class classification, define the specification based on the worst logit, and establish the objective that provides an upper bound of the robust training problem.

Notation We consider a c-class classification problem, where $\boldsymbol{x} \in \mathcal{X} \subset \mathbb{R}^N$ is an input, $y \in \mathcal{Y} = \{0, 1, ..., c-1\}$ is the label with respect to the input $\boldsymbol{x}$, and c is the number of classes. A mapping that takes an input $\boldsymbol{x}$ and outputs a logit vector $\boldsymbol{\zeta} = z(\boldsymbol{x}) \in \mathcal{Z}$ is denoted by $z : \mathcal{X} \to \mathcal{Z} \subset \mathbb{R}^c$, and the corresponding classifier is $f : \mathcal{X} \to \mathcal{Y}$ with $f(\boldsymbol{x}) = \arg\max_{m \in \mathcal{Y}} z_m(\boldsymbol{x})$ where z_m is the output logit for a class $m \in \mathcal{Y}$. We assume the classifier network is a feedforward network with K layers as $\mathbf{z}^{(k)} = h^{(k)}(\mathbf{z}^{(k-1)}),\ \ k = 1, \ldots, K$, where $\mathbf{z}^{(k)}$ is the vector of the activations in the k-th layer, $\mathbf{z}^{(K)} = \boldsymbol{\zeta}$, $\mathbf{z}^{(0)} = \boldsymbol{x}$, and $h^{(k)}$ is the operation in the k-th layer. Let $\mathbb{B}(\boldsymbol{x}, \epsilon)$ be a perturbation set around the input $\boldsymbol{x}$ with a level of perturbation ϵ. Then, for a classifier f, the robust classification error within the perturbation set $\mathbb{B}(\cdot, \epsilon)$ on a data distribution $\mathcal{D}$ is defined as $R(f) = \mathbb{P}_{\mathcal{D}}\big[\exists \boldsymbol{x}' \in \mathbb{B}(\boldsymbol{x}, \epsilon) \text{ s.t. } f(\boldsymbol{x}') \neq y\big]$. We omit the dependency on $\mathcal{D}$ and ϵ for simplicity.

Robust Training

The main goal of certifiable training is to minimize the robust classification error $R(f)$. However, because the exact verification for $R(f)$ is NP-complete [35], a simple surrogate of $R(f)$ is used to construct the objective of certifiable training. Our certifiable training minimizes an upper bound on $R(f)$ that builds a certificate of robustness whereas adversarial training [51] minimizes a lower bound on $R(f)$. To obtain the upper bound on $R(f)$, we propagate the perturbation set $\mathbb{B}(\boldsymbol{x}, \epsilon)$ and calculate the outer bound on the propagated image in the logit space $\mathcal{Z}$. For simplicity, $\mathbb{B}(\boldsymbol{x})$ denotes the input perturbation set $\mathbb{B}(\boldsymbol{x}, \epsilon)$. Let $\hat{z}(\mathbb{B}(\boldsymbol{x})) \subset \mathcal{Z}$ be an outer bound on the logit image of the perturbation set $z(\mathbb{B}(\boldsymbol{x}))$. Then, we can construct the following upper bound $\hat{R}(f)$ on $R(f)$:

$$\begin{aligned} R(f) &= \mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{D}}\big[\max_{\boldsymbol{\zeta}\in z(\mathbb{B}(\boldsymbol{x}))} \max_{y'\neq y} \mathbf{1}\big[\big(\zeta_y - \zeta_{y'}\big) \leq 0\big]\big] \\ &\leq \mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{D}}\big[\max_{\boldsymbol{\zeta}\in \hat{z}(\mathbb{B}(\boldsymbol{x}))} \max_{y'\neq y} \mathbf{1}\big[\big(\zeta_y - \zeta_{y'}\big) \leq 0\big]\big] = \hat{R}(f), \end{aligned} \tag{3.1.1}$$

where ζ_m is the m-th element of the logit vector $\boldsymbol{\zeta}$ and $\mathbf{1}[\cdot]$ denotes the indicator function.

Worst-Translated Logit

Based on the upper bound $\hat{R}(f)$ in (3.1.1), we can construct an objective for verifiable training as $\mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{D}}\big[\max_{\boldsymbol{\zeta}\in \hat{z}(\mathbb{B}(\boldsymbol{x}))} \mathcal{L}(\boldsymbol{\zeta}, y)\big]$ using cross-entropy loss $\mathcal{L}$ as a surrogate loss function for the 0-1 loss of $\hat{R}(f)$. However, it is still inefficient to find the optimal solution for the non-convex maximization problem $\max_{\boldsymbol{\zeta}\in \hat{z}(\mathbb{B}(\boldsymbol{x}))} \mathcal{L}(\boldsymbol{\zeta}, y)$. Dual relaxation approach addressed this problem by computing a differentiable upper bound on the robust classification error, using a feasible dual solution of the underlying relaxed LP [70, 72, 19]. In contrast, layer-wise propagation approach proposed the worst-case logit or the certifiable margin in the logit space to obtain a differentiable upper bound on the robust classification error [69, 27, 78]. In this study, we introduce the worst-translated logit $\underline{z}(\boldsymbol{x})$ that provides an upper bound on the cross-entropy loss over an outer bound $\hat{z}(\mathbb{B}(\boldsymbol{x}))$ as follows:

Definition 3.1.1. *The worst-translated logit over an outer bound $\hat{z}(\mathbb{B}(\boldsymbol{x}))$ for the input $\boldsymbol{x}$ and the corresponding label y is defined as $\underline{z}(\boldsymbol{x}; y) = z(\boldsymbol{x}) + t(\boldsymbol{x}; y)$ where the translation vector $t(\boldsymbol{x}; y)$ has its m-th element with*

$$t_m(\boldsymbol{x}; y) = (z_y(\boldsymbol{x}) - z_m(\boldsymbol{x})) - \min_{\boldsymbol{\zeta} \in \hat{z}(\mathbb{B}(\boldsymbol{x}))} \big(\zeta_y - \zeta_m\big). \tag{3.1.2}$$

When the context is clear, we omit y in $\underline{z}(\boldsymbol{x}; y)$ and $t(\boldsymbol{x}; y)$, and just write $\underline{z}(\boldsymbol{x})$ and $t(\boldsymbol{x})$ for brevity.

Proposition 3.1.1. *[[70]] For an outer bound $\hat{z}(\mathbb{B}(\boldsymbol{x})) \supset z(\mathbb{B}(\boldsymbol{x}))$ and its corresponding worst-translated logit $\underline{z}(\boldsymbol{x})$, the following inequality holds:*

$$\max_{\boldsymbol{\zeta} \in \hat{z}(\mathbb{B}(\boldsymbol{x}))} \mathcal{L}(\boldsymbol{\zeta}, y) \leq \mathcal{L}(\underline{z}(\boldsymbol{x}), y), \tag{3.1.3}$$

where $\mathcal{L}$ is the cross-entropy loss function.

Finally, the objective to be minimized is formulated as follows:

$$\mathcal{J}(f, \mathcal{D}) = \mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{D}}\big[\mathcal{L}(\underline{z}(\boldsymbol{x}), y)\big]. \tag{3.1.4}$$

Note that the worst-translated logit $\underline{z}(\boldsymbol{x})$ for the input $\boldsymbol{x}$ may not be inside the outer bound $\hat{z}(\mathbb{B}(\boldsymbol{x}))$. The remaining problem is how to calculate the outer bound $\hat{z}(\mathbb{B}(\boldsymbol{x}))$ of the logit image $z(\mathbb{B}(\boldsymbol{x}))$ and how to solve the minimization in (3.1.2) corresponding to the outer bound, which will be discussed in Section 3.2.1 and 3.2.1, respectively.

Furthermore, by using the worst-translated logit $\underline{z}(\boldsymbol{x})$ as a certificate that guarantees robustness to adversarial perturbations, we can obtain verification error of the model f on the test data $\mathcal{D}_{test}$ as follows:

$$R_V(f) = \hat{\mathbb{P}}_{(\boldsymbol{x},y)\sim\mathcal{D}_{test}}\big[\min_{y' \neq y} \big(\underline{z}_y(\boldsymbol{x}) - \underline{z}_{y'}(\boldsymbol{x})\big) \leq 0\big] \tag{3.1.5}$$

which is larger than the robust classification error $R(f)$ on the test data $\mathcal{D}_{test}$.

3.2 Tightness of the upper bound

3.2.1 Lipschitz-certifiable training with tight outer bound

In this section, we propose a tight outer bound estimation and an efficient algorithm for calculating the worst-translated logit. We mainly focus on ℓ_2-perturbation sets in the input space, but our method can be easily extended to any ℓ_p-perturbations for $p > 0$ and ℓ_∞-perturbations, as described later.

Notation The ℓ_2-perturbation set and the ℓ_∞-perturbation set in the input space are denoted by $\mathbb{B}_2(\boldsymbol{x}, \epsilon) = \{\boldsymbol{x}' : \|\boldsymbol{x}' - \boldsymbol{x}\|_2 \leq \epsilon\}$ and $\mathbb{B}_\infty(\boldsymbol{x}, \epsilon) = \{\boldsymbol{x}' : |x'_i - x_i| \leq \epsilon, \forall i\}$, respectively. To obtain a tight outer bound $\hat{z}(\mathbb{B}_2(\boldsymbol{x}, \epsilon))$, we propagate the perturbation sets through the layers and calculate layerwise outer bounds $\mathbb{B}_2^{(k)}$ and $\mathbb{B}_\infty^{(k)}$ in the k-th layer. The k-th layer ℓ_∞-bound $\mathbb{B}_\infty^{(k)}$ can be represented as the box constraint $\mathbb{B}_\infty^{(k)} = \mathrm{midrad}(\boldsymbol{m}^{(k)}, \boldsymbol{r}^{(k)}) \equiv \{\boldsymbol{p} : |p_i - m_i^{(k)}| \leq r_i^{(k)}, \forall i\}$ with the midpoint $\boldsymbol{m}^{(k)}$ and the radius $\boldsymbol{r}^{(k)}$ [55]. We call the $\mathbb{B}_2^{(k)}$ "ball outer bounds" and the $\mathbb{B}_\infty^{(k)}$ "box outer bounds".

Outer Bound Estimation

The outer bound from ℓ_2-perturbation $\hat{z}(\mathbb{B}_2(\boldsymbol{x}, \epsilon))$ can be simply constructed by using the global Lischiptz constant L of the logit function z, where $\hat{z}(\mathbb{B}_2(\boldsymbol{x}, \epsilon)) = \mathbb{B}_2(z(\boldsymbol{x}), \epsilon L)$ [69]. The global Lipschitz constant is efficiently computed as the product of all layer-wise Lipschitz constants, $L = \prod_{k=1}^{K} L^{(k)}$. To tighten the spherical outer bound in the logit space, [69] replaced it with the ellipsoidal outer bound, $\hat{z}(\mathbb{B}_2(\boldsymbol{x}, \epsilon)) = h^{(K)}(\mathbb{B}_2(\mathbf{z}^{(K-1)}, \rho^{(K-1)}))$, where $\rho^{(k)} = \epsilon \prod_{i=1}^{k} L^{(i)}$. In the objective (3.1.4), the ellipsoidal outer bound enables the Lipschitz-margin to solve the optimization in (3.1.2) explicitly. However, the global Lipschitz constant can still overestimate the outer bound and impose a strong penalty for it, limiting the expressiveness of the model [31]. It leads to a poor classification performance, not getting sharp transitions near decision boundary [30]. On the other hand, using local Lipschitz constants to estimate the outer bounds for each given input $\boldsymbol{x}$ is computationally infeasible to be integrated into the training loop for medium-sized networks, and thus limited to 2-layered networks [30, 23, 40].

To this end, we propose a method called BCP, using layer-wise propagation with the Lipschitz constant and interval arithmetic to efficiently approximate the propagation of the perturbation set adaptive to the input location. This addresses the problems of the global Lipschitz constant-based outer bound and remains the efficient computations for certifiable training. In addition, it enables us to obtain a certificate that can adapt to local properties of the classifier. We further discussed the intuition behind the design of BCP in the Appendix.

Box Constraint Propagation Our outer bound propagation starts with the ℓ_2- and ℓ_∞- perturbations sets $(\mathbb{B}_2^{(0)}, \mathbb{B}_\infty^{(0)})$, propagates them through layers, and derive the tight ℓ_∞-outer bound $\mathbb{B}_\infty^{(k)}$ in each layer by circumscribing the propagated images and finding the intersection of the circumscribed boxes. In the penultimate layer, we combine the propagated box constraint bound $\mathbb{B}_\infty^{(K-1)}$ and the propagated global Lipschitz bound $\mathbb{B}_2^{(K-1)}$ to tighten the final outer bound $\hat{z}(\mathbb{B}(\boldsymbol{x}))$.

For ℓ_2-certifiable training, we first consider the pair $(\mathbb{B}_2^{(0)}, \mathbb{B}_\infty^{(0)})$, where $\mathbb{B}_2^{(0)} \equiv \mathbb{B}_2(\boldsymbol{x}, \epsilon)$ and $\mathbb{B}_\infty^{(0)} \equiv \mathbb{B}_\infty(\boldsymbol{x}, \epsilon)$ circumscribing $\mathbb{B}_2^{(0)}$ in the input space. Next, we propagate them through the layers to compute the layerwise outer bound pair $(\mathbb{B}_2^{(k)}, \mathbb{B}_\infty^{(k)})$. Here, we assume a feedforward network, but we can extend it to residual networks (see the Appendix). The ball outer bound in the k-th layer $\mathbb{B}_2^{(k)}$ is the Lipschitz outer bound $\mathbb{B}_2(\mathbf{z}^{(k)}, \rho^{(k)})$ with the radius $\rho^{(k)} = \epsilon \prod_{i=1}^{k} L^{(i)}$, where we use the power iteration to estimate the layer-wise Lipschitz constants $L^{(i)}$. The box outer bound $\mathbb{B}_\infty^{(k+1)}$ in the $(k+1)$-th layer $(k = 0, 1, \cdots, K-2)$ is obtained by two box constraints that one circumscribes the propagated ellipse image $h^{(k+1)}(\mathbb{B}_2^{(k)})$ of the ball $\mathbb{B}_2^{(k)}$ and the other circumscribes the propagated parallelepiped image $h^{(k+1)}(\mathbb{B}_\infty^{(k)})$ of the box $\mathbb{B}_\infty^{(k)}$ as described in Figure 3.1.

In case of linear layers, the circumscribed box about the propagated ellipse image, $out_\infty(h^{(k+1)}(\mathbb{B}_2^{(k)}))$, is calculated as follows:

$$\begin{aligned} out_\infty(h^{(k+1)}(\mathbb{B}_2^{(k)})) &= \text{midrad}(\hat{\mathbf{m}}^{(k)}, \hat{\mathbf{r}}^{(k)}) \\ \text{s.t. } \hat{\mathbf{m}}^{(k)} = h^{(k+1)}(\mathbf{z}^{(k)}), &\;\; \hat{r}_i^{(k)} = \|\mathbf{W}_{i,:}^{(k+1)}\|_2\, \rho^{(k)}, \end{aligned} \tag{3.2.6}$$

where $\mathbf{W}_{i,:}^{(k+1)}$ is the i-th row of the weight matrix $\mathbf{W}^{(k+1)}$ of the linear func-

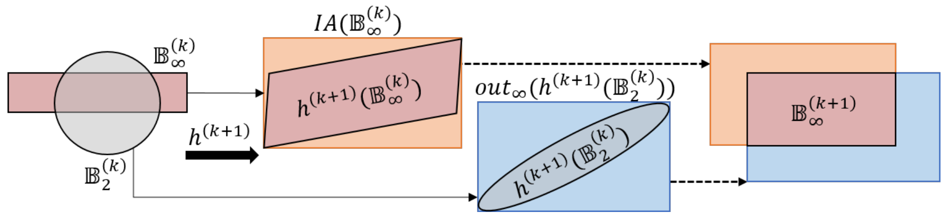

$IA(\mathbb{B}_\infty^{(k)})$
$\mathbb{B}_\infty^{(k)}$
$h^{(k+1)}(\mathbb{B}_\infty^{(k)})$
$out_\infty(h^{(k+1)}(\mathbb{B}_2^{(k)}))$
$h^{(k+1)}$
$\mathbb{B}_2^{(k)}$
$h^{(k+1)}(\mathbb{B}_2^{(k)})$
$\mathbb{B}_\infty^{(k+1)}$

Figure 3.1: Illustration of BCP. For the k-th layer, the k-th box $\mathbb{B}_\infty^{(k)}$(left) is propagated to the next box $\mathbb{B}_\infty^{(k+1)}$(right), both colored in red. Note that the k-th ball $\mathbb{B}_2^{(k)}$ is independently propagated to the next ball $\mathbb{B}_2^{(k+1)}$ which has $L^{(k)}$ times larger radius.

tion $h^{(k+1)}$. Simultaneously, we use the interval arithmetic to obtain the other box about the propagated parallelepiped image, $IA(\mathbb{B}_\infty^{(k)})$ as in [27]:

$$IA(\mathbb{B}_\infty^{(k)}) = \text{midrad}(\tilde{\mathbf{m}}^{(k)}, \tilde{\mathbf{r}}^{(k)}) \text{ s.t. } \tilde{\mathbf{m}}^{(k)} = h^{(k+1)}(\boldsymbol{m}^{(k)}),\ \tilde{\mathbf{r}}^{(k)} = |\mathbf{W}^{(k+1)}|\, \boldsymbol{r}^{(k)}, \tag{3.2.7}$$

where $|\mathbf{W}|$ takes the element-wise absolute values of $\mathbf{W}$. The above two propagations can be easily extended to nonlinear layers. The details are described in the Appendix.

Finally, we can obtain the box outer bound $\mathbb{B}_\infty^{(k+1)} = out_\infty(h^{(k+1)}(\mathbb{B}_2^{(k)})) \cap IA(\mathbb{B}_\infty^{(k)})$ for the next $(k+1)$-th layer as illustrated in Figure 3.1 with the following equations:

$$\begin{aligned}&\boldsymbol{m}^{(k+1)} = (\boldsymbol{ub}^{(k+1)} + \boldsymbol{lb}^{(k+1)})/2,\ \ \boldsymbol{r}^{(k+1)} = (\boldsymbol{ub}^{(k+1)} - \boldsymbol{lb}^{(k+1)})/2 \text{ s.t.}\\ &\boldsymbol{ub}^{(k+1)} = \max(\hat{\mathbf{m}}^{(k)} + \hat{\mathbf{r}}^{(k)}, \tilde{\mathbf{m}}^{(k)} + \tilde{\mathbf{r}}^{(k)}),\ \ \boldsymbol{lb}^{(k+1)} = \min(\hat{\mathbf{m}}^{(k)} - \hat{\mathbf{r}}^{(k)}, \tilde{\mathbf{m}}^{(k)} - \tilde{\mathbf{r}}^{(k)}),\end{aligned} \tag{3.2.8}$$

where max and min take the element-wise maximum and minimum values, respectively.

In the penultimate layer, we obtain the intersection $\mathbb{B}_\infty^{(K-1)} \cap \mathbb{B}_2^{(K-1)}$ of the box outer bound $\mathbb{B}_\infty^{(K-1)}$ and the ball outer bound $\mathbb{B}_2^{(K-1)}$. The intersection is propagated to the logit space through the last linear layer to obtain the tight outer bound, as $\hat{z}(\mathbb{B}(\boldsymbol{x})) = h^{(K)}(\mathbb{B}_\infty^{(K-1)} \cap \mathbb{B}_2^{(K-1)}) \subset \mathcal{Z}$.

Extension to ℓ_p-norm We note that BCP can be easily extended to ℓ_p-certifiable training for any $p > 0$ by modifying $\mathbb{B}_2^{(0)} = \mathbb{B}_2(\boldsymbol{x}, \epsilon)$ to $\mathbb{B}_2(\boldsymbol{x}, \epsilon')$

circumscribing $\mathbb{B}_p(\boldsymbol{x}, \epsilon)$ in the input space $\mathbb{R}^N$, where $\epsilon' = N^{1/2-1/max(p,2)}\epsilon$. For the ℓ_∞-case, we can use $\mathbb{B}_2^{(0)} = \mathbb{B}_2(\boldsymbol{x}, \sqrt{N}\epsilon)$ circumscribing $\mathbb{B}_\infty^{(0)}$. Thus, for ℓ_∞-bound, BCP can be considered as a generalized version of IBP (Interval Bound Propagation) [27]. We found that BCP shows a similar performance to IBP as an ℓ_∞-certified training (see the Appendix for the details). For now we focus on the performance of BCP under ℓ_2-perturbations.

Certifiable Training Algorithm

Formulation Our certifiable algorithm aims to minimize the objective $\mathcal{J}(f, \mathcal{D})$ in (3.1.4) to get a robust classifier. The objective contains the worst-translated logit $\underline{z}(\boldsymbol{x})$, which requires computation of the translation vector $t(\boldsymbol{x})$ in (3.1.2). In this section, we propose an efficient algorithm to calculate $t(\boldsymbol{x})$ for the tight propagated outer bound $\hat{z}(\mathbb{B}(\boldsymbol{x})) = h^{(K)}(\mathbb{B}_\infty^{(K-1)} \cap \mathbb{B}_2^{(K-1)})$ proposed in Section 3.2.1. In Equation (3.1.2), $z_y(\boldsymbol{x}) - z_m(\boldsymbol{x})$ is easily obtained by a forward pass through the network. However, the optimization $\min_{\boldsymbol{\zeta} \in \hat{z}(\mathbb{B}(\boldsymbol{x}))} \big(\zeta_y - \zeta_m\big)$ is nontrivial and dependent on the outer bound $\hat{z}(\mathbb{B}(\boldsymbol{x}))$.

Without loss of generality, we can assume that $y = 1$ and $m = 0$. Then, the optimal values ζ_0^*, ζ_1^* are as follows:

$$\zeta_0^*, \zeta_1^* = \operatorname*{arg\,min}_{(\zeta_0, \zeta_1) \in \Pi_{0,1}\hat{z}(\mathbb{B}(\boldsymbol{x}))} \big(\zeta_1 - \zeta_0\big), \tag{3.2.9}$$

where $\Pi_{0,1}$ is the projection onto the $\zeta_0\zeta_1$-plane. Then, we can formulate the following optimization:

$$\begin{aligned} \min_{\boldsymbol{\zeta} \in \hat{z}(\mathbb{B}(\boldsymbol{x}))} (\boldsymbol{e}_1 - \boldsymbol{e}_0)^T \boldsymbol{\zeta} &= \min_{\boldsymbol{\zeta}' \in \mathbb{B}_2^{(K-1)} \cap \mathbb{B}_\infty^{(K-1)}} (\boldsymbol{e}_1 - \boldsymbol{e}_0)^T h^{(K)}(\boldsymbol{\zeta}') \\ &= \min_{\boldsymbol{\zeta}' \in \mathbb{B}_2^{(K-1)} \cap \mathbb{B}_\infty^{(K-1)}} (\mathbf{W}_{1,:}^{(K)} - \mathbf{W}_{0,:}^{(K)})\boldsymbol{\zeta}' + b_1^{(K)} - b_0^{(K)}, \end{aligned} \tag{3.2.10}$$

where $\boldsymbol{e}_i$ is the i-th standard basis vector, and $\mathbf{W}^{(K)}$ and $\boldsymbol{b}^{(K)}$ is the weight matrix and the bias vector for the last linear layer $h^{(K)}$. Note that $\boldsymbol{\zeta}'$ is the vector in the penultimate layer. Therefore, we can construct the following optimization problem that finds the largest violation of the specification to verify the network:

$$\min_{\boldsymbol{\zeta}'} \ \mathbf{c}^T\boldsymbol{\zeta}' \ \text{s.t.} \ \ \|\boldsymbol{\zeta}' - \mathbf{z}^{(K-1)}\|_2 \leq \rho^{(K-1)}, \ |\boldsymbol{\zeta}' - \boldsymbol{m}^{(K-1)}| \leq \boldsymbol{r}^{(K-1)}, \tag{3.2.11}$$

where $\mathbf{c}$ is a specification vector with $\mathbf{c}^T = \mathbf{W}^{(K)}_{1,:} - \mathbf{W}^{(K)}_{0,:}$, and the second constraint takes the element-wise absolute value and the element-wise inequality. Since it is computationally expensive to integrate a typical optimization tool within the training loop, we propose a simple iterative algorithm that approaches to the optimal solution of (3.2.11) in a finite number of steps. We emphasize that by solving (3.2.11) we can obtain a better certificate than the global Lipschitz-based certificate because it uses additional box constraint, $|\boldsymbol{\zeta}' - \boldsymbol{m}^{(K-1)}| \leq \boldsymbol{r}^{(K-1)}$. We will see how this additional constraint affects the outer bound and the verification performance in Section 3.2.2.

Solving the optimization We solve the optimization (3.2.11) by using an efficient iterative algorithm that terminates when none of the elements violate the box constraint. Our algorithm starts with the initial point $\boldsymbol{p} = \mathbf{z}^{(K-1)} - \rho^{(K-1)}\frac{\mathbf{c}}{\|\mathbf{c}\|}$ which is the optimal solution of (3.2.11) when ignoring the box constraint. Then $\boldsymbol{p}$ satisfies the ball constraint but is not guaranteed to satisfy the box constraint. We decompose the indices of $\boldsymbol{p}$ into two parts, I and I^c, where $I \equiv \{l : |p_l - m^{(K-1)}_l| \geq r^{(K-1)}_l\}$. Then, we can represent $\boldsymbol{p} = \boldsymbol{p}[I] + \boldsymbol{p}[I^c]$, where $\boldsymbol{p}[J] = \sum_{l\in J} p_l \boldsymbol{e}_l$. Note that I or I^c can be empty, and we define $\boldsymbol{p}[\phi] = \mathbf{0}$. Then, we iterate the following two phases to find the optimal solution efficiently. In the first phase, $\boldsymbol{p}[I]$ is projected onto the box, denoted by $\Pi_{\mathbb{B}^{(K-1)}_\infty}\boldsymbol{p}[I]$. In the second phase, $\boldsymbol{p}[I^c]$ is scaled with an adaptive parameter $\eta \geq 1$, as computed by:

$$\eta = \frac{\sqrt{\left(\rho^{(K-1)}\right)^2 - \|\Pi_{\mathbb{B}^{(K-1)}_\infty}\boldsymbol{p}[I] - \mathbf{z}^{(K-1)}[I]\|^2}}{\|\boldsymbol{p}[I^c] - \mathbf{z}^{(K-1)}[I^c]\|}. \tag{3.2.12}$$

Based on (3.2.12), the next point $\boldsymbol{p} \leftarrow \Pi_{\mathbb{B}^{(K-1)}_\infty}\boldsymbol{p}[I] + \eta\boldsymbol{p}[I^c]$ is on the boundary $\partial\mathbb{B}^{(K-1)}_2$ of the ball $\mathbb{B}^{(K-1)}_2$ when $I^c \neq \phi$. We skip the scaling in the case of $I^c = \phi$. This iterative algorithm terminates when $\boldsymbol{p}$ satisfies the box constraint. The following proposition shows that our algorithm terminates within a finite step which is determined by the number of elements in $\mathbf{c}$.

Algorithm 1 Box Constraint Propagation (BCP) Certifiable Training

```
Input: training data $(\boldsymbol{x}, y) \sim \mathcal{D}$, target perturbation size $\epsilon_{target}$, network parameterized by $\theta$
Output: Robust network $f_\theta$
repeat
    Read mini-batch $B$ from $\mathcal{D}$ and adjust $\epsilon$ and $\lambda$ according to the schedule.
    // Compute the box outer bound and the ball outer bound //
    $\mathbb{B}_\infty^{(K-1)} = \text{midrad}(\boldsymbol{m}^{(K-1)}, \boldsymbol{r}^{(K-1)})$ where $\boldsymbol{m}^{(K-1)}, \boldsymbol{r}^{(K-1)} = \text{BCP}(\boldsymbol{x}, \epsilon; \theta)$ ((3.2.6)-(3.2.8)).
    $\mathbb{B}_2^{(K-1)} = \mathbb{B}_2(\mathbf{z}^{(K-1)}, \rho^{(K-1)})$ where $\mathbf{z}^{(K-1)} = h^{(K-1)} \circ \cdots \circ h^{(1)}(\boldsymbol{x})$ and $\rho^{(K-1)} = \epsilon \prod_{i=1}^{K-1} L^{(i)}$
    // Solve the optimization in (3.2.11) for each $m \neq y$ in parallel //
    Initialize $\boldsymbol{p} = \mathbf{z}^{(K-1)} - \rho^{(K-1)} \frac{\mathbf{c}}{\|\mathbf{c}\|}$.
    while not $|\boldsymbol{p} - \boldsymbol{m}^{(K-1)}| \leq \boldsymbol{r}^{(K-1)}$ do
        Decompose $\boldsymbol{p}$ into two parts: $\boldsymbol{p} = \boldsymbol{p}[I] + \boldsymbol{p}[I^c]$, where $I \equiv \{l : |p_l - m_l^{(K-1)}| \geq r_l^{(K-1)}\}$.
        First phase Project $\boldsymbol{p}[I]$ onto $\mathbb{B}_\infty^{(K-1)}$.
        Second phase With the scaling parameter $\eta$ in (3.2.12), update $\boldsymbol{p} \leftarrow \Pi_{\mathbb{B}_\infty^{(K-1)}} \boldsymbol{p}[I] + \eta \boldsymbol{p}[I^c]$.
    end while
    Calculate the worst-translated logit $\underline{z}(\boldsymbol{x}) = z(\boldsymbol{x}) + t(\boldsymbol{x})$ with (3.1.2) and (3.2.10):
        $t_m(\boldsymbol{x}) = \mathbf{c}^T(\mathbf{z}^{(K-1)} - \boldsymbol{p})$.
    // Update Parameters //
    Update the parameter $\theta$ with the objective (3.2.13):
        $\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{J}(f_\theta, B; \lambda)$.
until training phase ends
```

Proposition 3.2.1. *The while loop in Algorithm 1 finds the optimal solution $\boldsymbol{p} = (\boldsymbol{\zeta}')^*$ of the optimization problem (3.2.11) in a finite number of iterative steps less than the number of elements in* $\mathbf{c}$.

Proof. We denote the number of elements in $\mathbf{c}$ as N_c. For n-th iteration, we denote $I = \{l : |p_l - m_l^{(K-1)}| \geq r_l^{(K-1)}\}$ as I_n. Then, for each iteration in the while loop, at least one index does not satisfies the box constraints, i.e., $\exists i$ s.t. $|p_i - m_i^{(K-1)}| > r_i^{(K-1)}$, and the index i is added to I_n. And once an index

i is added to I_n then the i-th elements p_i is projected on the box $\mathbb{B}_\infty^{(K-1)}$ and after that it stays in I_k for $k \geq n$, i.e., I_n is a strictly increasing sequence of sets. And if $|I_n| = N_c$, then after the first phase of the iteration, there is no element that violates the box constraints, and the iteration stops (Note that when $|I_n| = N_c$, i.e., $I_n^c = \phi$, we skip the second phase). Therefore, $n \leq N_c$, i.e., our iterative algorithm stops within a finite number of iterative steps less than N_c. Now, to prove the proposition, it is enough to show the optimality of the final $\boldsymbol{p}$.

Without loss of generality, we can assume that $\mathbf{z}^{(K-1)} = 0$, $\rho^{(K-1)} = 1$ and $\mathbf{c} \leq 0$. Then we have $\mathbf{lb}^{(K-1)} \leq 0 \leq \mathbf{ub}^{(K-1)}$, and the final $\boldsymbol{p}$ satisfies $\boldsymbol{p} \geq 0$.

We put $\boldsymbol{v} = -\mathbf{c} \in \mathbb{R}^{N_c}$ and $J \equiv \{l : p_l = ub_l^{(K-1)}\}$. Then the final $\boldsymbol{p} = \boldsymbol{p}[J] + \boldsymbol{p}[J^c]$ satisfies $\boldsymbol{p}[J] \leq \alpha\boldsymbol{v}[J]$ and $\boldsymbol{p}[J^c] = \alpha\boldsymbol{v}[J^c]$ for some $\alpha \geq 1$ that is the product of all previous η's. We want to prove $\boldsymbol{p}$ is a local minimum of (3.2.11), then since (3.2.11) is a convex optimization, we can prove that $\boldsymbol{p}$ is the global optimum. We consider a closed local area $\mathbb{B}(\boldsymbol{p}, \delta > 0)$ such that for any $\boldsymbol{q} \in \mathbb{B}(\boldsymbol{p}, \delta)$, $\boldsymbol{q} \geq 0$ and we can ignore the box constraint for q_l for $l \in J^c$. We call a local optimal solution of (3.2.11) in $\mathbb{B}(\boldsymbol{p}, \delta)$ as $\boldsymbol{p}^*$. If $\boldsymbol{p}^*[J] = \boldsymbol{p}[J]$, $\boldsymbol{v}^T(\boldsymbol{p} - \boldsymbol{p}^*) = \boldsymbol{v}^T[J^c](\boldsymbol{p}[J^c] - \boldsymbol{p}^*[J^c]) = \alpha\|\boldsymbol{v}[J^c]\|^2 - \|\boldsymbol{v}[J^c]\|\|\boldsymbol{p}^*[J^c]\|\cos\phi = \|\boldsymbol{v}[J^c]\|(\|\boldsymbol{p}[J^c]\| - \|\boldsymbol{p}^*[J^c]\|\cos\phi) \geq 0$ since $\|\boldsymbol{p}[J^c]\| = \sqrt{1 - \|\boldsymbol{p}[J]\|^2} = \sqrt{1 - \|\boldsymbol{p}^*[J]\|^2} \geq \sqrt{\|\boldsymbol{p}^*\|^2 - \|\boldsymbol{p}^*[J]\|^2} = \|\boldsymbol{p}^*[J^c]\|$ where ϕ is the angle between the two vectors, $\boldsymbol{p}^*[J^c]$ and $\boldsymbol{v}[J^c]$. Thus $\boldsymbol{p}$ is a local optimal.

Therefore, to prove the proposition with a proof by contradiction, we suppose $\boldsymbol{p}^*[J] \neq \boldsymbol{p}[J]$, i.e., there is an index j such that $p_j^* < ub_j^{(K-1)}$. If $J^c = \phi$, then it contradicts the optimality of $\boldsymbol{p}^*$ since $\boldsymbol{v}^T\boldsymbol{p} > \boldsymbol{v}^T\boldsymbol{p}^*$. Therefore, we can further assume $J^c \neq \phi$, and thus $\|\boldsymbol{p}\| = 1$. Moreover, if $\|\boldsymbol{p}^*\| < 1$, then we can further extend $\boldsymbol{p}^*[J^c]$ to produce a larger inner product with $\boldsymbol{v}$, and this contradicts the assumption. Thus, $\|\boldsymbol{p}\| = \|\boldsymbol{p}^*\| = 1$ and $\|\boldsymbol{p}[J^c]\| < \|\boldsymbol{p}^*[J^c]\|$. We say $(p_j^*)^2 + \|\boldsymbol{p}^*[J^c]\|^2 = r^2$. Then we consider a two-dimensional space $\mathbf{U}$ spanned by two orthonormal vectors, $\boldsymbol{e}_j$ and $\boldsymbol{p}^*[J^c]/\|\boldsymbol{p}^*[J^c]\|$, say $\boldsymbol{u}^{(1)}$ and $\boldsymbol{u}^{(2)}$. Then $\boldsymbol{v}^{(1)} = \Pi_{\mathbf{U}}\boldsymbol{p}^* = p_j^*\boldsymbol{u}^{(1)} + \|\boldsymbol{p}^*[J^c]\|\boldsymbol{u}^{(2)} = r\cos\theta_1\boldsymbol{u}^{(1)} + r\sin\theta_1\boldsymbol{u}^{(2)}$ is on the sphere that has radius $r > 0$. We consider another vector on the sphere, $\boldsymbol{v}^{(2)} = ub_j^{(K-1)}\boldsymbol{u}^{(1)} + \beta\boldsymbol{u}^{(2)} = r\cos\theta_2\boldsymbol{u}^{(1)} + r\sin\theta_2\boldsymbol{u}^{(2)}$ with $\beta \geq 0$. Then, $\alpha\boldsymbol{v}$

projected on $\mathbf{U}$ is $\Pi_{\mathbf{U}}(\alpha\boldsymbol{v}) = (\alpha\boldsymbol{v}^T\boldsymbol{u}^{(1)})\boldsymbol{u}^{(1)} + (\alpha\boldsymbol{v}^T\boldsymbol{u}^{(2)})\boldsymbol{u}^{(2)} = (\alpha\boldsymbol{v}^T\boldsymbol{u}^{(1)})\boldsymbol{u}^{(1)} + (\boldsymbol{p}^T\boldsymbol{u}^{(2)})\boldsymbol{u}^{(2)}$ with $\alpha\boldsymbol{v}^T\boldsymbol{u}^{(1)} \geq ub_j^{(K-1)}$ because $j \in J$ and $ub_j^{(K-1)} = p_j \leq \alpha v_j$. And, $\boldsymbol{p}$ projected on $\mathbf{U}$ is $\Pi_{\mathbf{U}}\boldsymbol{p} = (\boldsymbol{p}^T\boldsymbol{u}^{(1)})\boldsymbol{u}^{(1)} + (\boldsymbol{p}^T\boldsymbol{u}^{(2)})\boldsymbol{u}^{(2)}$ with $\boldsymbol{p}^T\boldsymbol{u}^{(2)} \leq {\boldsymbol{v}^{(2)}}^T\boldsymbol{u}^{(2)} = \beta$ since $\|\boldsymbol{v}^{(2)}\| = \|\Pi_{\mathbf{U}}\mathbf{p}^*\| \geq \|\Pi_{\mathbf{U}}\boldsymbol{p}\|$. We can write $\boldsymbol{v}^T(a\boldsymbol{u}^{(1)} + b\boldsymbol{u}^{(2)}) = av_j + b\boldsymbol{v}[J^c]^T\boldsymbol{u}^{(2)} = ar_1\cos\theta_0 + br_1\cos\theta_0$ with $r_1 > 0$. Thus $\boldsymbol{v}^T\boldsymbol{v}^{(i)} = rr_1(\cos\theta_0\cos\theta_i + \sin\theta_0\sin\theta_i) = rr_1\cos(\theta_i - \theta_0)$ for $i = 1, 2$. Since $0 \leq p_j^* < ub_j^{(K-1)} \leq \alpha\boldsymbol{v}^T\boldsymbol{u}^{(1)}; 0 \leq \alpha\boldsymbol{v}^T\boldsymbol{u}^{(2)} = \boldsymbol{p}^T\boldsymbol{u}^{(2)} \leq \beta \leq \|\boldsymbol{p}^*[J^c]\|$ and $0 \leq \theta_0 \leq \theta_i \leq \pi/2$, we have $\tan\theta_0 = \alpha\boldsymbol{v}^T\boldsymbol{u}^{(2)}/\alpha\boldsymbol{v}^T\boldsymbol{u}^{(1)} \leq \boldsymbol{p}^T\boldsymbol{u}^{(2)}/ub_j^{(K-1)} \leq \tan\theta_2 = \beta/ub_j^{(K-1)} < \tan\theta_1 = \|\boldsymbol{p}^*[J^c]\|/p_j^*$. Therefore, we have $0 \leq \theta_2 - \theta_0 < \theta_1 - \theta_0 \leq \pi/2$ and $0 \leq \cos(\theta_0 - \theta_1) < \cos(\theta_0 - \theta_2)$. Therefore, $\boldsymbol{v}^T\boldsymbol{v}^{(1)} < \boldsymbol{v}^T\boldsymbol{v}^{(2)}$. However, $\Pi_{\mathbf{U}}\boldsymbol{p}$ is closer to $\boldsymbol{v}^{(2)}$ than to $\boldsymbol{v}^{(1)} = \Pi_{\mathbf{U}}\boldsymbol{p}^*$. Thus, we found $\boldsymbol{p}^*[J - \{j\}] + \boldsymbol{v}^{(2)} \in \mathbb{B}(\boldsymbol{p}, \delta)$ which yield a larger inner product with $\boldsymbol{v}$ than $\boldsymbol{p}^* = \boldsymbol{p}^*[J - \{j\}] + \boldsymbol{v}^{(1)}$ which contradicts the local optimality of $\boldsymbol{p}^*$.

□

Algorithm 1 illustrates the BCP training algorithm. Similar to [38], we train on a mixture of normal logit $z(\boldsymbol{x})$ and the worst logit $\underline{z}(\boldsymbol{x})$ as follows:

$$\mathcal{J}(f, \mathcal{D}; \lambda) = \mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{D}}\Big[(1-\lambda)\mathcal{L}(z(\boldsymbol{x}), y) + \lambda\mathcal{L}(\underline{z}(\boldsymbol{x}), y)\Big]. \tag{3.2.13}$$

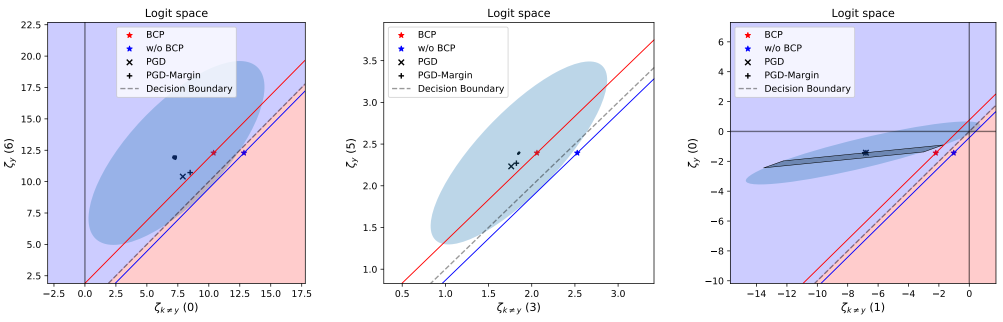
We gradually increase the perturbation ϵ from 0 to the target bound ϵ_{target} and increase λ in (3.2.13) from 0 to 1, stabilizing the initial phase of training and improving natural accuracy [27, 78]. Therefore, our algorithm enables fast certifiable training of the robust model with a tight outer bound and is, thus, scalable to large networks.

3.2.2 Experiments

We demonstrate that the proposed method can provide a tight outer bound for ℓ_2-perturbation set and train certifiably robust networks, comparing its performance against state-of-the-art certifiable training methods (LMT [69], CAP [72], and IBP [27]) on MNIST and CIFAR10. Moreover, we also show that the BCP scheme can scale to Tiny ImageNet and obtain a meaningful verification accuracy.[1] We further investigate the robustness under a wide

[1]`https://tiny-imagenet.herokuapp.com/`

range of perturbation. The details of hyper-parameters and architectures used in the experiments can be found in the Appendix.



(a) MNIST (b) CIFAR-10 (c) Acorn-or-seashore

Figure 3.2: Illustration of the outer bounds for the BCP trained models on (a) MNIST, (b) CIFAR-10, and (c) Acorn-or-seashore classification tasks. BCP cuts off the lower area under the red line from the elliptic area and tightens the outer bound. The shaded parallelogram area in (c) indicates the image of the feasible region for the box constraint after the last linear layer.

Visualization of Tightening Effects Figure 3.2 illustrates how BCP can tighten the outer region by introducing the box constraint $\mathbb{B}_\infty^{(K-1)}$ in (3.2.11). We can easily visualize the high-dimensional ellipsoid $h^{(K)}(\mathbb{B}_2^{(K-1)}) \subset \mathbb{R}^c$ in 2D plane with ζ_y- and $\zeta_{m'}$-axes by projection, where m' corresponds with the most probable class except the true class y. However, a high-dimensional parallelogram $h^{(K)}(\mathbb{B}_\infty^{(K-1)})$ is hard to visualize in the 2D plane. Thus, we use the red lines in Figure 3.2 (a)-(b) to indicate that the projection of the outer region $h^{(K)}(\mathbb{B}_2^{(K-1)} \cap \mathbb{B}_\infty^{(K-1)})$ must lie above the red line and inside the ellipsoid, showing how much area is cut off by the box constraint $\mathbb{B}_\infty^{(K-1)}$. We compute the worst-case margin ($\zeta_y - \zeta_{m'} \geq \zeta_y^* - \zeta_{m'}^*$) based on (3.2.9) and build the verification boundary with it, where the red line is obtained from the solution $\zeta_y^*, \zeta_{m'}^*$ of (3.2.9) for $\hat{z}(\mathbb{B}(\boldsymbol{x})) = h^{(K)}(\mathbb{B}_2^{(K-1)} \cap \mathbb{B}_\infty^{(K-1)})$, and the blue line is for $\hat{z}(\mathbb{B}(\boldsymbol{x})) = h^{(K)}(\mathbb{B}_2^{(K-1)})$. To verify the network, we utilize the verification boundary, where the verification for $\mathbb{B}_2(\boldsymbol{x}, \epsilon_{target})$ succeeds if the verification boundary is above the decision boundary ($\zeta_y = \zeta_{m'}$). Figure

Table 3.1: Computation time compared to CAP [72]. BCP is over 12 times faster than CAP (* For WRN, CAP uses two GPUs because of the memory limit).

Data	**Structure**	**Computation time**		**Speed up**
		CAP	**BCP**	
MNIST	4C3F	689	**57.5**	$\times$12.0
CIFAR-10	4C3F	645	**53.0**	$\times$12.2
	6C2F	1,369	**56.5**	$\times$24.2
	WRN	1,121*	**89.5**	$\times$12.5
Tiny ImageNet	8C2F	-	**3,268**	-

3.2c explicitly illustrates the ellipsoid $h^{(K)}(\mathbb{B}_2^{(K-1)})$ and the parallelogram $h^{(K)}(\mathbb{B}_\infty^{(K-1)})$ with the verification boundary for a toy binary classification problem between 'acorn' and 'seashore' derived from Tiny ImageNet dataset. We also indicate the logits for the adversarial examples against PGD attacks based on cross-entropy loss (PGD) and margin-based loss (PGD-Margin), which cannot go over the verification boundaries.

Quantitative Analysis of Tightness of Outer Bounds To quantitatively analyze how much BCP can tighten the outer bound, we use "normalized ℓ_1-norm" of the translation vector as a measure of tightness of the outer bound $\hat{z}(\mathbb{B}(\boldsymbol{x}))$ for given input $\boldsymbol{x}$, defined as $\tau(\boldsymbol{x}) = \sum_{i\neq j} t_i(\boldsymbol{x}; j)/c(c-1)$. Without BCP, this is a constant, $\tau = \sum_{i\neq j} \rho^{(K-1)} \|\mathbf{W}_{i,:}^{(K)} - \mathbf{W}_{j,:}^{(K)}\|_2/c(c-1)$, over $\mathcal{X}$ since it only considers the global Lipschitz constant and does not depend on the input. We indicate this constant tightness measure τ for each dataset as the dotted lines in Figure 3.3. On the other hand, using BCP, we can consider the local properties of inputs, and thus, we can obtain different tightness for each input. As shown in the violin plots of the tightness in Figure 3.3, BCP can tighten the outer bounds by 55.4% (MNIST), 45.8% (CIFAR-10), and 25.3% (Tiny ImageNet) on average.

Verification performance We evaluate our certifiable training algorithm and other state-of-the-art methods (LMT [69], CAP [72], and IBP [27]) with $\epsilon_{target} = 1.58, 36/255$, and $36/255$ on MNIST, CIFAR-10 and Tiny Ima-

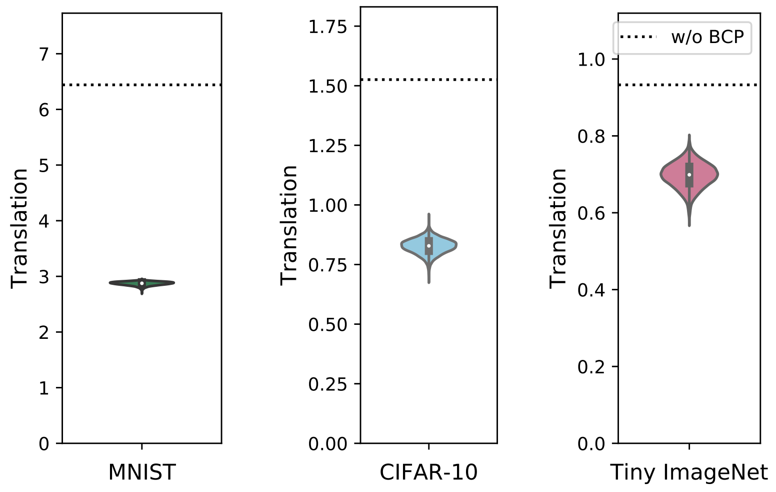


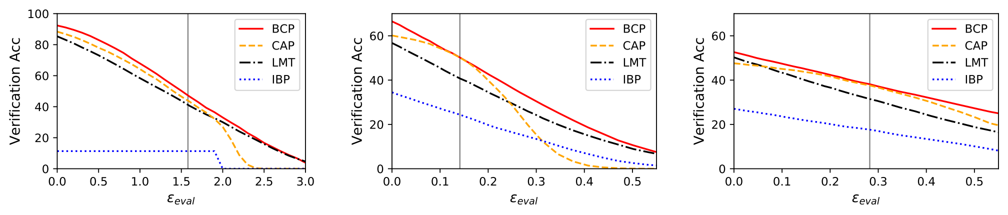
Figure 3.3: Violin plots of the tightness of the outer bounds. The dotted lines indicate the tightness without BCP. A smaller value indicates a better tightness.

Table 3.2: Comparison to other verifiable training methods. Best performances are highlighted in bold.

Data	**Structure**	**# parameters**	**Method**	**Accuracy (%)**		
				Standard	**PGD**	**Verification**
MNIST	4C3F	1974762	CAP	88.39	62.25	43.95
			LMT	86.48	53.56	40.55
			BCP	**92.41**	**64.70**	**47.95**
CIFAR-10	4C3F	2466858	CAP	60.14	55.67	**50.29**
			LMT	56.49	49.83	37.20
			IBP	34.50	31.79	24.39
			BCP	**63.88**	**58.75**	49.58
	6C2F	2250378	CAP	60.10	56.20	50.87
			LMT	63.05	58.32	38.11
			IBP	32.96	31.08	23.42
			BCP	**65.72**	**60.78**	**51.30**
	WRN	4214850	CAP	60.70	56.77	**51.63**
			LMT	61.33	56.39	33.35
			BCP	**64.79**	**59.16**	50.33
Tiny ImageNet	8C2F	4342984	BCP	**28.76**	**26.64**	**20.08**

geNet, respectively. We use the same bound for evaluation, $\epsilon_{eval} = \epsilon_{target}$. For MNIST, BCP outperforms other methods not only in terms of verification accuracy but also in terms of standard accuracy. For CIFAR-10, BCP outperforms LMT and IBP, and produces comparable performance with CAP in terms of verification accuracy, whereas outperforming in terms of both standard accuracy and robust accuracy against PGD. For Tiny ImageNet, BCP can achieve a verification accuracy of 20.08%, while LMT and IBP learn constant models and CAP is not scalable to Tiny ImageNet.

To further investigate robustness of the models, in Figure 3.4, we demonstrate the change of verification accuracy for different ℓ_2-perturbations ϵ_{eval}. We train the robust models with $\epsilon_{target} = 1.58$ on MNIST (Figure 3.4a) and $\epsilon_{target} = 36/255$ and $2\epsilon_{target} = 72/255$ on CIFAR-10 (Figure 3.4b,3.4c). Comparing to state-of-the-art methods, BCP achieves the highest verification accuracy in a wide range of ϵ_{eval}. The verification accuracy of BCP slowly decreases as increasing ϵ, and the decrease seems almost linear, while we observe a significant drop in verification accuracy when $\epsilon_{eval} \geq \epsilon_{target}$ for CAP. We emphasize that the verification accuracy against a range of perturbation involves more meaningful understanding of robustness than the verification performance at a specific perturbation bound ϵ_{eval} in Table 3.2 (see the Appendix for more detailed results).



(a) MNIST (b) CIFAR-10 (36/255) (c) CIFAR-10 (72/255)

Figure 3.4: Verification performances of verifiable training methods. The vertical lines indicate ϵ_{target}.

Computational Cost Table 3.1 shows that BCP is over 12 times faster than CAP. We evaluate the computation times on a single Titan X GPU.

For a fair comparison, we use the same batch size for both methods as 50 on MNIST and CIFAR-10 and 5 on Tiny ImageNet. Because CAP is memory-inefficient, they cannot increase the batch size, whereas we can further speed up with a larger batch size. In the case of WRN [77] on CIFAR-10, we can speed up to 61.1 sec/epoch using batch size of 128, while CAP needs two GPUs to run with a batch size of 50. It implies that our certifiable training is efficiently applicable to a large-scale dataset.

3.3 Smoothness of the objective

3.3.1 Background

First, we provide a brief overview of certifiable training methods. Then, we consider IBP [27] as a special case of linear relaxation-based methods. This unified view on certifiable training methods helps us to comprehensively analyze the differences between the two approaches: bound propagation and linear relaxation. We present the details of IBP in Appendix C.2.

Notations and Certifiable Training

We consider a c-class classification problem with a neural network $f(\boldsymbol{x};\boldsymbol{\theta})$ with the layerwise operations $\boldsymbol{z}^{(k)} = h^{(k)}(\boldsymbol{z}^{(k-1)})$ $(k = 1, \cdots, K)$ and the input $\boldsymbol{z}^{(0)} = \boldsymbol{x}$ in the input space $\mathcal{X}$. The corresponding probability function is denoted by $\boldsymbol{p}_f = \text{softmax} \circ f : \mathcal{X} \to [0,1]^c$ with subscript f. We denote a subnetwork with k operations as $h^{[k]} = h^{(k)} \circ \cdots \circ h^{(1)}$. For a linear operation $h^{(k)}$, we use $\boldsymbol{W}^{(k)}$ and $\boldsymbol{b}^{(k)}$ to denote the weight and the bias for the layer. We consider the robustness of the classifier against the norm-bounded perturbation set $\mathbb{B}(\boldsymbol{x},\epsilon) = \{\boldsymbol{x'} \in \mathcal{X} : \|\boldsymbol{x'} - \boldsymbol{x}\| \le \epsilon\}$ with the perturbation level ϵ. Here, we mainly focus on the ℓ_∞-norm bounded set. To compute the margin between the true class y for the input $\boldsymbol{x}$ and the other classes, we define a $c \times c$ matrix $\boldsymbol{C}(y) = \boldsymbol{I} - \boldsymbol{1}\boldsymbol{e}^{(y)^T}$ with $(\boldsymbol{C}(y)\boldsymbol{z}^{(K)})_m = \boldsymbol{z}^{(K)}_m - \boldsymbol{z}^{(K)}_y$ $(m = 0, \cdots, c-1)$. For the last linear layer, the weights $\boldsymbol{W}^{(K)}$ and the bias $\boldsymbol{b}^{(K)}$ are merged with $\boldsymbol{C}(y)$, that is, $\boldsymbol{W}^{(K)} \equiv \boldsymbol{C}(y)\boldsymbol{W}^{(K)}$ and $\boldsymbol{b}^{(K)} \equiv \boldsymbol{C}(y)\boldsymbol{b}^{(K)}$, yielding the margin score function $\boldsymbol{s}(\boldsymbol{x}, y; \theta) = \boldsymbol{C}(y)f(\boldsymbol{x};\theta) = f(\boldsymbol{x};\theta) - f_y(\boldsymbol{x};\theta)\boldsymbol{1}$ satisfying $\boldsymbol{p}_{\boldsymbol{s}} = \boldsymbol{p}_f$. Then we can define the worst-case margin score $\boldsymbol{s}^*(\boldsymbol{x}, y, \epsilon; \theta) =$

$\max_{\boldsymbol{x}' \in \mathbb{B}(\boldsymbol{x},\epsilon)} \boldsymbol{s}(\boldsymbol{x}', y; \theta)$ where max is element-wise maximization. With an upper bound $\overline{\boldsymbol{s}}$ on the worst-case margin score, $\overline{\boldsymbol{s}} \geq \boldsymbol{s}^*$, we can provide an upper bound on the worst-case loss over valid adversarial perturbations as follows:

$$\mathcal{L}(\overline{\boldsymbol{s}}(\boldsymbol{x}, y, \epsilon; \boldsymbol{\theta}), y) \geq \max_{\boldsymbol{x}' \in \mathbb{B}(\boldsymbol{x},\epsilon)} \mathcal{L}(f(\boldsymbol{x}'; \boldsymbol{\theta}), y) \tag{3.3.14}$$

for cross-entropy loss $\mathcal{L}$ [71]. We denote the empirical loss $\mathbb{E}_{(\boldsymbol{x},y)}[\mathcal{L}(\overline{\boldsymbol{s}}(\boldsymbol{x}, y, \epsilon; \boldsymbol{\theta}), y)]$ as $\mathcal{L}^{\epsilon}(\boldsymbol{\theta})$, Therefore, we can formulate certifiable training as a minimization of the upper bound, $\min_{\boldsymbol{\theta}} \mathcal{L}^{\epsilon}(\boldsymbol{\theta})$, instead of directly solving $\min_{\boldsymbol{\theta}} \mathbb{E}_{(\boldsymbol{x},y)}[\max_{\boldsymbol{x}' \in \mathbb{B}(\boldsymbol{x},\epsilon)} \mathcal{L}(f(\boldsymbol{x}'; \theta), y)]$ which is infeasible. Note that adversarial training [52] uses a strong iterative gradient-based attack (PGD) to provide a lower bound on the worst-case loss to be minimized, but it cannot provide a certifiably robust model. Whenever possible, we will simplify the notations by omitting variables such as $\boldsymbol{x}, y, \epsilon$, and $\boldsymbol{\theta}$.

Linear Relaxation-based Methods

For a subnetwork $h^{[k]}$, given with the pre-activation upper/lower bounds, $\boldsymbol{u}$ and $\boldsymbol{l}$, for each nonlinear activation function h in $h^{[k]}$, linear relaxation-based methods [71, 72, 78] use a relaxation of the activation function by two elementwise linear function bounds, $\overline{h}$ and $\underline{h}$, that is, $\underline{h}(\boldsymbol{z}) \leq h(\boldsymbol{z}) \leq \overline{h}(\boldsymbol{z})$ for $\boldsymbol{l} \leq \boldsymbol{z} \leq \boldsymbol{u}$. We denote the function bounds as $\overline{h}(\boldsymbol{z}) = \overline{\boldsymbol{a}} \odot \boldsymbol{z} + \overline{\boldsymbol{b}}$ and $\underline{h}(\boldsymbol{z}) = \underline{\boldsymbol{a}} \odot \boldsymbol{z} + \underline{\boldsymbol{b}}$ for some $\overline{\boldsymbol{a}}, \overline{\boldsymbol{b}}, \underline{\boldsymbol{a}}$, and $\underline{\boldsymbol{b}}$, where $\odot$ denotes the element-wise (Hadamard) product. Using all the function bounds $\overline{h}$'s and $\underline{h}$'s for the nonlinear activations in conjunction with the linear operations in $h^{[k]}$, an ith (scalar) activation $h_i^{[k]}(\cdot) \in \mathbb{R}$ can be upper bounded by a linear function $\boldsymbol{g}^T \cdot +b$ over $\mathbb{B}(\boldsymbol{x}, \epsilon)$ as in [79]. This can be equivalently explained with the dual relaxation viewpoint in [71]. Further details are provided in Appendix C.3. Now we are ready to upper bound the activation $h_i^{[k]}$ over $\mathbb{B}(\boldsymbol{x}, \epsilon)$.

Definition 3.3.1 (Linear Relaxation with Relaxed Gradient Approximation)**.** *For each neuron activation $h_i^{[k]}$, a linear relaxation method computes an upper approximation of the activation over $\mathbb{B}(\boldsymbol{x}, \epsilon)$ by using $\boldsymbol{g} \in \mathbb{R}^d$ and $b \in \mathbb{R}$ as follows:*

$$\max_{\boldsymbol{x}' \in \mathbb{B}(\boldsymbol{x},\epsilon)} h_i^{[k]}(\boldsymbol{x}') \leq \max_{\boldsymbol{x}' \in \mathbb{B}(\boldsymbol{x},\epsilon)} \boldsymbol{g}^T \boldsymbol{x}' + b = \boldsymbol{g}^T \boldsymbol{x} + \epsilon \|\boldsymbol{g}\|_* + b. \tag{3.3.15}$$

We call $\boldsymbol{g}$ *the relaxed gradient approximation of* $h_i^{[k]}$ *over* $\mathbb{B}(\boldsymbol{x}, \epsilon)$*.*

Similarly, we can obtain the corresponding lower bound. Inductively using these upper/lower bounds on the output of the subnetwork, we can obtain the bounds for the next subnetwork $h^{[k+1]}$ and then for the whole network $\boldsymbol{s}$. The final bound $\overline{\boldsymbol{s}}$ on the whole network $\boldsymbol{s}$ can then be used in the objective (3.3.14). The tightness of the bounds $\overline{\boldsymbol{s}}$ and $\mathcal{L}(\overline{\boldsymbol{s}}, y)$ highly depend on how the linear bounds $\overline{h}$ and $\underline{h}$ are chosen in each layer.

Unified view of IBP and linear relaxation-based methods IBP can also be considered as a linear relaxation-based method using zero-slope ($\overline{\boldsymbol{a}} = \underline{\boldsymbol{a}} = \boldsymbol{0}$) linear bounds, $\overline{h}(\boldsymbol{z}) = \boldsymbol{u}^+$ and $\underline{h}(\boldsymbol{z}) = \boldsymbol{l}^+$, where $\boldsymbol{v}^+ = \max(\boldsymbol{v}, \boldsymbol{0})$ and $\boldsymbol{v}^- = \min(\boldsymbol{v}, \boldsymbol{0})$. Thus, the bounds of a nonlinear activation depend only on the pre-activation bounds $\boldsymbol{u}$ and $\boldsymbol{l}$ for the activation layer, substantially reducing the feed-forward/backpropagation computations. CROWN-IBP [78] applies different linear relaxation schemes to the subnetworks and the whole network. It uses the same linear bounds as IBP for the subnetworks $h^{[k]}$ for $k < K$ except for the network $\boldsymbol{s} = h^{[K]}$ itself, and uses $\overline{h}(\boldsymbol{z}) = \frac{\boldsymbol{u}^+}{\boldsymbol{u}^+ - \boldsymbol{l}^-} \odot (\boldsymbol{z} - \boldsymbol{l}^-)$ and $\underline{h}(\boldsymbol{z}) = \boldsymbol{1}[\boldsymbol{u}^+ + \boldsymbol{l}^- > 0] \odot \boldsymbol{z}$ for the whole network $\boldsymbol{s}$. Moreover, CROWN-IBP uses interpolations between two bounds with the mixing weight β, the IBP bound and the CROWN-IBP bound, with the following objective:

$$\mathcal{L}\left((1-\beta)\overline{\boldsymbol{s}}^{\text{IBP}}(\boldsymbol{x}, y, \epsilon; \boldsymbol{\theta}) + \beta \overline{\boldsymbol{s}}^{\text{CROWN-IBP}}(\boldsymbol{x}, y, \epsilon; \boldsymbol{\theta}), y\right). \tag{3.3.16}$$

Convex Adversarial Polytope (CAP) [71, 72] uses the linear bounds $\overline{h}(\boldsymbol{z}) = \frac{\boldsymbol{u}^+}{\boldsymbol{u}^+ - \boldsymbol{l}^-} \odot (\boldsymbol{z} - \boldsymbol{l}^-)$ and $\underline{h}(\boldsymbol{z}) = \frac{\boldsymbol{u}^+}{\boldsymbol{u}^+ - \boldsymbol{l}^-} \odot \boldsymbol{z}$ for all subnetworks $h^{[k]}$ and the entire network. As CAP utilizes the linear bounds for each neuron, it is slow and memory-inefficient. It can be easily shown that tighter relaxations on nonlinear activations yield a tighter bound on the worst-case margin score $\boldsymbol{s}^*$. To specify the linear relaxation variable $\boldsymbol{\phi} \equiv \{\overline{\boldsymbol{a}}, \underline{\boldsymbol{a}}, \overline{\boldsymbol{b}}, \underline{\boldsymbol{b}}\}$ used in relaxation, we use the notation $\overline{\boldsymbol{s}}(\boldsymbol{x}, y, \epsilon; \boldsymbol{\theta}, \boldsymbol{\phi})$ with $\boldsymbol{\phi}$. CROWN-IBP and CAP generally yield a much tighter bound than IBP. These relaxation schemes are illustrated in Figure 3.5.

3.3.2 What factors influence the performance of certifiable training?

One would expect that a tighter upper bound on the worst-case loss in (3.3.14) is beneficial in certifiable training. However, several previous works have shown that this is not the case: IBP performs better than linear relaxation-based methods in many cases while utilizing a much looser bound. We investigate the loss landscape and the optimization behavior of IBP and other linear relaxation-based methods, and find that the non-smoothness of the relaxed gradient approximation of linear relaxations negatively affects their optimization. Detailed settings of the following analyses are presented in Appendix C.1.

Table 3.3: Train loss at the beginning and test error at the best checkpoint for the certifiable training methods.

	IBP		**CROWN-IBP** ($\beta = 1$)	**CAP**	**OURS**
train loss at the beginning	1.64	$>$	1.20	0.85	1.20
test error at the best checkpoint	73.19	$<$	75.82	73.91	70.92

Loss Landscape of Certifiable Training

We empirically show that CROWN-IBP [78] and CAP [71] tend to have non-smooth loss landscapes, which hinder optimization during training. We examine the learning curves of IBP and these linear relaxation-based methods. For a simple analysis on the relation between their optimization and the linear relaxation they used, we avoid considering the mixture of the two logits in (3.3.16), and use $\beta = 1$ to consider the CROWN-IBP logit only. Later, we will discuss how β-scheduling affects the optimization. Figure 3.6 shows the learning curves on CIFAR-10 under $\epsilon_{train} = {}^{8}/_{255}$. We use ϵ_t-scheduling with the warm-up (regular training, $\epsilon_t = 0$) for the first 10 epochs and the ramp-up ($0 \leq \epsilon_t \leq \epsilon_{train}$) during epochs 10-130 where we linearly increases the perturbation radius ϵ_t at iteration t from 0 to the target perturbation

ϵ_{train}. Thus, the training loss may increase even during learning.

In the early phase of the ramp-up period, in which the models are trained with small ϵ_t, CAP and CROWN-IBP have lower losses than IBP as expected because they use much tighter relaxation bounds than IBP. In particular, CAP has much tighter bounds than the others because CAP uses tighter relaxations for each subnetwork. This is consistent with the known results, that CAP tends to outperform the others at small perturbations, such as $\epsilon_{train} = {}^{2}/_{255}$ on CIFAR-10 (see Table 3.4 for details). However, at the end of the training, when the perturbation reaches its maximum target value ($\epsilon_{train} = 8/255$), the opposite result is observed where CAP and CROWN-IBP ($\beta = 1$) perform worse than IBP.

To understand this inconsistency, we measure the variation in the loss value along the gradient direction (zeroth-order smoothness) as in [62], which is represented as the shaded region in Figure 3.6. We find that linear relaxation-based methods have large variations, while IBP maintains a small variation throughout the entire training phase. It is known that a smooth loss landscape with a small loss variation induces stable and fast optimization with well-behaved gradients [62], which will be discussed in detail in the following section. Therefore, even though CAP and CROWN-IBP ($\beta = 1$) show better robustness in the early phase of training, the non-smooth loss landscape in the ramp-up period hinders the optimization, yielding less robust models.

Non-smoothness measures of loss landscape Next, we further investigate the smoothness of the loss landscape to establish a relationship between the optimization behavior and linear relaxation. Figure 3.7 (middle and bottom) shows the difference between two successive loss gradient steps during training in terms of ℓ_2- and cosine distance, respectively. We observe that some linear relaxation methods have less smooth loss landscapes with unstable gradients, especially in the early stage of CAP and the middle stage of CROWN-IBP ($\beta = 1$). Moreover, since the gradient directions are not well-aligned, they may not enjoy the advantages of momentum-based optimizers and be sensitive to the learning rate. As will be discussed in the following section, we find that the loss landscape is highly related to the relaxed gradient approximation $\boldsymbol{g}$ used in linear relaxation.

Smoothness of Relaxed Gradient Approximation

In this section, we investigate the optimization behavior further from a theoretical perspective to answer the question: "What makes some loss landscapes more favorable than others?" We find that the relaxed gradient approximation of a linear relaxation affects the smoothness of the landscape. Before that, in the following theorem, we first look at the relation between the optimization and the smoothness of the loss landscape in terms of the Hessian of the loss function with respect to weight parameters.

Lemma 3.3.1. *If $n \times n$-square matrix $\boldsymbol{H}$ is real-symmetric, then for any vector $\boldsymbol{u} \in \mathbb{R}^n$ with $\|\boldsymbol{u}\| = 1$, we have*

$$\boldsymbol{u}^T \boldsymbol{H} \boldsymbol{u} \leq ||\boldsymbol{H}\boldsymbol{u}|| \tag{3.3.17}$$

Proof. Since $\boldsymbol{H} \in \mathbb{R}^{n\times n}$ is real-symmetric, we have the eigen-decomposition of the form $\boldsymbol{H} = \boldsymbol{Q}\Lambda\boldsymbol{Q}^T$ where $\boldsymbol{Q} \in \mathbb{R}^{n\times n}$ is orthonormal and $\Lambda \in \mathbb{R}^{n\times n}$ is diagonal. Then

$$\begin{aligned}\boldsymbol{u}^T\boldsymbol{H}\boldsymbol{u} &= \boldsymbol{u}^T\boldsymbol{Q}\Lambda\boldsymbol{Q}^T\boldsymbol{u} = \boldsymbol{z}^T\Lambda\boldsymbol{z} = \langle \Lambda\boldsymbol{z}, \boldsymbol{z}\rangle \\ &\leq ||\Lambda\boldsymbol{z}|| \cdot ||\boldsymbol{z}|| = ||\Lambda\boldsymbol{z}|| = \sqrt{\boldsymbol{z}^T\Lambda^2\boldsymbol{z}} = \sqrt{\boldsymbol{u}^T\boldsymbol{Q}\Lambda^2\boldsymbol{Q}^T\boldsymbol{u}} = \sqrt{\boldsymbol{u}^T\boldsymbol{H}^T\boldsymbol{H}\boldsymbol{u}} = ||\boldsymbol{H}\boldsymbol{u}||.\end{aligned}$$

□

Theorem 3.3.1. *With gradient descent using a step size within an interval I_t during the ramp-up period ($0 \leq \epsilon_t \leq \epsilon$), the loss $\mathcal{L}^\epsilon$ for the target perturbation ϵ is reduced with*

$$\mathcal{L}^\epsilon(\boldsymbol{\theta}_{t+1}) \leq \mathcal{L}^\epsilon(\boldsymbol{\theta}_t)\big(1 - \frac{\mu}{2}\cos^2(\phi_t)\|\boldsymbol{H}_t^\epsilon\boldsymbol{u}_t\|^{-1}\big) \tag{3.3.18}$$

for $\boldsymbol{u}_t = \frac{\nabla_{\boldsymbol{\theta}}\mathcal{L}^{\epsilon_t}(\boldsymbol{\theta}_t)}{\|\nabla_{\boldsymbol{\theta}}\mathcal{L}^{\epsilon_t}(\boldsymbol{\theta}_t)\|}$ where $0 < \mu \leq \frac{\|\nabla_{\boldsymbol{\theta}}\mathcal{L}^\epsilon\|^2}{2\mathcal{L}^\epsilon}$, $\cos(\phi_t) = \frac{\nabla_{\boldsymbol{\theta}}\mathcal{L}^{\epsilon T}\nabla_{\boldsymbol{\theta}}\mathcal{L}^{\epsilon_t}}{\|\nabla_{\boldsymbol{\theta}}\mathcal{L}^\epsilon\|\|\nabla_{\boldsymbol{\theta}}\mathcal{L}^{\epsilon_t}\|}$ and $\boldsymbol{H}_t^\epsilon$ satisfies $\mathcal{L}^\epsilon(\boldsymbol{\theta}_{t+1}) = \mathcal{L}^\epsilon(\boldsymbol{\theta}_t) + \nabla_{\boldsymbol{\theta}}\mathcal{L}^\epsilon(\boldsymbol{\theta}_t)^T\Delta_t + \frac{1}{2}\Delta_t^T\boldsymbol{H}_t^\epsilon\Delta_t$ with $\Delta_t = \boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t$.

Proof.

$$\mathcal{L}^\epsilon(\boldsymbol{\theta}_{t+1}) = \mathcal{L}^\epsilon(\boldsymbol{\theta}_t) + \nabla_{\boldsymbol{\theta}}\mathcal{L}^\epsilon(\boldsymbol{\theta}_t)^T\Delta_t + \frac{1}{2}\Delta_t^T\boldsymbol{H}_t^\epsilon\Delta_t$$

$$
\begin{aligned}
&=\mathcal{L}^{\epsilon}(\boldsymbol{\theta}_t)-\eta_t\nabla_{\boldsymbol{\theta}}\mathcal{L}^{\epsilon}(\boldsymbol{\theta}_t)^T\nabla_{\boldsymbol{\theta}}\mathcal{L}^{\epsilon_t}(\boldsymbol{\theta}_t)+\frac{1}{2}\eta_t^2\nabla_{\boldsymbol{\theta}}\mathcal{L}^{\epsilon_t}(\boldsymbol{\theta}_t)^T\boldsymbol{H}_t^{\epsilon}\nabla_{\boldsymbol{\theta}}\mathcal{L}^{\epsilon_t}(\boldsymbol{\theta}_t)\\
&\leq\mathcal{L}^{\epsilon}(\boldsymbol{\theta}_t)-\frac{1}{\alpha}\frac{(\nabla_{\boldsymbol{\theta}}\mathcal{L}^{\epsilon}(\boldsymbol{\theta}_t)^T\nabla_{\boldsymbol{\theta}}\mathcal{L}^{\epsilon_t}(\boldsymbol{\theta}_t))^2}{2\nabla_{\boldsymbol{\theta}}\mathcal{L}^{\epsilon_t}(\boldsymbol{\theta}_t)^T\boldsymbol{H}_t^{\epsilon}\nabla_{\boldsymbol{\theta}}\mathcal{L}^{\epsilon_t}(\boldsymbol{\theta}_t)}\\
&=\mathcal{L}^{\epsilon}(\boldsymbol{\theta}_t)\big(1-\frac{1}{\alpha}\frac{(\nabla_{\boldsymbol{\theta}}\mathcal{L}^{\epsilon}(\boldsymbol{\theta}_t)^T\nabla_{\boldsymbol{\theta}}\mathcal{L}^{\epsilon_t}(\boldsymbol{\theta}_t))^2}{2\mathcal{L}^{\epsilon}(\boldsymbol{\theta}_t)}\frac{1}{\nabla_{\boldsymbol{\theta}}\mathcal{L}^{\epsilon_t}(\boldsymbol{\theta}_t)^T\boldsymbol{H}_t^{\epsilon}\nabla_{\boldsymbol{\theta}}\mathcal{L}^{\epsilon_t}(\boldsymbol{\theta}_t)}\big)\\
&=\mathcal{L}^{\epsilon}(\boldsymbol{\theta}_t)\big(1-\frac{1}{\alpha}\frac{(\nabla_{\boldsymbol{\theta}}\mathcal{L}^{\epsilon}(\boldsymbol{\theta}_t)^T\boldsymbol{u})^2}{2\mathcal{L}^{\epsilon}(\boldsymbol{\theta}_t)}\frac{1}{\boldsymbol{u}_t^T\boldsymbol{H}_t^{\epsilon}\boldsymbol{u}_t}\big)\\
&\leq\mathcal{L}^{\epsilon}(\boldsymbol{\theta}_t)\big(1-\frac{\mu}{\alpha}\cos^2(\phi_t)\frac{1}{\boldsymbol{u}_t{}^T\boldsymbol{H}_t^{\epsilon}\boldsymbol{u}_t}\big)\\
&\leq\mathcal{L}^{\epsilon}(\boldsymbol{\theta}_t)\big(1-\frac{\mu}{\alpha}\cos^2(\phi_t)||\boldsymbol{H}_t^{\epsilon}\boldsymbol{u}_t||^{-1}\big)
\end{aligned}
$$

for any $\alpha>1$ where in the first inequality a learning rate $\eta_t\in I_t\equiv[(1-\sqrt{1-\frac{1}{\alpha}})\eta_t^*,(1+\sqrt{1-\frac{1}{\alpha}})\eta_t^*]$ is used with $\eta_t^*=\frac{\nabla_{\boldsymbol{\theta}}\mathcal{L}^{\epsilon}(\boldsymbol{\theta}_t)^T\nabla_{\boldsymbol{\theta}}\mathcal{L}^{\epsilon_t}(\boldsymbol{\theta}_t)}{\nabla_{\boldsymbol{\theta}}\mathcal{L}^{\epsilon_t}(\boldsymbol{\theta}_t)^T\boldsymbol{H}_t^{\epsilon}\nabla_{\boldsymbol{\theta}}\mathcal{L}^{\epsilon_t}(\boldsymbol{\theta}_t)}$. And the last inequality comes from Lemma 3.3.1. Using $\alpha=2$, we can derive the final inequality (3.3.18). □

Here, μ is a modified Polyak-Lojasiewicz (PL) constant which is important for the optimization together with the Hessian term $\|\boldsymbol{H}_t^{\epsilon}\boldsymbol{u}_t\|$ [6, 47]. In this chapter, we mainly focus on the Hessian term. Note that $\boldsymbol{H}_t^{\epsilon}=\nabla_{\boldsymbol{\theta}}^2\mathcal{L}^{\epsilon}(\tilde{\boldsymbol{\theta}}_t)$ is the Hessian of $\mathcal{L}^{\epsilon}$ at some $\tilde{\boldsymbol{\theta}}_t$ between $\boldsymbol{\theta}_t$ and $\boldsymbol{\theta}_{t+1}$. Thus, since $\Delta_t=-\eta_t\nabla_{\boldsymbol{\theta}}\mathcal{L}^{\epsilon_t}(\boldsymbol{\theta}_t)$ with the learning rate η_t, we can approximate

$$\|\boldsymbol{H}_t^{\epsilon}\boldsymbol{u}_t\|\propto\|\boldsymbol{H}_t^{\epsilon}\Delta_t\|\approx\|\nabla_{\boldsymbol{\theta}}\mathcal{L}^{\epsilon}(\boldsymbol{\theta}_{t+1})-\nabla_{\boldsymbol{\theta}}\mathcal{L}^{\epsilon}(\boldsymbol{\theta}_t)\|. \tag{3.3.19}$$

Therefore, it optimizes better with a smaller loss gradient difference (first-order smoothness). Figure 3.8 shows that the certifiable training methods with smaller loss gradient differences are more effective in optimizing the target objective $\mathcal{L}^{\epsilon}$, empirically supporting Theorem 3.3.1. Note that in Figure 3.7 we measure the gradient difference in (3.3.19) with $\epsilon=\epsilon_t$ to measure the non-smoothness of the loss landscape for $\mathcal{L}^{\epsilon_t}$ locally at each step during training, while in Figure 3.8 we use the target perturbation $\epsilon=\epsilon_{train}$.

To see what factors influence the loss gradient difference, we first need some mild smoothness assumptions that are natural when the network parameters $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$ are close to each other, especially they are two consecutive parameters from gradient descent update, i.e., $\boldsymbol{\theta}_t$ and $\boldsymbol{\theta}_{t+1}$.

Assumption 1. *Given a linear relaxation method, we make the following assumptions on the bias $b(\boldsymbol{x};\boldsymbol{\theta})$ in the linear relaxation (3.3.15) and the probability function $\boldsymbol{p}(\boldsymbol{x};\boldsymbol{\theta})$:*
(i) $\|\nabla_{\boldsymbol{\theta}} b(\boldsymbol{x};\boldsymbol{\theta}_2) - \nabla_{\boldsymbol{\theta}} b(\boldsymbol{x};\boldsymbol{\theta}_1)\| \leq L^b_{\boldsymbol{\theta}\boldsymbol{\theta}}\|\boldsymbol{\theta}_2 - \boldsymbol{\theta}_1\|$ and
(ii) $\|\boldsymbol{p}(\boldsymbol{x};\boldsymbol{\theta}_2) - \boldsymbol{p}(\boldsymbol{x};\boldsymbol{\theta}_1)\| \leq L^{\boldsymbol{p}}_{\boldsymbol{\theta}}\|\boldsymbol{\theta}_2 - \boldsymbol{\theta}_1\|$, $\forall \boldsymbol{\theta}_1, \boldsymbol{\theta}_2$ and $\boldsymbol{x}$.

With the above assumptions, we can provide an upper bound on the loss gradient difference for linear relaxation-based methods to understand the optimization behavior as follows:

Theorem 3.3.2. *Suppose $\boldsymbol{x} \in \mathcal{X}$ is bounded $\|\boldsymbol{x}\| \leq M$ with some $M > 0$. For a linear relaxation-based method with the upper bound $\overline{\boldsymbol{s}}_m(\boldsymbol{x};\boldsymbol{\theta}) = \max_{\boldsymbol{x}'\in\mathbb{B}(\boldsymbol{x},\epsilon)} \boldsymbol{g}^{(m)}_{\boldsymbol{\theta}}(\boldsymbol{x})^T\boldsymbol{x}' + b^{(m)}_{\boldsymbol{\theta}}(\boldsymbol{x})$, if each $b^{(m)}_{\boldsymbol{\theta}}$ and $\boldsymbol{p}_{\boldsymbol{s}}$ satisfies Assumption 1, then*

$$\begin{aligned}&\|\nabla_{\boldsymbol{\theta}}\mathcal{L}^{\epsilon}(\boldsymbol{\theta}_2) - \nabla_{\boldsymbol{\theta}}\mathcal{L}^{\epsilon}(\boldsymbol{\theta}_1)\| \\ &\leq \mathbb{E}_{(\boldsymbol{x},y)}\big[\max_m \big(2\epsilon\|\nabla_{\boldsymbol{\theta}}\boldsymbol{g}^{(m)}_{\boldsymbol{\theta}_{1,2}}(\boldsymbol{x})\| + M\|\nabla_{\boldsymbol{\theta}}\boldsymbol{g}^{(m)}_{\boldsymbol{\theta}_2}(\boldsymbol{x}) - \nabla_{\boldsymbol{\theta}}\boldsymbol{g}^{(m)}_{\boldsymbol{\theta}_1}(\boldsymbol{x})\| + L^{(m)}\|\boldsymbol{\theta}_2 - \boldsymbol{\theta}_1\|\big)\big]\end{aligned} \tag{3.3.20}$$

for any $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2$, where $L^{(m)} = L^{b^{(m)}}_{\boldsymbol{\theta}\boldsymbol{\theta}} + L^{\boldsymbol{p_s}}_{\boldsymbol{\theta}}\|\nabla_{\boldsymbol{\theta}}\overline{\boldsymbol{s}}(\boldsymbol{x};\boldsymbol{\theta}_{1,2})\|$ and $\boldsymbol{\theta}_{1,2}$ can be any of $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$.

To prove Theorem 3.3.2, we first prove the following proposition. We note that $\boldsymbol{\theta}$ and $\boldsymbol{g}$ are vectorized and the matrix norm of Jacobian is naturally defined - for example, $||\nabla_{\boldsymbol{\theta}}\boldsymbol{g}||$ is induced by the vector norms defined in $\mathcal{X}$ and Θ.

Proposition 3.3.1. *Given input $\boldsymbol{x} \in \mathcal{X}$ and perturbation radius ϵ, let $M = \max\{||\boldsymbol{x}'|| : \boldsymbol{x}' \in \mathbb{B}(\boldsymbol{x},\epsilon)\}$. Then, for the upper bound $\overline{s}(\boldsymbol{x};\boldsymbol{\theta}) = \max_{\boldsymbol{x}'\in\mathbb{B}(\boldsymbol{x},\epsilon)} \boldsymbol{g}(\boldsymbol{x};\boldsymbol{\theta})^T\boldsymbol{x}' + b(\boldsymbol{x};\boldsymbol{\theta})$ with b satisfying Assumption 1 (1), we have*

$$\begin{aligned}&||\nabla_{\boldsymbol{\theta}}\overline{s}(\boldsymbol{x};\boldsymbol{\theta}_1) - \nabla_{\boldsymbol{\theta}}\overline{s}(\boldsymbol{x};\boldsymbol{\theta}_2)|| \\ \leq& 2\epsilon||\nabla_{\boldsymbol{\theta}}\boldsymbol{g}(\boldsymbol{x};\boldsymbol{\theta}_{1,2})|| + M||\nabla_{\boldsymbol{\theta}}\boldsymbol{g}(\boldsymbol{x};\boldsymbol{\theta}_1) - \nabla_{\boldsymbol{\theta}}\boldsymbol{g}(\boldsymbol{x};\boldsymbol{\theta}_2)|| + L^b_{\boldsymbol{\theta}\boldsymbol{\theta}}||\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2||\end{aligned} \tag{3.3.21}$$

for any $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2$, where $\boldsymbol{\theta}_{1,2}$ can be any of $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$.

Proof. Say $\overline{f}(\boldsymbol{x}';\boldsymbol{\theta}) = \boldsymbol{g}(\boldsymbol{x};\boldsymbol{\theta})^T\boldsymbol{x}' + b(\boldsymbol{x};\boldsymbol{\theta})$ and the maximizer $\boldsymbol{x}_i^* = \arg\max_{\boldsymbol{x}'\in\mathbb{B}(\boldsymbol{x},\epsilon)} \overline{f}(\boldsymbol{x}';\boldsymbol{\theta}_i)$ for each $\boldsymbol{\theta}_i = \boldsymbol{\theta}_1, \boldsymbol{\theta}_2$. Then, we have

$$\begin{aligned}
&||\nabla_{\boldsymbol{\theta}}\overline{s}(\boldsymbol{x};\boldsymbol{\theta}_1) - \nabla_{\boldsymbol{\theta}}\overline{s}(\boldsymbol{x};\boldsymbol{\theta}_2)||\\
=&||\nabla_{\boldsymbol{\theta}}\overline{f}(\boldsymbol{x}_1^*;\boldsymbol{\theta}_1) - \nabla_{\boldsymbol{\theta}}\overline{f}(\boldsymbol{x}_2^*;\boldsymbol{\theta}_2)||\\
=&||\nabla_{\boldsymbol{\theta}}\overline{f}(\boldsymbol{x}_1^*;\boldsymbol{\theta}_1) - \nabla_{\boldsymbol{\theta}}\overline{f}(\boldsymbol{x}_2^*;\boldsymbol{\theta}_1) + \nabla_{\boldsymbol{\theta}}\overline{f}(\boldsymbol{x}_2^*;\boldsymbol{\theta}_1) - \nabla_{\boldsymbol{\theta}}\overline{f}(\boldsymbol{x}_2^*;\boldsymbol{\theta}_2)||\\
\leq&||\nabla_{\boldsymbol{\theta}}\overline{f}(\boldsymbol{x}_1^*;\boldsymbol{\theta}_1) - \nabla_{\boldsymbol{\theta}}\overline{f}(\boldsymbol{x}_2^*;\boldsymbol{\theta}_1)|| + ||\nabla_{\boldsymbol{\theta}}\overline{f}(\boldsymbol{x}_2^*;\boldsymbol{\theta}_1) - \nabla_{\boldsymbol{\theta}}\overline{f}(\boldsymbol{x}_2^*;\boldsymbol{\theta}_2)||. \quad (3.3.22)
\end{aligned}$$

The first term on the RHS can be upper bounded as follows:

$$\begin{aligned}
||\nabla_{\boldsymbol{\theta}}\overline{f}(\boldsymbol{x}_1^*;\boldsymbol{\theta}_1) - \nabla_{\boldsymbol{\theta}}\overline{f}(\boldsymbol{x}_2^*;\boldsymbol{\theta}_1)|| =&||\nabla_{\boldsymbol{\theta}}(\tilde{\boldsymbol{g}}_1{}^T\tilde{\boldsymbol{x}}_1^* - \tilde{\boldsymbol{g}}_1{}^T\tilde{\boldsymbol{x}}_2^*)||\\
=&||\nabla_{\boldsymbol{\theta}}(\boldsymbol{g}_1^T\boldsymbol{x}_1^* - \boldsymbol{g}_1{}^T\boldsymbol{x}_2^*)||\\
=&||\nabla_{\boldsymbol{\theta}}\boldsymbol{g}_1(\boldsymbol{x}_1^* - \boldsymbol{x}_2^*)||\\
\leq&2\epsilon||\nabla_{\boldsymbol{\theta}}\boldsymbol{g}_1||,
\end{aligned}$$

where $\boldsymbol{g}_i = \boldsymbol{g}(\boldsymbol{x};\boldsymbol{\theta}_i)$, $b_i = b(\boldsymbol{x};\boldsymbol{\theta}_i)$, $\tilde{\boldsymbol{g}}_i^T = [\boldsymbol{g}_i^T; b_i]$ and $\tilde{\boldsymbol{x}}^T = [\boldsymbol{x}^T; 1]$. And the second term on the RHS can be upper bounded as follows:

$$\begin{aligned}
||\nabla_{\boldsymbol{\theta}}\overline{f}(\boldsymbol{x}_2^*;\boldsymbol{\theta}_1) - \nabla_{\boldsymbol{\theta}}\overline{f}(\boldsymbol{x}_2^*;\boldsymbol{\theta}_2)|| =&||\nabla_{\boldsymbol{\theta}}(\tilde{\boldsymbol{g}}_1{}^T\tilde{\boldsymbol{x}}_2^* - \tilde{\boldsymbol{g}}_2{}^T\tilde{\boldsymbol{x}}_2^*)||\\
=&||\nabla_{\boldsymbol{\theta}}(\tilde{\boldsymbol{g}}_1 - \tilde{\boldsymbol{g}}_2)\tilde{\boldsymbol{x}}_2^*||\\
\leq&||\nabla_{\boldsymbol{\theta}}(\boldsymbol{g}_1 - \boldsymbol{g}_2)||||\boldsymbol{x}_2^*|| + ||\nabla_{\boldsymbol{\theta}}(b_1 - b_2)||\\
\leq&M||\nabla_{\boldsymbol{\theta}}(\boldsymbol{g}_1 - \boldsymbol{g}_2)|| + L^b_{\boldsymbol{\theta}\boldsymbol{\theta}}||\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2||,
\end{aligned}$$

Therefore, we obtain

$$\begin{aligned}
&||\nabla_{\boldsymbol{\theta}}\overline{s}(\boldsymbol{x};\boldsymbol{\theta}_1) - \nabla_{\boldsymbol{\theta}}\overline{s}(\boldsymbol{x};\boldsymbol{\theta}_2)||\\
\leq&2\epsilon||\nabla_{\boldsymbol{\theta}}\boldsymbol{g}_1|| + M||\nabla_{\boldsymbol{\theta}}(\boldsymbol{g}_1 - \boldsymbol{g}_2)|| + L^b_{\boldsymbol{\theta}\boldsymbol{\theta}}||\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2||\\
=&2\epsilon||\nabla_{\boldsymbol{\theta}}\boldsymbol{g}(\boldsymbol{x};\boldsymbol{\theta}_1)|| + M||\nabla_{\boldsymbol{\theta}}\boldsymbol{g}(\boldsymbol{x};\boldsymbol{\theta}_1) - \nabla_{\boldsymbol{\theta}}\boldsymbol{g}(\boldsymbol{x};\boldsymbol{\theta}_2)|| + L^b_{\boldsymbol{\theta}\boldsymbol{\theta}}||\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2||.
\end{aligned}$$

Note that $\boldsymbol{\theta}_1$ in the first term is arbitrarily chosen in (3.3.22). Therefore, this leads to the final inequality (3.3.21). □

proof of 3.3.2. We start with the fact that the norm of the expected value of a random vector is smaller than expected norm of the random vector.

$$||\nabla_{\boldsymbol{\theta}}\mathcal{L}^\epsilon(\boldsymbol{\theta}_2) - \nabla_{\boldsymbol{\theta}}\mathcal{L}^\epsilon(\boldsymbol{\theta}_1)|| =||\nabla_{\boldsymbol{\theta}}\mathbb{E}_{(\boldsymbol{x},y)}[\mathcal{L}(\overline{\boldsymbol{s}}(\boldsymbol{x};\boldsymbol{\theta}_2)] - \nabla_{\boldsymbol{\theta}}\mathbb{E}_{(\boldsymbol{x},y)}[\mathcal{L}(\overline{\boldsymbol{s}}(\boldsymbol{x};\boldsymbol{\theta}_1)]||$$

$$\leq \mathbb{E}_{(\boldsymbol{x},y)}[||\nabla_{\boldsymbol{\theta}}\mathcal{L}(\overline{\boldsymbol{s}}(\boldsymbol{x};\boldsymbol{\theta}_2) - \nabla_{\boldsymbol{\theta}}\mathcal{L}(\overline{\boldsymbol{s}}(\boldsymbol{x};\boldsymbol{\theta}_1)||].$$

We simplify the notation $\boldsymbol{p}_{\boldsymbol{s}}$ as $\boldsymbol{p}$. Then we have

$$\begin{aligned}
&||\nabla_{\boldsymbol{\theta}}\mathcal{L}(\overline{\boldsymbol{s}}(\boldsymbol{x};\boldsymbol{\theta}_1)) - \nabla_{\boldsymbol{\theta}}\mathcal{L}(\overline{\boldsymbol{s}}(\boldsymbol{x};\boldsymbol{\theta}_2))|| \\
=&||\nabla_{\boldsymbol{\theta}}\overline{\boldsymbol{s}}(\boldsymbol{x};\boldsymbol{\theta}_1)\nabla_{\overline{\boldsymbol{s}}}\mathcal{L}(\overline{\boldsymbol{s}}(\boldsymbol{x};\boldsymbol{\theta}_1)) - \nabla_{\boldsymbol{\theta}}\overline{\boldsymbol{s}}(\boldsymbol{x};\boldsymbol{\theta}_2)\nabla_{\overline{\boldsymbol{s}}}\mathcal{L}(\overline{\boldsymbol{s}}(\boldsymbol{x};\boldsymbol{\theta}_2))|| \\
=&||\sum_m \nabla_{\boldsymbol{\theta}}\overline{\boldsymbol{s}}_m(\boldsymbol{x};\boldsymbol{\theta}_1)(\boldsymbol{p}_m(\boldsymbol{x};\boldsymbol{\theta}_1) - \delta_{y,m}) - \nabla_{\boldsymbol{\theta}}\overline{\boldsymbol{s}}_m(\boldsymbol{x};\boldsymbol{\theta}_2)(\boldsymbol{p}_m(\boldsymbol{x};\boldsymbol{\theta}_2) - \delta_{y,m})|| \\
=&||\nabla_{\boldsymbol{\theta}}\overline{\boldsymbol{s}}(\boldsymbol{x};\boldsymbol{\theta}_1)(\boldsymbol{p}(\boldsymbol{x};\boldsymbol{\theta}_1) - \boldsymbol{e}^{(y)}) - \nabla_{\boldsymbol{\theta}}\overline{\boldsymbol{s}}(\boldsymbol{x};\boldsymbol{\theta}_2)(\boldsymbol{p}(\boldsymbol{x};\boldsymbol{\theta}_2) - \boldsymbol{e}^{(y)})|| \\
=&||\nabla_{\boldsymbol{\theta}}\overline{\boldsymbol{s}}(\boldsymbol{x};\boldsymbol{\theta}_1)\boldsymbol{p}(\boldsymbol{x};\boldsymbol{\theta}_1) - \nabla_{\boldsymbol{\theta}}\overline{\boldsymbol{s}}(\boldsymbol{x};\boldsymbol{\theta}_2)\boldsymbol{p}(\boldsymbol{x};\boldsymbol{\theta}_2)|| \\
=&||\nabla_{\boldsymbol{\theta}}\overline{\boldsymbol{s}}(\boldsymbol{x};\boldsymbol{\theta}_1)\boldsymbol{p}(\boldsymbol{x};\boldsymbol{\theta}_1) - \nabla_{\boldsymbol{\theta}}\overline{\boldsymbol{s}}(\boldsymbol{x};\boldsymbol{\theta}_1)\boldsymbol{p}(\boldsymbol{x};\boldsymbol{\theta}_2) + \nabla_{\boldsymbol{\theta}}\overline{\boldsymbol{s}}(\boldsymbol{x};\boldsymbol{\theta}_1)\boldsymbol{p}(\boldsymbol{x};\boldsymbol{\theta}_2) - \nabla_{\boldsymbol{\theta}}\overline{\boldsymbol{s}}(\boldsymbol{x};\boldsymbol{\theta}_2)\boldsymbol{p}(\boldsymbol{x};\boldsymbol{\theta}_2)|| \\
=&||\nabla_{\boldsymbol{\theta}}\overline{\boldsymbol{s}}(\boldsymbol{x};\boldsymbol{\theta}_1)(\boldsymbol{p}(\boldsymbol{x};\boldsymbol{\theta}_1) - \boldsymbol{p}(\boldsymbol{x};\boldsymbol{\theta}_2)) + (\nabla_{\boldsymbol{\theta}}\overline{\boldsymbol{s}}(\boldsymbol{x};\boldsymbol{\theta}_1) - \nabla_{\boldsymbol{\theta}}\overline{\boldsymbol{s}}(\boldsymbol{x};\boldsymbol{\theta}_2))\boldsymbol{p}(\boldsymbol{x};\boldsymbol{\theta}_2)|| \\
\leq&||\nabla_{\boldsymbol{\theta}}\overline{\boldsymbol{s}}(\boldsymbol{x};\boldsymbol{\theta}_1)||||\boldsymbol{p}(\boldsymbol{x};\boldsymbol{\theta}_1) - \boldsymbol{p}(\boldsymbol{x};\boldsymbol{\theta}_2)|| + \max_m ||\nabla_{\boldsymbol{\theta}}\overline{\boldsymbol{s}}_m(\boldsymbol{x};\boldsymbol{\theta}_1) - \nabla_{\boldsymbol{\theta}}\overline{\boldsymbol{s}}_m(\boldsymbol{x};\boldsymbol{\theta}_2)|| \\
\leq&||\nabla_{\boldsymbol{\theta}}\overline{\boldsymbol{s}}(\boldsymbol{x};\boldsymbol{\theta}_1)||L^{\boldsymbol{p}}_{\boldsymbol{\theta}}||\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2|| + \max_m ||\nabla_{\boldsymbol{\theta}}\overline{\boldsymbol{s}}_m(\boldsymbol{x};\boldsymbol{\theta}_1) - \nabla_{\boldsymbol{\theta}}\overline{\boldsymbol{s}}_m(\boldsymbol{x};\boldsymbol{\theta}_2)|| \\
\leq&\max_m \left(2\epsilon||\nabla_{\boldsymbol{\theta}}\boldsymbol{g}^{(m)}(\boldsymbol{x};\boldsymbol{\theta}_{1,2})|| + M||\nabla_{\boldsymbol{\theta}}\boldsymbol{g}^{(m)}(\boldsymbol{x};\boldsymbol{\theta}_1) - \nabla_{\boldsymbol{\theta}}\boldsymbol{g}^{(m)}(\boldsymbol{x};\boldsymbol{\theta}_2)|| + L^{(m)}||\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2||\right).
\end{aligned}$$

□

According to Theorem 3.3.2, the relaxed gradient approximations $\boldsymbol{g}^{(m)}$ in the linear relaxation play a major role in shaping the loss landscape. The first two terms in the right hand side of (3.3.20) indicate the zeroth- and first-order smoothness of the relaxed gradient approximation with respect to weight parameters, respectively. The smoother the relaxed gradient approximations are, the smoother the loss landscape is. Especially for IBP, using the zero-slope relaxed gradient approximation $\boldsymbol{g}^{(m)} \equiv \boldsymbol{0}$ for all m, the loss gradient difference is upper bounded by only the last term, $\max L^{(m)}\|\boldsymbol{\theta}_2 - \boldsymbol{\theta}_1\|$, and it is relatively small for a single gradient step. On the other hand, for other linear relaxation-based methods using non-zero relaxed gradient approximation $\boldsymbol{g}^{(m)} \neq \boldsymbol{0}$, the gradient updates used in the training are more unstable than IBP. It is consistent with the empirical results shown in Figure 3.6 and Figure 3.7 that there are significant differences between the loss landscape of IBP and others.

3.3.3 Tightness and smoothness

Our analyses so far suggest that not only tightness of the upper bound on the worst-case loss but also smoothness of the loss landscape is important for building a certifiably robust model. Therefore, we aim to design a new certifiable training method to improve the aforementioned factors (favorable landscape and tighter bound).

Favorable landscape with smooth $\boldsymbol{g}$ via sparse $\underline{\boldsymbol{a}}$ To have a smooth loss landscape, we aim to build a linear relaxation with a smooth relaxed gradient approximation by using sparse $\underline{\boldsymbol{a}}$. For example, in a simple 2-layered network case with $\boldsymbol{s}_m(\boldsymbol{x}) = \boldsymbol{w}^T h(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) + b$, we have $\boldsymbol{g}^{(m)} = \boldsymbol{W}^T(\underline{\boldsymbol{a}} \odot \boldsymbol{w}^- + \overline{\boldsymbol{a}} \odot \boldsymbol{w}^+)$ and $\|\nabla_{\boldsymbol{w}} \boldsymbol{g}^{(m)}\| = \|\boldsymbol{W}^T \mathrm{diag}(\underline{\boldsymbol{a}} \odot \mathbf{1}[\boldsymbol{w} < 0] + \overline{\boldsymbol{a}} \odot \mathbf{1}[\boldsymbol{w} \geq 0])\| \leq \|\boldsymbol{W}^T\|(\|\underline{\boldsymbol{a}}\| + \|\overline{\boldsymbol{a}}\|)$ where $\mathrm{diag}(\boldsymbol{v})$ is the diagonal matrix whose entries are the elements of $\boldsymbol{v}$. This simple example implies that a sparse $\underline{\boldsymbol{a}}$ may help to smooth the relaxed gradient approximation.

To this end, we investigate variants of CROWN-IBP with different $\underline{\boldsymbol{a}}$ settings for unstable ReLUs to see their effects on the smoothness of the loss landscape. For each setting, we sample $\underline{a} \in \{0, 1\}$ with different (p, q) with $P(\underline{a} = 1 \mid |l| > |u|) = p$ and $P(\underline{a} = 1 \mid |l| \leq |u|) = q$ for each neuron with pre-activation bounds l and u. For example, CROWN-IBP uses $(p, q) = (0, 1)$. We use $\underline{\boldsymbol{a}} = \mathbf{1}[\boldsymbol{u}^+ + \boldsymbol{l}^- > \mathbf{0}]$ for the other stable ReLUs. We consider the variants $(p, q) = (0, 0), (0, 0.5), (0, 1), (0.5, 1)$, and $(1, 1)$, in order of decreasing the sparsity. For the other elements of the linear relaxation variable $\boldsymbol{\phi} = \{(\overline{\boldsymbol{a}}, \underline{\boldsymbol{a}}, \overline{\boldsymbol{b}}, \underline{\boldsymbol{b}})\}$, we fix $\overline{\boldsymbol{a}} = \frac{\boldsymbol{u}^+}{\boldsymbol{u}^+ - \boldsymbol{l}^-}, \overline{\boldsymbol{b}} = -\frac{\boldsymbol{u}^+ \boldsymbol{l}^-}{\boldsymbol{u}^+ - \boldsymbol{l}^-}$, and $\underline{\boldsymbol{b}} = \mathbf{0}$ for each activation node, because they are the optimal choices for tightening the bound (see Appendix C.3.2 for details). As expected, Figure 3.9 shows that it tends to have a more smooth landscape as $\underline{\boldsymbol{a}}$ becomes more sparse.

However, the sparsity only is not enough to achieve robustness unless the tightness is guaranteed. A variant of CROWN-IBP with $(p, q) = (0, 0)$ achieves a favorable landscape, but they show looser upper bounds which lead to a worse performance (details are presented in Appendix C.4). Therefore, it is required to search for appropriate $\underline{\boldsymbol{a}}$ that can achieve both tightness and favorable landscape.

Tighter bound via optimization Now, we aim to make $\underline{\boldsymbol{a}}$ sparse and to tighten the upper bound in (3.3.14), simultaneously. We can achieve both by minimizing the upper bound over the linear relaxation variable $\boldsymbol{\phi}$ as follows:

$$\mathcal{L}(\overline{\boldsymbol{s}}(\boldsymbol{x})) \geq \min_{\boldsymbol{\phi}} \mathcal{L}(\overline{\boldsymbol{s}}(\boldsymbol{x};\boldsymbol{\phi})) \geq \max_{\boldsymbol{x}' \in \mathbb{B}(\boldsymbol{x},\epsilon)} \mathcal{L}(f(\boldsymbol{x}')). \tag{3.3.23}$$

It can be equivalently understood as solving the dual optimization in CAP rather than using a dual feasible solution. However, solving the dual optimization is computationally prohibited for the linear relaxation of CAP. To resolve this problem, we use the same linear relaxation as IBP for the subnetworks of $\boldsymbol{s}$ except for $\boldsymbol{s}$ itself, similar to CROWN-IBP. Further, we efficiently compute a surrogate $\hat{\underline{\boldsymbol{a}}}$ of the minimizer $\underline{\boldsymbol{a}}^* = \arg\min_{\underline{\boldsymbol{a}}} \mathcal{L}(\overline{\boldsymbol{s}}(\boldsymbol{x};\boldsymbol{\phi} = \{(\overline{\boldsymbol{a}}, \underline{\boldsymbol{a}}, \overline{\boldsymbol{b}}, \underline{\boldsymbol{b}})\}))$ using the one-step projected gradient update of the relaxation variable $\underline{\boldsymbol{a}}$ for unstable ReLUs. Specifically, we have

$$\hat{\underline{\boldsymbol{a}}} = \Pi_{[0,1]^n}\left(\underline{\boldsymbol{a}}_0 - \eta \operatorname{sign}(\nabla_{\underline{\boldsymbol{a}}} \mathcal{L}(\overline{\boldsymbol{s}}(\boldsymbol{x};\boldsymbol{\phi}_0)))\right) \tag{3.3.24}$$

with an initial variable $\boldsymbol{\phi}_0 = \{(\overline{\boldsymbol{a}}, \underline{\boldsymbol{a}}_0, \overline{\boldsymbol{b}}, \underline{\boldsymbol{b}})\}$ where $\underline{\boldsymbol{a}}_0 \sim U[0,1]^n$ and $\eta \geq 1$, yielding the final objective $\mathcal{L}(\overline{\boldsymbol{s}}(\boldsymbol{x};\hat{\boldsymbol{\phi}}))$ where $\hat{\boldsymbol{\phi}} = \{(\overline{\boldsymbol{a}}, \hat{\underline{\boldsymbol{a}}}, \overline{\boldsymbol{b}}, \underline{\boldsymbol{b}})\}$.

The update (3.3.24) naturally leads to a sparse $\hat{\underline{\boldsymbol{a}}}$. For example, in a network with $\boldsymbol{s}_m(\boldsymbol{x}) = \boldsymbol{w}^T h(\boldsymbol{z})$ for $\boldsymbol{l} \leq \boldsymbol{z} \leq \boldsymbol{u}$, we have $\overline{\boldsymbol{s}}_m(\boldsymbol{x}) = \boldsymbol{w}^{-T}\text{diag}(\underline{\boldsymbol{a}})\boldsymbol{l} + \boldsymbol{w}^{+T}\text{diag}(\overline{\boldsymbol{a}})\boldsymbol{u}$ and $\nabla_{\underline{\boldsymbol{a}}} \overline{\boldsymbol{s}}_m(\boldsymbol{x}) = \boldsymbol{w}^- \odot \boldsymbol{l} \geq \boldsymbol{0}$ for unstable ReLUs with $\boldsymbol{l} \leq \boldsymbol{0}$. Thus, the gradient descent direction, $-\text{sign}(\nabla_{\underline{\boldsymbol{a}}} \mathcal{L}(\overline{\boldsymbol{s}}(\boldsymbol{x};\boldsymbol{\phi}_0)))$, is likely to yield a sparse $\hat{\underline{\boldsymbol{a}}}$, especially for large perturbations. This is consistent with the results shown in Figure 3.11 which will be discussed in the following section.

3.3.4 Experiments

In this section, we demonstrate the proposed method satisfies two key criteria required for building certifiably robust models: 1) tightness of the upper bound on the worst-case loss, and 2) smoothness of the loss landscape. Subsequently, we evaluate the performance of the method by comparing with others certifiable training methods. Details on the experimental settings are in Appendix C.1.

Table 3.4: Test errors (Standard / PGD / Verified error) of IBP, CROWN-IBP ($\beta = 1$), CAP, and OURS on MNIST, CIFAR-10, and SVHN. Bold and underline numbers are the first and second lowest verified error.

Data	$\epsilon_{test}(l_\infty)$	**IBP**	**CROWN-IBP** ($\beta = 1$)	**CAP**	**OURS**
MNIST	0.1	1.18 / 2.16 / 3.52	1.07 / 1.69 / **2.10**	0.80 / 1.73 / 3.19	1.09 / 1.77 / $\underline{2.36}$
	0.2	2.00 / 3.29 / $\underline{6.31}$	2.99 / 5.50 / 7.97	3.22 / 6.72 / 11.06	1.70 / 3.44 / **4.34**
	0.3	3.50 / 5.85 / $\underline{10.45}$	5.73 / 10.76 / 16.28	19.19 / 35.84 / 47.85	3.49 / 5.59 / **9.79**
	0.4	3.50 / 7.30 / $\underline{17.96}$	5.73 / 14.63 / 23.80	-	3.49 / 6.77 / **15.42**
CIFAR 10	$2/255$	37.98 / 49.40 / 55.39	32.48 / 42.77 / 50.15	28.80 / 38.95 / **48.50**	31.49 / 42.73 / $\underline{49.42}$
	$4/255$	46.42 / 57.42 / 62.80	45.56 / 58.24 / 64.47	40.78 / 52.62 / $\underline{61.88}$	42.53 / 55.55 / **61.52**
	$6/255$	52.84 / 63.92 / $\underline{68.79}$	54.72 / 65.28 / 71.04	49.20 / 60.85 / 69.03	50.19 / 61.88 / **66.90**
	$8/255$	55.71 / 66.79 / $\underline{70.95}$	61.37 / 70.66 / 75.37	56.77 / 66.78 / 73.02	56.01 / 66.17 / **69.70**
	$16/255$	67.10 / 75.12 / $\underline{78.26}$	76.65 / 81.90 / 84.42	75.11 / 80.67 / 82.07	65.93 / 75.39 / **77.87**
SVHN	0.01	19.91 / 34.12 / 43.83	17.25 / 30.84 / 39.88	16.88 / 30.16 / **37.09**	16.41 / 30.43 / $\underline{39.44}$

Tightness To validate that the proposed method (OURS) has sufficiently tight bounds, we analyze various linear relaxation methods in Figure 3.10. We define a tightness measure as a sum over the worst-case margin for each class m, $\sum_{m=0}^{c-1} \overline{\boldsymbol{s}}_m(\boldsymbol{x})$, obtained from (3.3.15). Then, we evaluate multiple methods on a single fixed model pre-trained with the proposed training method. The compared methods are, from left to right, OURS, CROWN-IBP, CAP-IBP, and RANDOM. All methods use the same IBP relaxation for subnetworks, but use different linear relaxation variables $\underline{\boldsymbol{a}}$ for the whole network $\boldsymbol{s}$. CROWN-IBP, CAP-IBP, and RANDOM use $\underline{\boldsymbol{a}} = \mathbf{1}[\boldsymbol{u}^+ + \boldsymbol{l}^- > 0]$, $\underline{\boldsymbol{a}} = \frac{\boldsymbol{u}^+}{\boldsymbol{u}^+ - \boldsymbol{l}^-}$ and $\underline{\boldsymbol{a}} \sim U[0,1]^n$, respectively. We fix the other variables $\overline{\boldsymbol{a}}, \overline{\boldsymbol{b}}$, and $\underline{\boldsymbol{b}}$, as in Section 3.3.3. In both figures, our method shows the lowest value on average, which indicates that a single gradient step in (3.3.24) is sufficient to obtain tighter bounds. See Appendix C.11 for the equivalent tightness violin plots of other models.

Smoothness Figure 3.11 shows that our method has successfully yielded more sparse $\underline{\boldsymbol{a}}$ than CROWN-IBP ($\beta = 1$). This leads to the results shown in Figure 3.7 that the proposed method has a smooth loss landscape as with IBP, whereas the others have less smooth ones.

Table 3.5: Test errors (Standard / Verified error) compared to the best errors reported in the literature. Bold numbers are the lowest verified error. The results in RS [74] and COLT [3] are evaluated with a MILP based exact verifier [65].

Data	**MNIST**		**CIFAR-10**	
$\epsilon_{test}(l_\infty)$	0.1	0.3	$2/255$	$8/255$
RS	1.32 / 4.87	2.67 / 19.32	38.88 / 54.07	59.55 / 79.72
DiffAI	1.3 / 4.2	3.4 / 10.7	37.7 / 54.5	53.8 / 72.8
COLT	0.8 / 2.9	2.7 / 14.3	21.6 / **39.5**	48.3 / 72.5
OURS	1.09 / **2.28**	2.42 / **7.84**	31.49 / 49.42	56.01 / **69.70**

Certified Robustness We evaluate the performance of the proposed method and compare it to that of state-of-the-art certifiable training methods: IBP [27], CROWN-IBP ($\beta = 1$) [78], and CAP [72], as in Section 3.3.2. On MNIST, we follow [78] and use $\epsilon_{train} \geq \epsilon_{test}$; whereas for CAP, we use the same $\epsilon_{train} = \epsilon_{test}$ which yields better results. We used three evaluation metrics: standard (clean) error, 100-step PGD error, and verified error. For the verified error, we evaluated with the bound $\overline{\boldsymbol{s}}$ of each method.

Table 3.4 summarizes the evaluation results under different ϵ_{test} for each dataset. In general, when ϵ_{test} is low, methods with tighter linear relaxations show good performance, whereas IBP tends to perform better as ϵ_{test} increases. In short, the current state-of-the-art methods perform well for a specific range of ϵ_{test}. For example, IBP shows relatively better performance in the case of $\epsilon_{test} = 0.3, 0.4$ on MNIST and $\epsilon_{test} = 6/255, 8/255, 16/255$ on CIFAR-10. On the other hand, CAP and CROWN-IBP ($\beta = 1$) outperform IBP in the case of $\epsilon_{test} = 0.1$ on MNIST, $\epsilon_{test} = 2/255$ on CIFAR-10 and $\epsilon_{test} = 0.001$ on SVHN. This result is consistent with the analysis shown in Figure 3.6 that CAP and CROWN-IBP ($\beta = 1$) have lower loss than IBP at small ϵ, but their loss landscape is less smooth than IBP, leading to worse performance at large ϵ. Moreover, CAP cannot be trained on MNIST when $\epsilon_{train} = 0.4$. As the case is also not specified in [72], it seems that CAP is hard to be robust to $\epsilon_{train} \geq 0.4$. On the other hand, the proposed method shows consistent performance in a wide range of ϵ_{test} values, achieving the best performance in

Table 3.6: Test errors (Standard / PGD / Verified error) of OURS and CROWN-IBP$_{1\to0}$ on CIFAR-10. Bold numbers indicate the lower error.

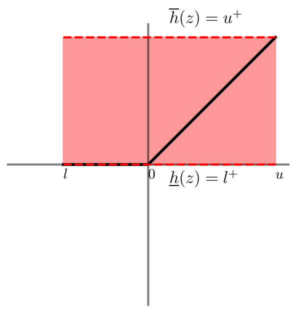
$\epsilon_{test}(\ell_\infty)$	**OURS**	**CROWN-IBP$_{1\to0}$**
$2/255$	**31.49** / **42.73** / **49.42**	36.30 / 47.13 / 52.70
$4/255$	**42.53** / **55.55** / **61.52**	45.92 / 57.58 / 62.07
$6/255$	**50.19** / **61.88** / **66.90**	53.54 / 64.14 / 67.35
$8/255$	56.01 / **66.17** / **69.70**	**55.09** / 66.68 / 69.97
$16/255$	**65.93** / **75.39** / **77.87**	66.62 / 76.13 / 77.88

most cases, since it has tighter bounds and a favorable landscape, not overfitting to a local minimum during the ϵ_t-scheduling. We also conduct additional experiments on the hyperparameters in Appendix C.7, C.8, and C.9.

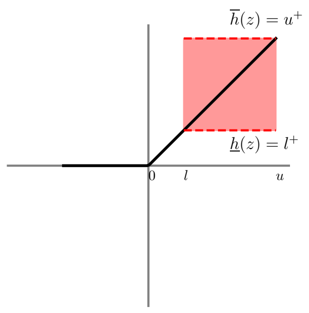
We also compared our method with other prior work, ReLU Stability (RS) [74], DiffAI [54] and COLT [3], in Table 3.5. All experiments and results, except for Table 3.5, in this chapter are based on our own reimplementation. The proposed method shows the best verified error, except for the case $\epsilon_{test} = 2/255$ on CIFAR-10, in which COLT performs better. This is because COLT is based on the linear relaxation used in CAP.

Unlike standard training, certifiable training requires ϵ_t-scheduling. It is implicitly assumed that a set of weights that makes the network robust to a small ϵ_t is a good initial point to learn robustness to a large ϵ_{train}. However, linear relaxation-based methods with tighter bounds start with a lower loss at a small ϵ_t, but with an unfavorable loss landscape, they cannot explore a sufficiently large area of the parameter space. Hence, they overfit to be robust to a small perturbation, and cannot generalize to a large perturbation. CAP and CROWN-IBP ($\beta = 1$) are typical examples that demonstrate the overfitting. This may overegularize the weight norm and decrease the model capacity [72, 78]. The tightness of the proposed method improves the performance for a small ϵ_{train}, while the smoothness of the proposed method helps the optimization process, which also leads to better performance for a large ϵ_{train}. To conclude, the proposed method can achieve a decent performance under a wide range of perturbations.

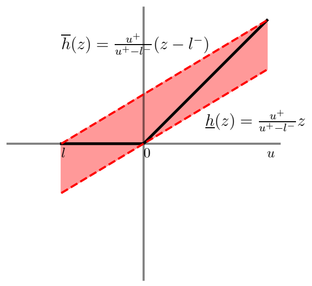
Understanding β-scheduling For CROWN-IBP, we use two different settings of β, CROWN-IBP$_{1\to1}$ and CROWN-IBP$_{1\to0}$, where the subscript $\beta_{start} \to \beta_{end}$ refers to the linear scheduling on β from β_{start} to β_{end}. [78] found that the β-scheduling of CROWN-IBP$_{1\to0}$ could help to improve the robustness performance. And they argued that this is because training with a tighter bound of CROWN-IBP at the beginning can provide a good initialization for later IBP training. On the other hand, we provide another explanation that CROWN-IBP$_{1\to0}$ starts with a tighter bound (CROWN-IBP only) but not overfits to small perturbation by gradually introducing the IBP objective which has a smoother landscape. Despite using a single objective without the mixture parameter β, the proposed method outperforms or is comparable to CROWN-IBP$_{1\to0}$ on CIFAR-10 as shown in Table 3.6.



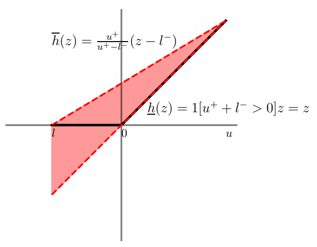
(a) IBP



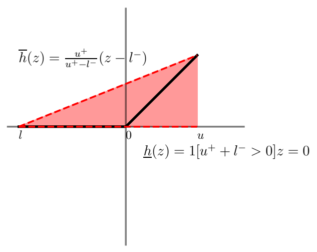
(b) IBP $(u, l > 0)$


$\overline{h}(z) = \frac{u^+}{u^+ - l^-}(z - l^-)$
$\underline{h}(z) = \frac{u^+}{u^+ - l^-}z$
l
0
u

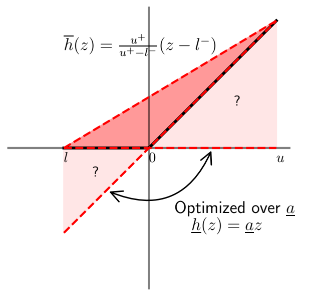
(c) CAP


$\overline{h}(z) = \frac{u^+}{u^+ - l^-}(z - l^-)$
$\underline{h}(z) = 1[u^+ + l^- > 0]z = z$
l
0
u

(d) CROWN-IBP $(u^+ + l^- > 0)$


$\overline{h}(z) = \frac{u^+}{u^+ - l^-}(z - l^-)$
l
0
u
$\underline{h}(z) = 1[u^+ + l^- > 0]z = 0$

(e) CROWN-IBP $(u^+ + l^- \leq 0)$


$\overline{h}(z) = \frac{u^+}{u^+ - l^-}(z - l^-)$
?
l
0
u
?
Optimized over $\underline{a}$
$\underline{h}(z) = \underline{a}z$

(f) The proposed method

Figure 3.5: Illustrations of linear relaxation methods. Except for (b), they illustrate the relaxations when $l \leq 0 \leq u$ (Unstable ReLU). (b) Illustration of the relaxation of IBP when $u, l > 0$ (Active ReLU).

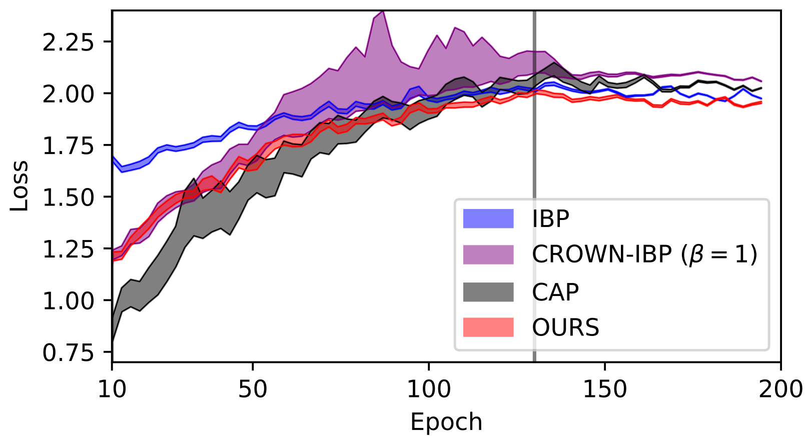


Figure 3.6: The learning curves for the scheduled value of ϵ_t with the loss variation along the gradient descent direction. The vertical line indicates when the ramp-up ends. IBP starts with a higher loss but ends with a relatively lower loss (error), demonstrating smaller loss variations. Our method uses tight bounds like CROWN-IBP, while its landscape is as favorable as IBP, achieving the best performance among these four methods (see together with Table 3.3).

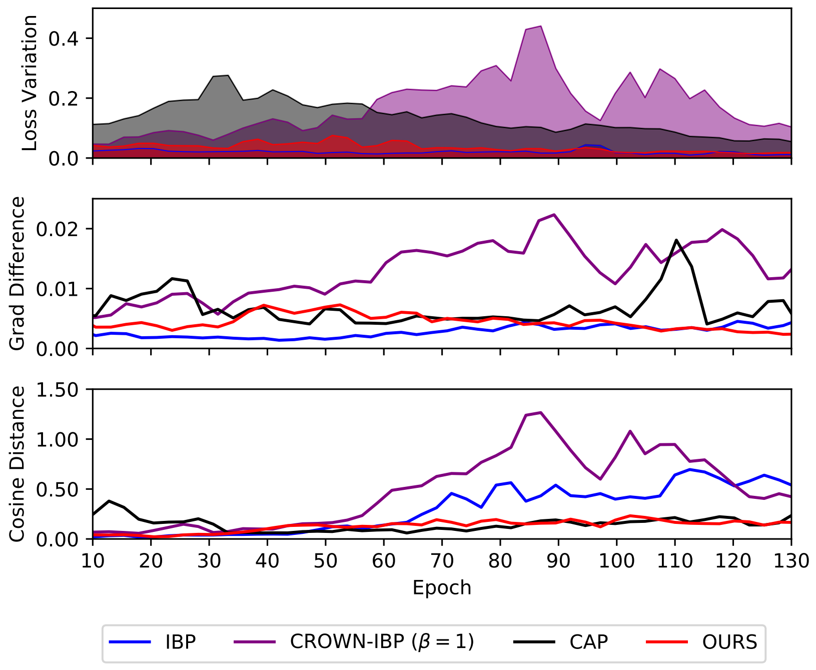


Figure 3.7: Non-smoothness measures of the loss landscape between the certifiable training methods. Lower is better. (Top) Loss variations along the gradient descent direction, (Middle) ℓ_2-distance between two consecutive loss gradients and (Bottom) the cosine distance between them during the ramp-up phase. Note that they are evaluated with the scheduled value of ϵ_t.

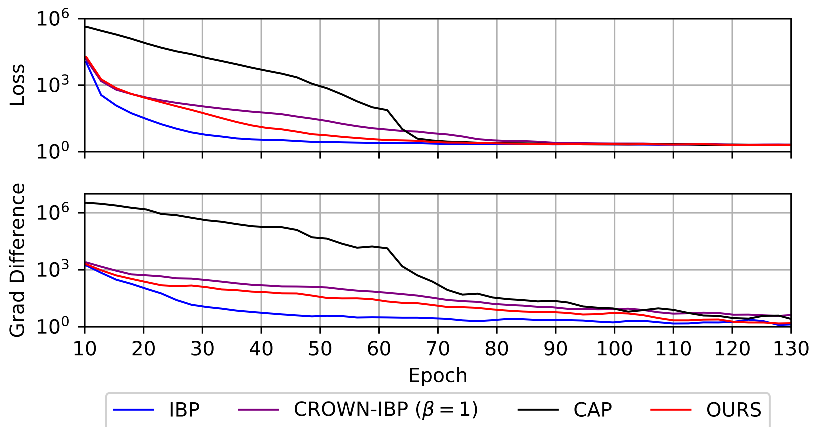


Figure 3.8: (Top) The learning curves for the target perturbation and (Bottom) the loss gradient difference in (3.3.19) for the target perturbation.

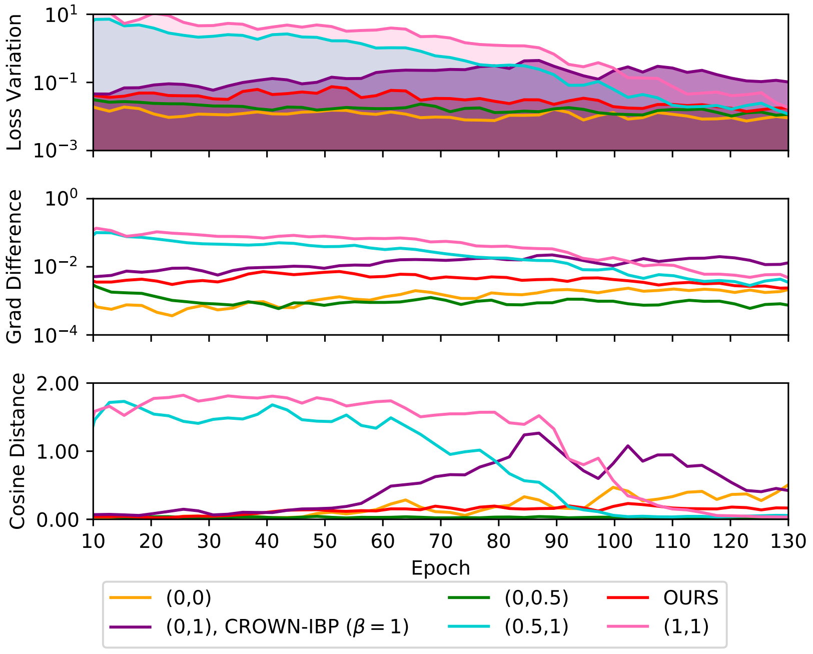


Figure 3.9: The non-smoothness measures of the models with different levels of sparsity of $\underline{\boldsymbol{a}}$ during the ramp-up period (same as in Figure 3.7). Notation (p, q) denotes the variant with sampling $\underline{a} \in \{0, 1\}$ with $P(\underline{a} = 1 \mid |l| > |u|) = p$ and $P(\underline{a} = 1 \mid |l| \leq |u|) = q$ for unstable ReLUs. As $\underline{\boldsymbol{a}}$ becomes sparse, the loss landscape is much smoother.

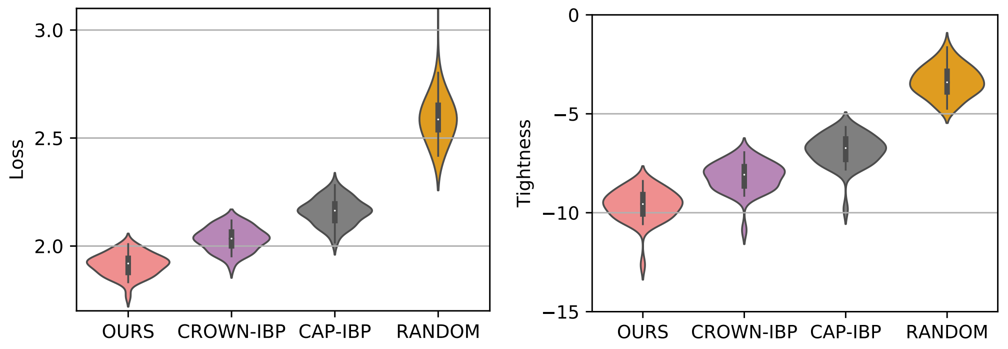


Figure 3.10: Violin plots of the test loss (*Left*) and of tightness (*Right*) for various linear relaxations. Lower is better. This shows that the proposed relaxation method has a tighter bound than the others.

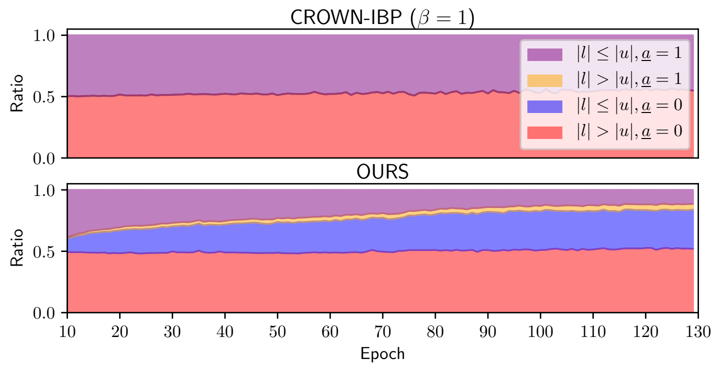


Figure 3.11: The configuration of $\underline{a}$ used for unstable ReLUs during the ramp-up period. It indicates that the proposed method reduces the number of nonzero $\underline{a}$ (purple+yellow).

Chapter 4

Conclusion and Open Problems

In this dissertation, we studied how to build a robust model against adversarial attacks.

In Chapter 2, we first focused on heuristic defense methods based on randomized neural networks, covering the work done in [43]. Randomized defenses are potentially promising defense methods against adversarial attacks. Unfortunately, previous attempts have been broken by the EOT attack. We investigated the effect of EOT on the randomized neural networks and demonstrated that the EOT attack is less effective when the gradients are more scattered. Based on the analysis, we proposed GradDiv regularizations that penalize the gradients concentration to confuse the adversary about the true gradient. We show that GradDiv improves the robustness of randomized neural networks against the EOT attack. We hope the future work on randomized defenses can adopt GradDiv to further improve their performance.

In Chapter 3, we moved to the certified defense framework. In Section 3.2, we focused on the tightness of the upper bound on the worst-case loss over valid adversarial perturbations, covering the work done in [44]. To obtain a tight outer bound, we proposed BCP that efficiently computes box constraints which can tighten the outer bound. Then, we trained a certifiably robust model by minimizing the certificate loss based on the worst-translated logit over the tight outer bound. By doing so, we can build the first certifiable robust model on Tiny ImageNet under the ℓ_2-perturbation. We hope that our method can serve as a strong benchmark for certifiable training on a large-scale dataset.

In Section 3.3, we have investigated the loss landscape of certifiable training and found that the smoothness of the loss landscape is an important factor that influences in building certifiably robust models, covering the work done in [45]. To this end, we proposed a method that satisfies the two criteria: tightness of the upper bound on the worst-case loss and smoothness of the loss landscape. Then, we empirically demonstrated that the proposed method outperforms the current state-of-the-art methods under a wide range of perturbations. We believe that with a better understanding of the loss landscape, better certifiably robust models can be built.

However, it is still far from reaching human performance. The state-of-the-art robust accuracy evaluated against AutoAttack [16] ($\ell_\infty = {}^{8}/_{255}$) is about 66-67% on CIFAR-10 [60]. Moreover, previous work mainly focused on the restrictive threat model such as ℓ_p-bounded perturbations. It is required to consider more realistic perturbations [39].

Appendix A

Appendix for 2.2

A.1 Experimental Settings

A.1.1 Network Architectures

In this section, we provide the details of the network architectures used in the experiments. We denote the convolutional layer with the output channel c, the kernel k, the stride s as C(c, k, s) (or C(c, k, s, p) if it uses the padding p), the linear layer with the output channel c as F(c), the maxpool layer and the average pool layer with kernel size 2 and stride 2 as MP and AP, respectively, and the batch normalization layer [34] as BN. We also denote ReLU layer and Leaky ReLU layer as ReLU and LReLU, respectively. Note that we omit the flatten layer before the first linear layer.

- MNIST: C(32,3,1,1)-LReLU-C(64,3,1,1)-LReLU-MP-C(128,3,1,1) -LReLU-MP-F(1024)-LReLU-F(10)
- CIFAR10: VGG-16
- STL10: C(32,3,1,1)-BN-ReLU-MP-C(64,3,1,1)-BN-ReLU-MP -C(128,3,1,1)-BN-ReLU-MP-C(256,3,1,1) -BN-ReLU-MP-C(256,3,1)-BN-ReLU-C(512,3,1)-BN-ReLU-AP-F(10)

A.1.2 Batch-size, Training Epoch, Learning rate decay, Warm-up, and Ramp-up periods

Table A.1 summarizes the details of the training parameters. We use the Adam optimizer for all datasets.

A.2 Variants of GradDiv-mean (2.2.17)

As mentioned in the main paper, we also tested variants of GradDiv-mean (2.2.17). We propose two variants, $R_{max}(X;\theta)$ and $R_{smoothmax}(X;\theta)$. $R_{max}(X;\theta)$ uses the maximum value of the cosine similarity instead of the mean value: $R_{max}(X;\theta) = \max\{\cos(g_i, g_j)\}_{i\neq j}$, and $R_{smoothmax}(X;\theta)$ uses the smooth maximum (LogSumExp): $R_{smoothmax}(X;\theta) = \log\{\sum_{i\neq j} \exp\cos(g_i, g_j)\}$.

Figure A.1 shows the effects of the variants on each objective and Figure A.2 shows the test accuracy against EOT-PGD on MNIST ($n = 20$, $\alpha = \epsilon/4$, and $m = 20$). There is no significant performance difference among the regularizers.

A.3 Additional Results on "Effects of GradDiv during Training"

Figure A.3 and A.4 show the effects of GradDiv on the concentration measures (2.2.16), (2.2.17), and (2.2.19) during training on STL10 and CIFAR10, respectively. We observed similar results with Figure 2.3.

Table A.1: The details of the training parameters. The learning rate is decayed at the epochs listed in the lr decay column with decay rate 0.1.

Datasets	batch-size	epoch	learning rate (initial)	lr decay	warm-up	ramp-up
MNIST	128	60	0.001	[30]	3	20
STL10		120		[60,100]	6	60
CIFAR10		200		[80,140,180]	6	90

A.4 Additional Results on Table 2.1

In Table A.2 and A.3, we also present additional results as Table 2.1 in the main paper for the other datasets, MNIST and STL10.

A.5 In the case of $n > 20$ in Figure 2.7

As mentioned in the main paper, we use the gradient sample size $n = 20$ in Figure 2.7. Figure A.5 shows accuracy when different gradient sample sizes n are used. Note that when $n > 20$ the EOT attack on Adv-BNN+GradDiv becomes even weaker as n increases. Therefore, we reported the worst-case results when $n = 20$. In the experiment, we use the EOT attack with $\epsilon = 0.38$, $\alpha = \epsilon/4$ and the attack iteration $m = 20$ on MNIST.

A.6 RSE [48] as a baseline

In the main paper, we use Adv-BNN [49] as a baseline and show increase of robustness with our regularizers. In addition to this, we test the proposed method on RSE [48] as a new baseline and demonstrate the results in Table A.5. In the experiment, RSE using (GradDiv-κ, 1) outperforms the baseline on CIFAR10.

Table A.2: The test accuracy against adversarial attacks on MNIST. The best results are highlighted in bold.

Dataset	Attack		Method	0	0.1	0.2	0.3	0.32	0.34	0.36	0.38	0.4
MNIST	EOT-FGSM		None	99.41	92.48	61.24	27.3	23.66	20.67	18.5	17.08	15.95
			Adv.train [51]	99.4	98.7	97.95	97.42	96.69	93.22	86.47	77.05	62.74
			RSE [48]	**99.44**	96.08	82.52	47.76	40.11	33.73	28.37	23.71	20.03
			Adv-BNN [49]	99.38	**98.85**	**98.18**	**97.23**	**97.28**	**97.11**	**96.28**	92.82	85.02
			Adv-BNN [49]+GradDiv	99.14	98.43	97.39	96.03	95.46	95.02	94.18	**93.57**	**92.32**
	EOT -PGD	-	None	-	77.87	5.46	1.12	1.06	1.05	1.05	1.05	1.05
			Adv.train [51]	-	98.53	97.16	94.85	86.41	50.26	26.57	12.44	3.48
		$n = 5$	RSE [48]	-	94.71	47.55	1.63	0.82	0.47	0.38	0.34	0.33
			Adv-BNN [49]	-	**98.8**	**97.65**	**94.32**	**90.62**	74.83	28.55	5.18	0.15
			Adv-BNN [49]+GradDiv	-	98.27	96.33	90.12	86.67	**80.23**	**68.34**	**47.24**	**21.37**
		$n = 10$	RSE [48]	-	94.37	41.39	1.03	0.51	0.38	0.3	0.22	0.25
			Adv-BNN [49]	-	**98.76**	**97.59**	**93.88**	**90.07**	73.87	25.38	3.26	0.05
			Adv-BNN [49]+GradDiv	-	98.19	96.08	89.21	85.36	**78.34**	**64.88**	**43.07**	**17.31**
		$n = 20$	RSE [48]	-	94.23	37.25	0.7	0.44	0.27	0.27	0.19	0.15
			Adv-BNN [49]	-	**98.74**	**97.43**	**93.83**	**90**	73.62	25.82	2.33	0.02
			Adv-BNN [49]+GradDiv	-	97.99	95.8	88.73	84.54	**77.58**	**63.34**	**40.05**	**15.57**

Table A.3: The test accuracy against adversarial attacks on STL. The best results are highlighted in bold.

Dataset	Attack		Method	0	0.01	0.02	0.03	0.04	0.05	0.06	0.07
STL10	EOT-FGSM		None	**75.69**	31.08	16.35	10.5	8.05	7.14	6.9	6.66
			Adv.train [51]	62.43	48.53	37.7	29.22	22.96	18.16	15.1	12.76
			RSE [48]	73.65	47.86	26.29	13.01	7.18	4.43	2.9	1.99
			Adv-BNN [49]	54.85	48.38	41.64	35.24	30.21	25.4	20.46	17.14
			Adv-BNN [49]+GradDiv	60.31	**56.39**	**51.76**	**47.94**	**43.84**	**40.54**	**35.93**	**32.62**
	EOT -PGD	-	None	-	18.35	2.2	0.11	0.01	0	0	0
			Adv.train [51]	-	47.84	34.94	25.18	17.51	11.91	7.76	4.74
		$n = 5$	RSE [48]	-	44.99	21.31	8.55	3.39	1.18	0.34	0.08
			Adv-BNN [49]	-	47.4	40.49	33.63	27.18	21.08	16.48	11.21
			Adv-BNN [49]+GradDiv	-	**54.8**	**48.6**	**42.43**	**36.49**	**30.49**	**25.28**	**21.04**
		$n = 10$	RSE [48]	-	44.33	20.38	8.1	3.06	0.99	0.26	0.05
			Adv-BNN [49]	-	47.64	39.98	32.41	25.69	19.35	13.84	9.74
			Adv-BNN [49]+GradDiv	-	**54.19**	**47.14**	**40.75**	**34.56**	**28.15**	**22.79**	**17.75**
		$n = 20$	RSE [48]	-	43.65	19.64	7.69	2.93	0.88	0.25	0.04
			Adv-BNN [49]	-	47.16	38.99	31.65	24.59	18.53	13.06	9.05
			Adv-BNN [49]+GradDiv	-	**53.36**	**46.18**	**38.94**	**32.19**	**26.23**	**19.98**	**15.05**

Table A.4: The test accuracy against adversarial attacks on CIFAR10. The best results are highlighted in bold.

Dataset	Attack		Method	0	0.01	0.02	0.03	0.04	0.05	0.06	0.07
CIFAR10	EOT-FGSM		None	**92.39**	26.57	12.06	8.37	7.18	6.83	6.81	6.96
			Adv.train [51]	78.83	68.12	57.65	49.63	42.72	37.38	33.14	29.26
			RSE [48]	84.06	68.49	51.83	35.67	21.6	12.9	7.08	4.12
			Adv-BNN [49]	75.97	67.94	59.28	49.9	41.59	33.91	27.47	22.1
			Adv-BNN [49]+GradDiv	76.45	**71.37**	**65.65**	**59.02**	**53.69**	**47.71**	**42.72**	**37.69**
	EOT -PGD	-	None	-	3.16	0	0	0	0	0	0
			Adv.train [51]	-	66.75	52.84	39.49	28.32	18.26	11.47	7.23
		$n = 5$	RSE [48]	-	66.39	44.5	23.86	10.48	3.98	1.4	0.44
			Adv-BNN [49]	-	67.57	58.06	46.81	36.13	25.15	16.24	9.18
			Adv-BNN [49]+GradDiv	-	**69.81**	**61.88**	**53.63**	**44.52**	**36.4**	**28.52**	**20.91**
		$n = 10$	RSE [48]	-	65.95	43.1	22.4	9.43	3.24	1.19	0.37
			Adv-BNN [49]	-	67.41	56.85	45.46	33.73	23.04	13.81	7.31
			Adv-BNN [49]+GradDiv	-	**68.59**	**59.31**	**49.17**	**38.96**	**29.53**	**20.74**	**13.56**
		$n = 20$	RSE [48]	-	65.48	42.12	21.65	9.02	3.18	1.01	0.3
			Adv-BNN [49]	-	**67.17**	56.36	44.49	32.75	21.5	12.27	6.29
			Adv-BNN [49]+GradDiv	-	66.39	**56.73**	**44.88**	**33.81**	**22.95**	**14.52**	**8.6**

Table A.5: The test accuracy against adversarial attacks on the baseline, RSE [48] on CIFAR10. The best results are highlighted in bold.

Dataset	Attack		Method	0	0.01	0.02	0.03	0.04	0.05	0.06	0.07
CIFAR10	EOT-FGSM		RSE [48]	84.06	68.49	51.83	35.67	21.6	12.9	7.08	4.12
			RSE [48]+GradDiv	**84.88**	**77.77**	**69.42**	**59.88**	**50.1**	**41.53**	**33.71**	**28.52**
	EOT -PGD	$n = 1$	RSE [48]	-	68.76	48.15	28.07	14.06	5.84	2.29	0.77
			RSE [48]+GradDiv	-	**72.05**	**54.75**	**36.95**	**21.83**	**11.4**	**6.09**	**3.12**
		$n = 5$	RSE [48]	-	66.39	44.5	23.86	10.48	3.98	1.4	0.44
			RSE [48]+GradDiv	-	**68.49**	**47.9**	**27.69**	**12.83**	**5.57**	**2.38**	**1.11**
		$n = 10$	RSE [48]	-	65.95	43.1	22.4	9.43	3.24	1.19	0.37
			RSE [48]+GradDiv	-	**67.61**	**44.91**	**24.53**	**10.46**	**4.12**	**1.69**	**0.73**
		$n = 20$	RSE [48]	-	65.48	42.12	21.65	9.02	3.18	1.01	0.3
			RSE [48]+GradDiv	-	**67.07**	**44.88**	**25.43**	**11.60**	**5.20**	**2.59**	**1.46**

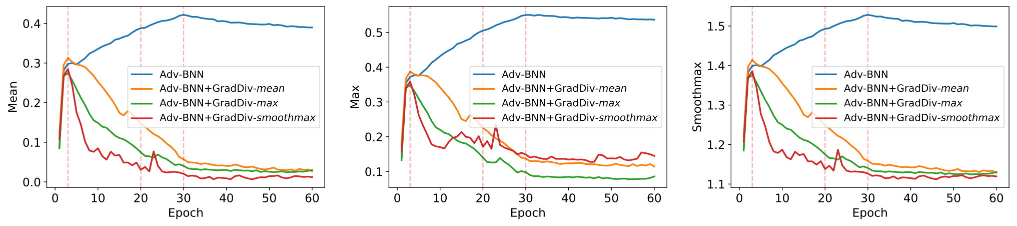


Figure A.1: The change in the additional concentration measures during training. The two leftmost vertical lines in each graph indicate the end of the warm-up and ramp-up periods, and the third vertical line indicates when the learning rate has decayed.

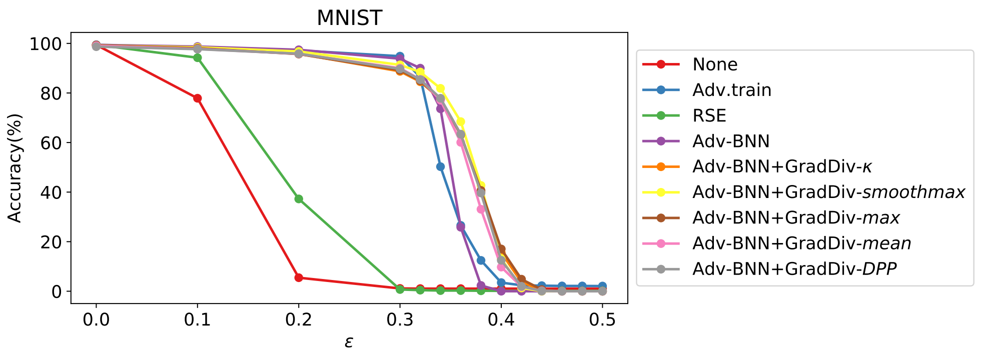


Figure A.2: The test accuracy against the EOT attack on MNIST with the proposed regularizers.

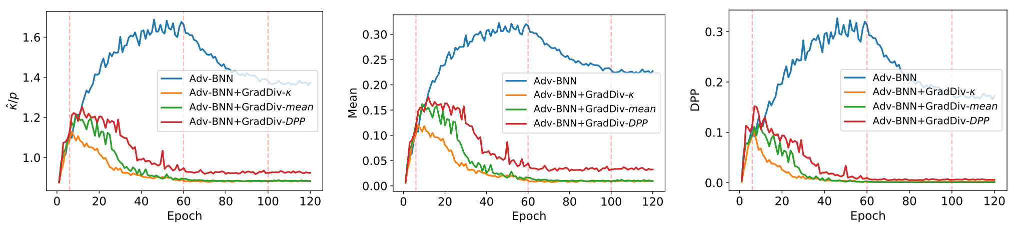


Figure A.3: The change in the concentration measures (2.2.16) (left), (2.2.17) (middle), and (2.2.19) (right) during training on STL10. The two leftmost vertical lines in each graph indicate the end of the warm-up and ramp-up periods, and the two rightmost vertical lines indicate when the learning rate has decayed.

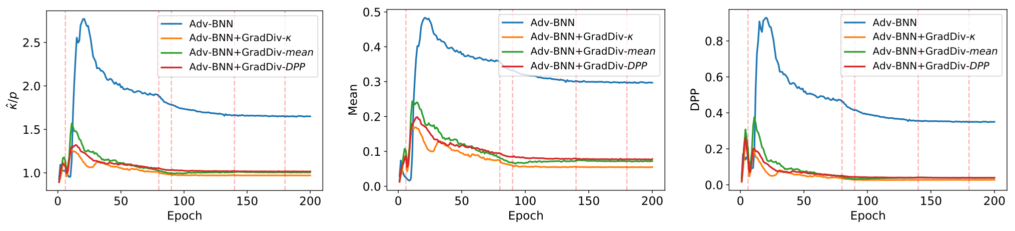


Figure A.4: The change in the concentration measures (2.2.16) (left), (2.2.17) (middle), and (2.2.19) (right) during training on CIFAR10. The two leftmost vertical lines in each graph indicate the end of the warm-up and ramp-up period, and the three rightmost vertical lines indicate when the learning rate has decayed.

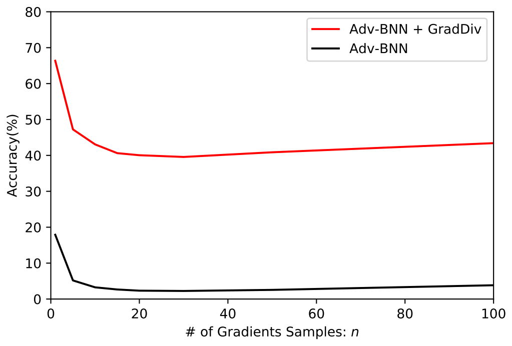


Figure A.5: The effectiveness of EOT with different gradient sample size n.

Appendix B

Appendix for 3.2

B.1 The proof of the proposition 3.1.1

Proposition 3.1.1. *[[70]] For an outer bound $\hat{z}(\mathbb{B}(\boldsymbol{x})) \supset z(\mathbb{B}(\boldsymbol{x}))$ and its corresponding worst-translated logit $\underline{z}(\boldsymbol{x})$, the following inequality holds:*

$$\max_{\boldsymbol{\zeta} \in \hat{z}(\mathbb{B}(\boldsymbol{x}))} \mathcal{L}(\boldsymbol{\zeta}, y) \leq \mathcal{L}(\underline{z}(\boldsymbol{x}), y), \tag{3.1.3}$$

where $\mathcal{L}$ is the cross-entropy loss function.

Proof.

$$\begin{aligned}
&\max_{\boldsymbol{\zeta} \in \hat{z}(\mathbb{B}(\boldsymbol{x}))} \mathcal{L}(\boldsymbol{\zeta}, y) \\
&= \log\Big(1 + \max_{\boldsymbol{\zeta} \in \hat{z}(\mathbb{B}(\boldsymbol{x})))} \sum_{k \neq y} \exp\left(-(\zeta_y - \zeta_k)\right)\Big) \\
&\leq \log\Big(1 + \sum_{k \neq y} \max_{\boldsymbol{\zeta} \in \hat{z}(\mathbb{B}(\boldsymbol{x})))} \exp\left(-(\zeta_y - \zeta_k)\right)\Big) \\
&= \log\Big(1 + \sum_{k \neq y} \exp\Big(-\min_{\boldsymbol{\zeta} \in \hat{z}(\mathbb{B}(\boldsymbol{x}))} (\zeta_y - \zeta_k)\Big)\Big) \\
&= \mathcal{L}(\underline{z}(\boldsymbol{x}), y)
\end{aligned}$$

□

B.2 Outer Bound Propagation

In this section, we present intuition behind the design of BCP and the deferred explanation for calculating outer bound propagation such as layer-wise Lipschitz constant, propagated circumscribed box, and extension to a residual layer. We further provide complexity analysis on BCP.

B.2.1 Intuition behind BCP

BCP provides a tight outer bound that addresses the overestimation problems of the global Lipschitz constant. An outer bound computed by the global Lipschitz constant is highly overestimated because of the following reasons. First, it is overestimated when propagating through a linear layer. For a linear layer (including convolutional, average pooling, and normalization layers), the layer-wise Lipschitz constant is the maximum eigenvalue of the weight matrix (the factor by which the corresponding eigenvector is scaled); thus, it overestimates stretching along directions except for the corresponding eigenvector. On the other hand, with BCP, the box constraint bound $out_\infty(h^{(k+1)}(\mathbb{B}_2^{(k)}))$ calculates the radius vector $\hat{\mathbf{r}}^{(k+1)}$ by considering scaling along all basis axes. Second, the outer bound is overestimated because of ReLU layers. After propagating a ball $\mathbb{B}_2(\boldsymbol{\mu}, \rho)$ through a ReLU layer, we can estimate the propagated outer bound with a new ball $\mathbb{B}_2(\boldsymbol{\mu}^+, \rho)$ where $\boldsymbol{\mu}^+ = \max(\boldsymbol{\mu}, 0)$. However, the true image $\text{ReLU}(\mathbb{B}_2(\boldsymbol{\mu}, \rho))$ has no negative elements. In the case of BCP, the box constraint $IA(\mathbb{B}_\infty^{(k)})$ tightens the image of $\mathbb{B}_\infty^{(k)}$ in the ReLU layers by cutting off the negative regions.

B.2.2 Power iteration algorithm

In our algorithm, we apply the power iteration to compute the layer-wise Lipschitz constants efficiently. The obtained Lipschitz constants are used to compute the layer-wise outer bounds, $(\mathbb{B}_2^{(k)}, \mathbb{B}_\infty^{(k)})$. We run 1 iteration per batch during training as mentioned in [21], which is enough since SGD only makes small updates to $\mathbf{W}$ and its spectral norm. On the other hand, we run the power iteration until convergence for inference. Note that once we get the layer-wise Lipschitz constants of the trained model, we don't have to run the iteration again.

In Algorithm 2, we provide the well-known power iteration method which calculates Lipschitz constant for a linear layer to make the main paper self-contained.

Algorithm 2 Power iteration.

```
Input: weight $\mathbf{W}$, initial value $\boldsymbol{u}$, maximum iteration $n$
Output: spectral norm $\sigma$
Initialize u with a random vector with the same shape of the input for the
linear layer if $u$ is not given.
$i \leftarrow 0$
repeat
    $\boldsymbol{v} \leftarrow \mathbf{W}\boldsymbol{u}/\|\mathbf{W}\boldsymbol{u}\|_2$
    $\boldsymbol{u} \leftarrow \mathbf{W}^T\boldsymbol{v}/\|\mathbf{W}^T\boldsymbol{v}\|_2$
    $i \leftarrow i + 1$
until it converges or $i \geq n$
$\sigma \leftarrow \boldsymbol{v}^T\mathbf{W}\boldsymbol{u}$
```

In case of a convolutional layer, we used the revised version of the power iteration method in [21]. Algorithm 3 illustrates the power iteration algorithm for a convolutional layer. We denote the conv_transpose operation as $conv^T$.

Algorithm 3 Convolutional Power iteration [21].

```
Input: Convolutional weight $\mathbf{W}$, initial value $\boldsymbol{u}$, maximum iteration $n$
Output: spectral norm $\sigma$
Initialize u with a random vector with the same shape of the input for the
convolutional layer if $u$ is not given.
$i \leftarrow 0$
repeat
    $\boldsymbol{v} \leftarrow conv(\mathbf{W}, \boldsymbol{u})/\|conv(\mathbf{W}, \boldsymbol{u})\|_2$
    $\boldsymbol{u} \leftarrow conv^T(\mathbf{W}, \boldsymbol{v})/\|conv^T(\mathbf{W}, \boldsymbol{v})\|_2$
    $i \leftarrow i + 1$
until it converges or $i \geq n$
$\sigma \leftarrow \boldsymbol{v} \cdot conv(\mathbf{W}, \boldsymbol{u})$
```

B.2.3 The circumscribed box $out_\infty(h^{(k+1)}(\mathbb{B}_2^{(k)}))$

In Section 3.2.1, we explained the computation of the circumscribed box $out_\infty(h^{(k+1)}(\mathbb{B}_2^{(k)}))$ for the affine transformation $h^{(k+1)}$. In this section, we extend it to nonlinear case with a general proof. For ease of explanation, we re-write the equation (3.2.6) for the affine case as follows:

$$\begin{aligned} &out_\infty(h^{(k+1)}(\mathbb{B}_2^{(k)})) = \text{midrad}(\hat{\mathbf{m}}^{(k)}, \hat{\mathbf{r}}^{(k)}) \text{ s.t.} \\ &\hat{\mathbf{m}}^{(k)} = h^{(k+1)}(\boldsymbol{\mu}^{(k)}),\ \hat{r}_i^{(k)} = \|\mathbf{W}_{i,:}^{(k+1)}\|_2\, \rho^{(k)}. \end{aligned} \tag{B.2.1}$$

Proof. It is clear that $\hat{\mathbf{m}}^{(k)} = h^{(k+1)}(\boldsymbol{\mu}^{(k)})$. The desired radius $\hat{r}_i^{(k)}$ along i-th axis is the maximum of the inner product $\boldsymbol{e}_i^T \mathbf{W}^{(k+1)} \boldsymbol{x} = \mathbf{W}_{i,:}^{(k+1)} \boldsymbol{x}$, where $\|\boldsymbol{x}\|_2 \le \rho^{(k)}$ and $\boldsymbol{W}_{i,:}^{(k+1)}$ is the i-th row of the weight matrix $\mathbf{W}^{(k+1)}$ of the linear function $h^{(k+1)}$. Therefore, $\mathbf{W}_{i,:}^{(k+1)} \boldsymbol{x} \le \|\mathbf{W}_{i,:}^{(k+1)}\| \|\boldsymbol{x}\|$, and we finally get the radius $\hat{r}_i^{(k)} = \|\mathbf{W}_{i,:}^{(k+1)}\|\, \rho^{(k)}$. □

In the case of nonlinear activation function σ, we can derive the results $\hat{\mathbf{m}}^{(k)} = \sigma(\boldsymbol{\mu}^{(k)}), \hat{r}_i = L(\sigma_i)\rho^{(k)}$ with the Lipschitz constant $L(\sigma_i)$ of the nonlinear function σ_i, because the desired radius $\hat{r}_i$ is the maximum of the inner product $\boldsymbol{e}_i^T \sigma(\boldsymbol{x}) = \sigma_i(\boldsymbol{x})$, where $\|\boldsymbol{x}\|_2 \le \rho^{(k)}$. In the case of ReLU activation, we have $\hat{r}_i = \rho^{(k)}$ since the Lipschitz constant $L(\sigma_i)$ is 1.

B.2.4 BCP through residual layers

Our method can be applied to a wide range of network architectures including residual networks. In this section, we consider an operation through the residual layer as $h^{(k+1)}(\boldsymbol{x}) = f^{(k+1)}(\boldsymbol{x}) + g^{(k+1)}(\boldsymbol{x})$ for some functions $f^{(k+1)}$ and $g^{(k+1)}$. Therefore, we propagate the pair $(\mathbb{B}_2^{(k)}, \mathbb{B}_\infty^{(k)})$ through $f^{(k+1)}$ and $g^{(k+1)}$ independently, and obtain two pairs $(\mathbb{B}_2^{(k+1),1}, \mathbb{B}_\infty^{(k+1),1})$ and $(\mathbb{B}_2^{(k+1),2}, \mathbb{B}_\infty^{(k+1),2})$, respectively. We denote $\mathbb{B}_2^{(k+1),i} = \mathbb{B}_2(\mathbf{z}^{(k+1),i}, \rho^{(k+1),i})$ and $\mathbb{B}_\infty^{(k+1),i} = \text{midrad}(\boldsymbol{m}^{(k+1),i}, \boldsymbol{r}^{(k+1),i})$, where $i = 1, 2$. Finally, we can get the pair $(\mathbb{B}_2^{(k+1)}, \mathbb{B}_\infty^{(k+1)})$ for $h^{(k+1)}$ as follows:

$$\begin{aligned}
\mathbb{B}_2^{(k+1)} &= \mathbb{B}_2(\mathbf{z}^{(k+1)}, \rho^{(k+1)}), \\
\mathbb{B}_\infty^{(k+1)} &= \text{midrad}(\boldsymbol{m}^{(k+1)}, \boldsymbol{r}^{(k+1)}) \text{ s.t} \\
\mathbf{z}^{(k+1)} &= \mathbf{z}^{(k+1),1} + \mathbf{z}^{(k+1),2}, \\
\rho^{(k+1)} &= \rho^{(k+1),1} + \rho^{(k+1),2}, \\
\boldsymbol{m}^{(k+1)} &= \boldsymbol{m}^{(k+1),1} + \boldsymbol{m}^{(k+1),2}, \\
\boldsymbol{r}^{(k+1)} &= \boldsymbol{r}^{(k+1),1} + \boldsymbol{r}^{(k+1),2}.
\end{aligned} \tag{B.2.2}$$

B.2.5 Complexity Analysis

In this section, we provide the computational complexity analysis on the proposed algorithm. To simplify the discussion, we consider a K'-layered feedforward network which has K' linear layers followed by non-linear activations. We further suppose the network has n neurons for all $K'-1$ layers except for the output layers which has $c \ll n$ neurons. The simple forward propagation costs $O((K'-1)n^2+nc) = O(K'n^2)$ for each input. For the proposed method, it takes $O(2(K'-1)n^2)$ for (3.2.6) and (3.2.7). In addition, it takes $O(2s(K'-1)n^2)$ for the power iteration with the iteration $s(=1)$, and takes $O(2tn)$ for computing η in (3.2.12) with the iteration $t \ll n$. Lastly, to compute the worst-translated logit, it takes $O(cn)$. Therefore, the total computation of the BCP costs $O((4+2s)(K'-1)n^2 + 2tn + cn) = O(K'n^2)$ which is only $O(1)$ times slower than regular training. In detail, BCP is about 6 times slower than regular training for the iteration $s = 1$, and we empirically found that it is roughly correct (e.g. 8 vs 53 sec/epoch for 4C3F on CIFAR-10).

B.3 Experimental Settings

B.3.1 Data Description

MNIST: 10 classes, 600K training images, 100K test images.
CIFAR-10 [36]: 10 classes, 500K training images, 100K test images.
Tiny ImageNet: 200 classes, 100K training images, 10K validation images, 10K test images.

B.3.2 Hyper-parameters

We list the hyper-parameters used in the proposed certifiable training in Table B.1. They are obtained using grid search. We considered the parameters, learning rate $\in [0.0001, 0.0003, 0.001, 0.003, 0.01]$, the length of the warm-up period $\in [1, 3, 5]$, and the length of the ramp-up period $\in [10, 20, 50]$. We also considered the hyperparameters used in CRONW-IBP [78]. In our certifiable training, we used the objective (3.2.13). During warm-up period, we conducted the standard training, i.e. $\lambda = 0$ while during ramp-up period, we gradually increased λ from 0 to 1. The sensitivity analysis on the schedule of λ will be discussed in Section B.3.4. For CAP [72], we used the pretrained models given by the authors.[1] We used a single iteration for the power iteration as mentioned in Section B.2.2, and used 10 iterations for the optimization (3.2.11) during training on MNIST. On CIFAR-10, we found that it is enough to run a single iteration for the optimization (3.2.11) during training. On Tiny ImageNet, we cannot run more than one iteration because of the memory constraint, but we can obtain a verification accuracy of 20.1%. In test phase, we run both iterations until convergence. To evaluate the PGD accuracy, we set the step-size to $\epsilon_{eval}/4$ and the number of iterations to 100.

Table B.1: Hyper-parameters used in the certifiable training.

Data	image size	# class	optimizer	optimizer parameters	epoch	learning rate	warm-up/ ramp-up	weight decay ($\times\gamma$)
MNIST	(28,28)	10	Adam	$\gamma = 0.1$	60	0.0003	1/20	[21,30,40]
CIFAR-10	(3,32,32)	10	Adam	$\gamma = 0.5$	100	0.001	10/121	every 10 epochs after 131
Tiny ImageNet	(3,64,64)	200	SGD	$\gamma = 0.1$ momentum=0.1 weight decay=$2e^{-4}$	100	0.001	2/50	[50,70,90]

B.3.3 Network architectures

We denote the convolutional layer with the output channel c, the kernel k, and the stride s as $\mathrm{C}(c, k, s)$ (or $\mathrm{C}(c, k, s, p)$ if it uses the padding $p \neq 0$) and the linear layer with the output channel c as $\mathrm{F}(c)$. We apply ReLU activation after every convolutional layers and linear layers except for the

[1]`https://github.com/locuslab/convex_adversarial/model_scaled_l2`

last linear layer. For brevity, we omit the notation for the ReLU activation layers and the flatten layer before the first linear layer. The network 4C3F is the same as that used in [72].

- 4C3F:
  C(32,3,1,1)-C(32,4,2,1)-C(64,3,1,1)-C(64,4,2,1)-F(512)-F(512)-F(10)
- 6C2F:
  C(32,3,1,1)-C(32,3,1,1)-C(32,4,2,1)-C(64,3,1,1)-C(64,3,1,1)-C(64,4,2,1)-F(512)-F(10)
- 8C2F (Tiny ImageNet):
  C(64,3,1,1)-C(64,3,1,1)-C(64,4,2)-C(128,3,1,1)-C(128,3,1,1)-C(128,4,2)-C(256,3,1,1)-C(256,4,2)-F(256)-F(200)

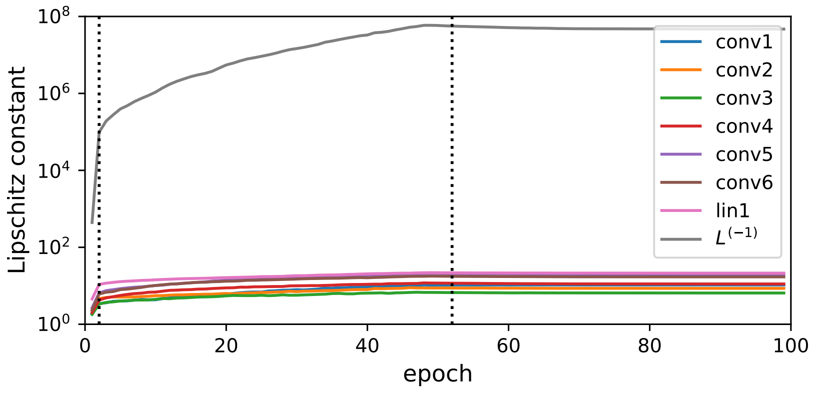
B.3.4 Additional Experiments

Lipschitz constant. The Lipschitz constant of a neural network is tightly correlated with the excess risk, the difference between the test error and the training error. This implies that a network with a large Lipschitz constant shows poor generalization performance [5]. However, the Lipschitz constant keeps increasing through standard training as shown in Figure B.1 (top). On the other hand, the Lipschitz constant keeps decreasing in the BCP training phase as shown in Figure B.1 (bottom). The left dotted vertical line indicates the step when warm-up ends, and the other line indicates the step when the ramp-up ends. It demonstrates that our objective (3.2.13) encourages the model to be robust and to generalize better than standard training.

On the other hand, strong constraints on the global Lipschitz constant may reduce the expressive capacity and the performance of the network [31]. As shown in Figure 3.2 and 3.3, BCP can compute a tighter outer bound, and thus it can relieve the constraints on the Lipschitz constant of the model.

Figure B.2 shows the ratio of the Lipschitz constant of the model trained with BCP ($L_{BCP}^{(-1)} = \Pi_{k=1}^{K-1} L_{BCP}^{(k)}$) to its counterpart trained without BCP ($L_0^{(-1)} = \Pi_{k=1}^{K-1} L_0^{(k)}$). Using BCP had 17.2% larger Lipschitz constant, achieving a higher model capacity as shown in the results of the standard accuracy in Table 3.2.

Therefore, we can conclude that our certifiable training can keep generalization performance without excessive loss of the expressive capacity.



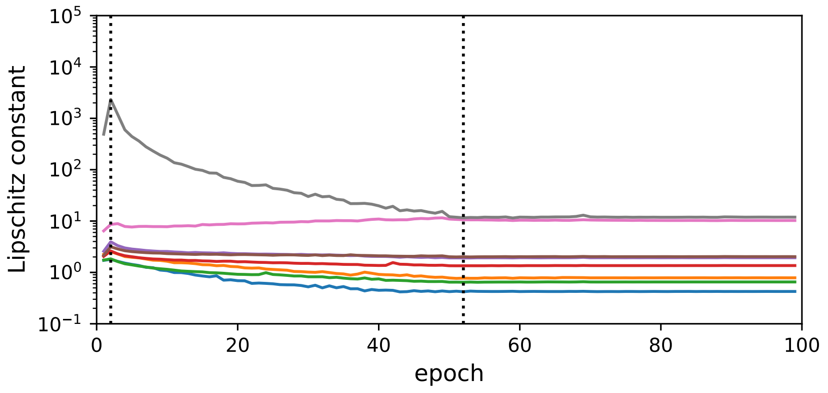


Figure B.1: The change of the Lipschitz constants in a log scale during standard training (top) and the BCP training (bottom).

Verification Performance As in Figure 3.4 in Section 3.2.2, we conducted the same analysis on different settings as demonstrated in Figure B.3. On MNIST, we use $\epsilon'_{target} = 2$ instead of $2\epsilon_{target}$ because $\epsilon_{target} = 1.58$ is already large enough. The verification accuracies of BCP and LMT slowly decrease as increasing ϵ_{eval} almost linearly. However, CAP fails to achieve robustness under the large perturbations for ϵ_{target} and to train the robust model for ϵ'_{target}. For CIFAR-10, we also tested on 6C2F architecture and found that BCP outperforms the other state-of-the-art methods. Unlike in the case of BCP where the verification accuracy decreases slowly as increasing ϵ_{eval}, the

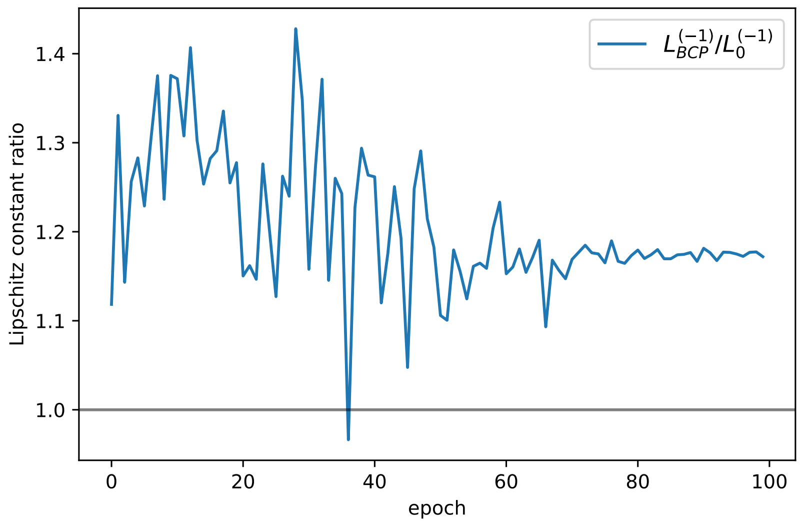


Figure B.2: The change of the ratio of the Lipschitz constant between a model trained with BCP and a model trained without BCP during training.

verification accuracy of CAP with ϵ_{target} drastically decreases when ϵ_{eval} becomes larger than ϵ_{target}.

We also analyze the trade-off between standard accuracy and certifiable robustness. We used different target bounds $\epsilon_{target} = \{30, 32, 34, 36, 38, 40, 42\}/255$ during training to obtain different models while fixing the evaluation bound with $\epsilon_{eval} = 36/255$. Figure B.4 shows the trade-off for 7-layer convolutional networks, 4C3F, trained with BCP, without BCP, and with LMT [69] on CIFAR-10. The true robust classification error is lower bounded by the classification error against PGD [51] and is upper bounded by the verification error. In Figure B.4, the x-axis indicates standard accuracy (a_s) and on the y-axis, we plot line segments representing intervals $[l, u]$ containing the true robust classification accuracy with the lower bound of verification accuracy ($l = a_v$) and with the upper bound of classification accuracy against PGD ($u = a_{PGD}$). We connected the points, (a_s, a_v), with the dotted lines in Figure B.4, demonstrating that BCP has the best trade-off among those methods. The smaller the target bound, the better the standard accuracy achieved. We also found that the robust accuracy against PGD increases with standard accuracy.

Comparison to IBP [27] and CAP [72] as an ℓ_∞-certifiable training As mentioned in Section 3.2.1, BCP can be extended to other norm-bounded perturbations, e.g., ℓ_∞-norm. We compare the performance of BCP to IBP and CAP as an ℓ_∞-certifiable training. We repeated training 4C3F networks 9 times for BCP and IBP models respectively, evaluated them, and reported the results in Table B.2. We used the pre-trained 4C4F network for CAP as in the previous experiments. We set ϵ_{target} and ϵ_{eval} to 8/255. For BCP and IBP, we present median, maximum, and minimum values of 9 results. Table B.2 shows that BCP has comparable performance to IBP and outperforms CAP.

Table B.2: Comparison to other ℓ_∞-verifiable training methods. Best performances are highlighted in bold.

Method	**Accuracy** (%); median (max/min)		
	standard	**PGD**	**Verification**
BCP	**36.50** (38.56/**34.65**)	**27.82** (29.53/**26.85**)	**24.20** (25.45/**22.01**)
IBP	35.81 (**38.62**/31.23)	27.30 (**30.20**/23.73)	23.68 (**26.68**/21.18)
CAP	19.00	17.33	16.06

Sensitivity analysis on the schedule parameter λ_0 and λ_1 We gradually increase λ in (3.2.13) from $\lambda_0 = 0$ to $\lambda_1 = 1$ during training. In this section, we perform a sensitivity analysis on the parameters λ_0 and λ_1. We train a model with the 4C3F architecture on CIFAR-10. Table B.3 shows that the performance has little to do with the initial weight λ_0 but it is highly related with the final weight λ_1. When the final weight λ_1 is close to 1, we get a lower standard accuracy and a higher verification accuracy. We trained three models for each pair (λ_0, λ_1), and report average performance measures in Table B.3.

Table B.3: Sensitivity analysis on the parameters λ_0 and λ_1. We provide three performance measures (standard/PGD/verification accuracy (%)).

	λ_0			
λ_1	**1**	**0.9**	**0.5**	**0.01**
1	66.13/59.92/49.72	66.29/59.90/49.90	66.47/59.65/49.96	66.09/60.00/49.95
0.9		67.18/60.52/48.99	66.90/60.23/49.19	66.91/60.50/49.41
0.5			69.86/61.08/42.32	69.96/61.19/42.70
0.01				71.20/59.57/20.69

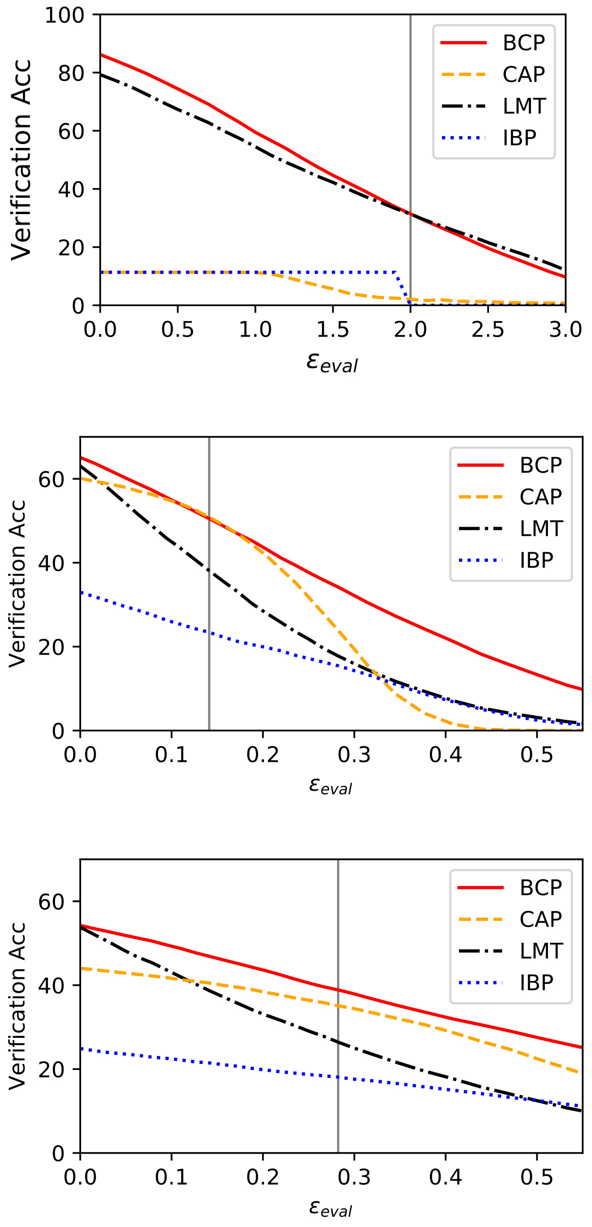


Figure B.3: The verification accuracy when varying the ℓ_2-perturbations ϵ_{eval}. We use $\epsilon'_{target} = 2$ on MNIST dataset (Top), and ϵ_{target} (Middle) and $2\epsilon_{target}$ (Bottom) on CIFAR-10 dataset for training, which are represented with the vertical lines.

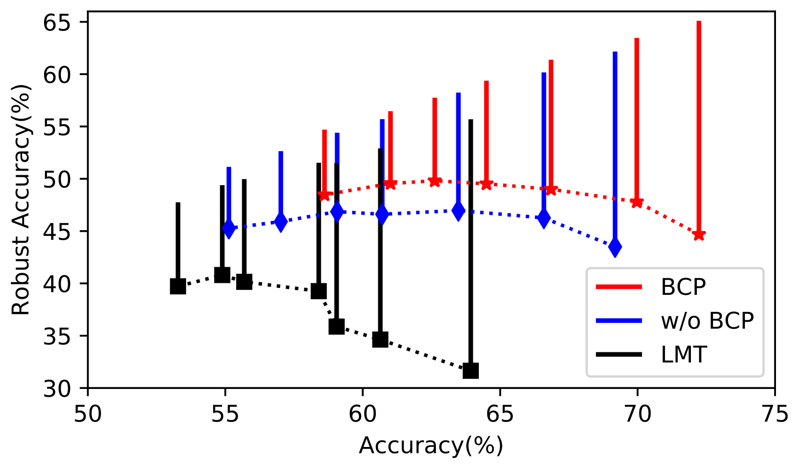


Figure B.4: Trade-off graph between standard accuracy and robustness. The vertical line segments indicate the verification accuracy (lower end points) and the PGD accuracy (upper end points).

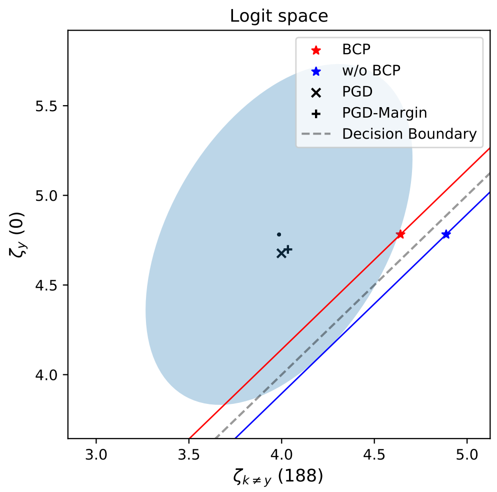


Figure B.5: Illustration of the outer bounds for the BCP trained model on ImageNet.

Appendix C

Appendix for 3.3

C.1 Experimental Settings

Datasets and Architectures In the experiments, we use three datasets: MNIST, CIFAR-10 and SVHN and model architectures (Small, Medium, and Large) in [27] and their variants (Small* and Large*) as follows:

- Small: Conv(·,16,4,2) - Conv(16,32,4,1) - Flatten - FC(·,100) - FC(100,c)
- Small*: Conv(·,16,4,2) - Conv(16,32,4,2) - Flatten - FC(·,100) - FC(100,c)
- Medium: Conv(·,32,3,1) - Conv(32,32,4,2) - Conv(32,64,3,1) - Conv(64,64,4,2) - Flatten - FC(·,512) - FC(512,512) - FC(512,c)
- Large: Conv(·,64,3,1) - Conv(64,64,3,1) - Conv(64,128,3,2) - Conv(128,128,3,1) - Conv(128,128,3,1) - Flatten - FC(·,512) - FC(512,c)
- Large*: Conv(·,64,3,1) - Conv(64,128,3,2) - Conv(128,128,3,1) - Conv(128,128,3,1) - Conv(128,128,3,1) - Flatten - FC(·,512) - FC(512,c)

where Conv(c_1, c_2, k, s) is a conv layer with input channel c_1, output channel c_2, kerner size k, and stride s, and FC(d_1, d_2) is a fully-connected layer with input dimension d_1 and output dimension d_2. All layers are followed by ReLU activation except for the last layer and the flatten layer (Flatten).

Loss and training schedules For general training schedules, we refer to Appendix C, D of [78] with a single GPU (Titan Xp). We use the following mixed cross-entropy loss as in [78]:

$$\kappa\mathcal{L}\left(f(\boldsymbol{x};\boldsymbol{\theta})\right) + (1-\kappa)\mathcal{L}\left((1-\beta)\overline{s}^{\mathrm{IBP}}(\boldsymbol{x};\boldsymbol{\theta}) + \beta\overline{s}^{\mathrm{MODEL}}(\boldsymbol{x};\boldsymbol{\theta})\right), \tag{C.1.1}$$

where κ is the mixing weight between the natural loss and the robust loss, and β is the mixing weight between the two bounds obtained with IBP and given relaxation method (e.g. CROWN-IBP).

C.1.1 Settings in Section 3.3.2

Figure 3.6 We conduct the experiment in Figure 3.6 on CIFAR-10 dataset with Medium architecture over all four methods. We train the model with $\epsilon_{train} = {}^{8}/_{255}$ for 200 epochs using ϵ_t-scheduling with 10 warm-up epochs and 120 ramp-up epochs. We use Adam optimizer with learning rate 0.001. We reduce the learning rate by 50% every 10 epochs after ϵ_t-scheduling ends.

To demonstrate the instability of each training, we describe the variation of the loss along the gradient direction as [62]. We take steps of different lengths in the direction of the gradient and measure the loss values obtained at each step. For the sake of consistency, we fix a Cauchy random matrix when evaluating CAP to obtain deterministic loss landscapes, not introducing randomness. The loss variation is computed with

$$\mathcal{L}^{\epsilon_t}(\boldsymbol{\theta}([0,5])) \text{ where } \boldsymbol{\theta}(\lambda) \equiv \boldsymbol{\theta}_t - \lambda\eta\nabla_{\boldsymbol{\theta}}\mathcal{L}^{\epsilon_t}(\boldsymbol{\theta}_t), \tag{C.1.2}$$

where $\boldsymbol{\theta}_t$ is the current model parameters and η is the learning rate. For the step of length λ, we sample ten points from a range of [0,5] on a log scale.

Figure 3.7 In Figure 3.7, with the same model used in Figure 3.6, we plot the ℓ_2- and cosine distance between two successive loss gradient steps during training as follows:

$$\text{Grad Difference (Middle)} = \|\nabla_{\boldsymbol{\theta}}\mathcal{L}^{\epsilon_t}(\boldsymbol{\theta}_t), \nabla_{\boldsymbol{\theta}}\mathcal{L}^{\epsilon_t}(\boldsymbol{\theta}_{t+1})\| \text{ and}$$
$$\text{Cosine Distance (Bottom)} = 1 - \cos(\nabla_{\boldsymbol{\theta}}\mathcal{L}^{\epsilon_t}(\boldsymbol{\theta}_t), \nabla_{\boldsymbol{\theta}}\mathcal{L}^{\epsilon_t}(\boldsymbol{\theta}_{t+1})),$$

where $\cos(\boldsymbol{v}_1, \boldsymbol{v}_2)$ is the cosine value of the angle between two vectors $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$.

C.1.2 Settings in Table 3.4

For MNIST, we use the same hyper-parameters as in Appendix C of [78]. We train for 200 epochs (10 warm-up epochs and 50 ramp-up epochs) on Large model with batch sizes of 100. we decay the learning rate, 0.0005, by 10% in [130,190] epochs. As mentioned in [78], we also found the same issue when training with small ϵ (see Appendix C.9 for details). To alleviate the issue, we use $\epsilon_{train} = \min(0.4, \epsilon_{test} + 0.1)$ for each ϵ_{test} as Table 2 of [78].

For CIFAR-10, we train for 400 epochs (20 warm-up epochs and 240 ramp-up epochs) on Medium model with batch sizes of 128. We decay the learning rate, 0.003, by $2\times$ every 10 epochs after the ramp-up period.

For SVHN, we train for 200 epochs (10 warm-up epochs and 120 ramp-up epochs) on Large model with batch sizes of 128 (OURS with batch sizes of 80 to avoid out of memory). We decay the learning rate, 0.0003, by $2\times$ every 10 epochs after the ramp-up period. Only for SVHN, we apply normalization with mean (0.438, 0.444, 0.473) and standard deviation (0.198, 0.201, 0.197) for each channel.

In Table 3.4, we use κ-scheduling from 1 to 0. For the corresponding results of κ-scheduling from 0 to 0, we refer the reader to Table C.2.

We modify the source code for CAP[1] to match our settings. For example, we introduce the warm-up period and linear ϵ-scheduling. We avoid using the reported results in the literature and aim to make a fair comparison under the same settings with only minor differences - for example, because CAP does not support the channel-wise normalization, we could not use the input normalization. Also, due to the memory limit of CAP, we use a smaller batch size of 32 and try other smaller architectures. We found that it often achieves better results with smaller architectures (similar to the results in Table 3 of [72]). Thus, we present the performance with Large*, Medium, and Small* on MNIST, CIFAR-10, and SVHN, respectively. Throughout the experiments, CAP uses the fixed $\kappa = 0$.

[1]https://github.com/locuslab/convex_adversarial

C.2 Interval Bound Propagation (IBP)

IBP [27] starts from the interval bound $\mathcal{I}^{(0)} \equiv \{\boldsymbol{z} : \boldsymbol{l}^{(0)} \leq \boldsymbol{z} \leq \boldsymbol{u}^{(0)}\} = \mathbb{B}(\boldsymbol{x}, \epsilon)$ in the input space with the upper bound $\boldsymbol{u}^{(0)} = \boldsymbol{x} + \epsilon\mathbf{1}$ and the lower bound $\boldsymbol{l}^{(0)} = \boldsymbol{x} - \epsilon\mathbf{1}$ where $\mathbf{1}$ is a column vector filled with 1. Then we propagate the interval bound $\mathcal{I}^{(k-1)} \equiv \{\boldsymbol{z} : \boldsymbol{l}^{(k-1)} \leq \boldsymbol{z} \leq \boldsymbol{u}^{(k-1)}\}$ by using following equations iteratively:

$$\boldsymbol{u}^{(k)} = h^{(k)}(\boldsymbol{u}^{(k-1)}) \text{ and } \boldsymbol{l}^{(k)} = h^{(k)}(\boldsymbol{l}^{(k-1)}) \tag{C.2.3}$$

for element-wise monotonic increasing nonlinear activation $h^{(k)}$ with the pre-activation bounds $\boldsymbol{u}^{(k-1)}$ and $\boldsymbol{l}^{(k-1)}$, and

$$\begin{aligned}\boldsymbol{u}^{(k)} &= \boldsymbol{W}^{(k)}\left(\frac{\boldsymbol{u}^{(k-1)} + \boldsymbol{l}^{(k-1)}}{2}\right) + |\boldsymbol{W}^{(k)}|\left(\frac{\boldsymbol{u}^{(k-1)} - \boldsymbol{l}^{(k-1)}}{2}\right) + \boldsymbol{b}^{(k)} \text{ and} \\ \boldsymbol{l}^{(k)} &= \boldsymbol{W}^{(k)}\left(\frac{\boldsymbol{u}^{(k-1)} + \boldsymbol{l}^{(k-1)}}{2}\right) - |\boldsymbol{W}^{(k)}|\left(\frac{\boldsymbol{u}^{(k-1)} - \boldsymbol{l}^{(k-1)}}{2}\right) + \boldsymbol{b}^{(k)}\end{aligned}$$

for linear function $h^{(k)}$ $(k = 1, \cdots, K)$. Finally, IBP uses the worst-case margin $\overline{\boldsymbol{s}} = \boldsymbol{u}^{(K)}$ to formulate the objective in (3.3.14) for certifiable training.

C.3 Details on Linear Relaxation

C.3.1 Linear relaxation explained in CROWN [79]

To make the paper self-contained, we provide details of linear relaxation given in the supplementary material of CROWN [79]. We refer readers to the supplementary for more details. Given a network $h^{[k]}$, we want to upper bound the activation $h_i^{[k]}$. We have $h_i^{[k]}(\boldsymbol{x}') = \boldsymbol{W}_{i,:}^{(k)} h^{(k-1)}(h^{[k-2]}(\boldsymbol{x}')) + \boldsymbol{b}_i^{(k)} = \boldsymbol{W}_{i,:}^{(k)} h^{(k-1)}(\boldsymbol{z}^{(k-2)\prime}) + \boldsymbol{b}_i^{(k)}$ where $\boldsymbol{z}^{(k-2)\prime} = h^{[k-2]}(\boldsymbol{x}')$. With the linear function bounds of $\overline{h}^{(k-1)}$ and $\underline{h}^{(k-1)}$ on the activation function $h^{(k-1)}$, we have

$$\begin{aligned}h_i^{[k]}(\boldsymbol{x}') =& \boldsymbol{W}_{i,:}^{(k)} h^{(k-1)}(\boldsymbol{z}^{(k-2)\prime}) + \boldsymbol{b}_i^{(k)} \\ &\leq \sum_{\boldsymbol{W}_{i,j}^{(k)} < 0} \boldsymbol{W}_{i,j}^{(k)} \underline{h}_j^{(k-1)}(\boldsymbol{z}^{(k-2)\prime}) + \sum_{\boldsymbol{W}_{i,j}^{(k)} \geq 0} \boldsymbol{W}_{i,j}^{(k)} \overline{h}_j^{(k-1)}(\boldsymbol{z}^{(k-2)\prime}) + \boldsymbol{b}_i^{(k)}\end{aligned}$$

$$
\begin{aligned}
&= \sum_{\boldsymbol{W}_{i,j}^{(k)}<0} \boldsymbol{W}_{i,j}^{(k)} \underline{a}_j^{(k-1)} \boldsymbol{z}_j^{(k-2)'} + \sum_{\boldsymbol{W}_{i,j}^{(k)}\geq 0} \boldsymbol{W}_{i,j}^{(k)} \overline{a}_j^{(k-1)} \boldsymbol{z}_j^{(k-2)'} \\
&\quad + \sum_{\boldsymbol{W}_{i,j}^{(k)}<0} \boldsymbol{W}_{i,j}^{(k)} \underline{b}_j^{(k-1)} + \sum_{\boldsymbol{W}_{i,j}^{(k)}\geq 0} \boldsymbol{W}_{i,j}^{(k)} \overline{b}_j^{(k-1)} + \boldsymbol{b}_i^{(k)} \\
&= \tilde{\boldsymbol{W}}_{i,:}^{(k)} \boldsymbol{z}^{(k-2)'} + \tilde{\boldsymbol{b}}_i^{(k)} \\
&= \tilde{\boldsymbol{W}}_{i,:}^{(k)} h^{[k-2]}(\boldsymbol{x}') + \tilde{\boldsymbol{b}}_i^{(k)} \\
&= \tilde{\boldsymbol{W}}_{i,:}^{(k)} \left(\boldsymbol{W}^{(k-2)} (h^{[k-3]}(\boldsymbol{x}')) + \boldsymbol{b}^{(k-2)} \right) + \tilde{\boldsymbol{b}}_i^{(k)} \\
&= \hat{\boldsymbol{W}}_{i,:}^{(k-2)} h^{(k-3)}(\boldsymbol{z}^{(k-3)'}) + \hat{\boldsymbol{b}}_i^{(k-2)},
\end{aligned}
$$

where $\tilde{\boldsymbol{W}}_{i,:}^{(k)} = \boldsymbol{W}_{i,:}^{(k)} \boldsymbol{D}^{(k-1)}$ with the diagonal matrix $\boldsymbol{D}_{j,j}^{(k-1)} = \underline{a}_j^{(k-1)}$ for j satisfying $\boldsymbol{W}_{i,j}^{(k)} < 0$ and $\boldsymbol{D}_{j,j}^{(k-1)} = \overline{a}_j^{(k-1)}$ for j satisfying $\boldsymbol{W}_{i,j}^{(k)} \geq 0$, $\tilde{\boldsymbol{b}}_i^{(k)} = \sum_{\boldsymbol{W}_{i,j}^{(k)}<0} \boldsymbol{W}_{i,j}^{(k)} \underline{b}_j^{(k-1)} + \sum_{\boldsymbol{W}_{i,j}^{(k)}\geq 0} \boldsymbol{W}_{i,j}^{(k)} \overline{b}_j^{(k-1)} + \boldsymbol{b}_i^{(k)}$, $\hat{\boldsymbol{W}}_{i,:}^{(k-2)} = \tilde{\boldsymbol{W}}_{i,:}^{(k)} \boldsymbol{W}^{(k-2)}$, and $\hat{\boldsymbol{b}}_i^{(k-2)} = \tilde{\boldsymbol{W}}_{i,:}^{(k)} \boldsymbol{b}^{(k-2)} + \tilde{\boldsymbol{b}}_i^{(k)}$. Applying similar method iteratively, we can obtain $\boldsymbol{g}$ and b in (3.3.15) for the linear relaxation of $h_i^{[k]}$.

C.3.2 Dual Optimization View

We first modify some notations in the main paper and use the notations similar to [71]. We use the following hat notations: $\hat{\boldsymbol{z}}^{(k+1)} = \boldsymbol{W}^{(k+1)} \boldsymbol{z}^{(k)} + \boldsymbol{b}^{(k+1)}$ and $\boldsymbol{z}^{(k)} = h^{(k)}(\hat{\boldsymbol{z}}^{(k)})$ where $h^{(k)}$ is the k-th nonlinear activation function. We can build a primal problem with $\boldsymbol{c}^T = \boldsymbol{C}_{m,:}$ as follows:

$$
\max_{\boldsymbol{z}^{(K)}} \boldsymbol{c}^T \hat{\boldsymbol{z}}^{(K)} \tag{C.3.4}
$$

such that

$$
\begin{aligned}
\boldsymbol{x} - \epsilon \mathbf{1} &\leq \boldsymbol{z}^{(0)}, \\
\boldsymbol{z}^{(0)} &\leq \boldsymbol{x} + \epsilon \mathbf{1}, \\
\hat{\boldsymbol{z}}^{(k+1)} &= \boldsymbol{W}^{(k+1)} \boldsymbol{z}^{(k)} + \boldsymbol{b}^{(k+1)} \ (k = 0, \cdots, K-1), \text{ and} \\
\boldsymbol{z}^{(k)} &= h^{(k)}(\hat{\boldsymbol{z}}^{(k)}) \ (k = 1, \cdots, K-1).
\end{aligned}
$$

Note that our $\boldsymbol{c}$ is negation of that of [71]. Now we can derive the dual of the primal (C.3.4) as follows:

$$
\begin{aligned}
\min_{\substack{\boldsymbol{\xi}^+,\boldsymbol{\xi}^-\geq \mathbf{0}\\ \boldsymbol{\nu}_k}} \sup_{\boldsymbol{z}^{(k)},\hat{\boldsymbol{z}}^{(k)}} & \boldsymbol{c}^T\hat{\boldsymbol{z}}^{(K)} + \boldsymbol{\xi}^{-T}(\boldsymbol{x}-\epsilon\mathbf{1}-\boldsymbol{z}^{(0)}) + \boldsymbol{\xi}^{+T}(\boldsymbol{z}^{(0)}-\boldsymbol{x}-\epsilon\mathbf{1}) \\
& + \sum_{k=0}^{K-1}\boldsymbol{\nu}_{k+1}^T\left(\hat{\boldsymbol{z}}^{(k+1)} - (\boldsymbol{W}^{(k+1)}\boldsymbol{z}^{(k)} + \boldsymbol{b}^{(k+1)})\right) + \sum_{k=1}^{K-1}\hat{\boldsymbol{\nu}}_k^T\left(\boldsymbol{z}^{(k)} - h^{(k)}(\hat{\boldsymbol{z}}^{(k)})\right) \\
& = (\boldsymbol{c}+\boldsymbol{\nu}_K)^T\hat{\boldsymbol{z}}^{(K)} + (\boldsymbol{\xi}^+ - \boldsymbol{\xi}^- - \boldsymbol{W}^{(1)T}\boldsymbol{\nu}_1)^T\boldsymbol{z}^{(0)} + \sum_{k=1}^{K-1}(-\boldsymbol{W}^{(k+1)T}\boldsymbol{\nu}_{k+1} + \hat{\boldsymbol{\nu}}_k)^T\boldsymbol{z}^{(k)} \\
& + \sum_{k=1}^{K-1}(\hat{\boldsymbol{\nu}}_k^T h^{(k)}(\hat{\boldsymbol{z}}^{(k)}) - \boldsymbol{\nu}_k^T\hat{\boldsymbol{z}}^{(k)}) \qquad \text{(C.3.5)} \\
& - \boldsymbol{\nu}_1^T\boldsymbol{b}^{(1)} - \boldsymbol{\xi}^T\boldsymbol{x} - \epsilon||\boldsymbol{\xi}||_1.
\end{aligned}
$$

It leads to $\boldsymbol{c}+\boldsymbol{\nu}_K = \mathbf{0}, \boldsymbol{\xi}^+ - \boldsymbol{\xi}^- - \boldsymbol{W}^{(1)T}\boldsymbol{\nu}_1 = \mathbf{0}$, and $-\boldsymbol{W}^{(k+1)T}\boldsymbol{\nu}_{k+1} + \hat{\boldsymbol{\nu}}_k = \mathbf{0}$ ($k = 1, \cdots, K-1$). Alternatively, they are represented as follows:

$$
\begin{aligned}
\boldsymbol{\nu}_K &= -\boldsymbol{c}, \\
\hat{\boldsymbol{\nu}}_k &= \boldsymbol{W}^{(k+1)T}\boldsymbol{\nu}_{k+1}\ (k = K-1, \cdots, 1), \text{ and} \\
\boldsymbol{\xi} &= \hat{\boldsymbol{\nu}}_1.
\end{aligned}
$$

Now we need relationship between $\hat{\boldsymbol{\nu}}_k$ and $\boldsymbol{\nu}_k$, i.e., $\boldsymbol{\nu}_k = g(\hat{\boldsymbol{\nu}}_k)$. With the further relaxation $\boldsymbol{\nu}_k = \boldsymbol{\alpha}_k \odot \hat{\boldsymbol{\nu}}_k$, we have a relaxed problem as follows:

$$
\min_{\boldsymbol{\alpha}_k} \sup_{\boldsymbol{z}^{(k)},\hat{\boldsymbol{z}}^{(k)}} \sum_{k=1}^{K-1}(\hat{\boldsymbol{\nu}}_k^T h^{(k)}(\hat{\boldsymbol{z}}^{(k)}) - \boldsymbol{\nu}_k^T\hat{\boldsymbol{z}}^{(k)}) - \boldsymbol{\nu}_1^T\boldsymbol{b}^{(1)} - \boldsymbol{\xi}^T\boldsymbol{x} - \epsilon||\boldsymbol{\xi}||_1 \qquad \text{(C.3.6)}
$$

such that

$$
\begin{aligned}
\boldsymbol{\nu}_K &= -\boldsymbol{c}, \\
\hat{\boldsymbol{\nu}}_k &= \boldsymbol{W}^{(k+1)T}\boldsymbol{\nu}_{k+1}\ (k = K-1, \cdots, 1), \\
\boldsymbol{\nu}_k &= \boldsymbol{\alpha}_k \odot \hat{\boldsymbol{\nu}}_k\ (k = K-1, \cdots, 1), \text{ and} \\
\boldsymbol{\xi} &= \hat{\boldsymbol{\nu}}_1.
\end{aligned}
$$

We decompose the first term in (C.3.6), and ignore the subscript k as follows $\hat{\boldsymbol{\nu}}^T h(\hat{\boldsymbol{z}}) - (\boldsymbol{\alpha}\odot\hat{\boldsymbol{\nu}})^T\hat{\boldsymbol{z}}$. Further, we decompose this for each element, $\hat{\nu}h(\hat{z}) -$

$\alpha\hat{\nu}\hat{z} = \hat{\nu}(h(\hat{z}) - \alpha\hat{z})$. If the pre-activation bounds for h are both positive (active ReLU), then α should be 1 not to make the inner supremum > 0. Similarly, if the pre-activation bounds for h are both negative (dead ReLU), then α should be 0. In the case of unstable ReLU ($l \leq 0 \leq u$), if $\hat{\nu} < 0$, then we need to solve $\max_\alpha \inf_{\hat{z}} h(\hat{z}) - \alpha\hat{z}$. The inner infimum is 0 for $0 \leq \alpha \leq 1$, and is < 0 otherwise. On the other hand, if $\hat{\nu} \geq 0$, then we need to solve $\min_\alpha \sup_{\hat{z}} h(\hat{z}) - \alpha\hat{z}$. The inner supremum is $\max\{u - \alpha u, -\alpha l\}$, and thus the optimal dual variable is $\alpha^* = \frac{u}{u-l}$ which yields the optimal value (multiplied by $\hat{\nu}$) as $\hat{\nu}(u - \frac{u}{u-l}u) = -\frac{ul}{u-l}\hat{\nu}$ which is equivalent to using linear relaxation with $\overline{\boldsymbol{a}} \odot \boldsymbol{z} + \overline{\boldsymbol{b}} = \frac{\boldsymbol{u}}{\boldsymbol{u}-\boldsymbol{l}} \odot (\boldsymbol{z} - \boldsymbol{l})$. We can represent it as $\overline{\boldsymbol{a}} \odot \boldsymbol{z} + \overline{\boldsymbol{b}} = \frac{\boldsymbol{u}^+}{\boldsymbol{u}^+ - \boldsymbol{l}^-} \odot (\boldsymbol{z} - \boldsymbol{l}^-)$ to include the case of active/dead ReLU. For the lower linear bound $\underline{h}(\boldsymbol{z}) = \underline{\boldsymbol{a}} \odot \boldsymbol{z} + \underline{\boldsymbol{b}}$ in case of unstable ReLU, we can use any $\boldsymbol{0} \leq \underline{\boldsymbol{a}} \leq \boldsymbol{1}$ and $\underline{\boldsymbol{b}} = \boldsymbol{0}$ according to the dual relaxation with $\boldsymbol{\alpha}$. While CAP and CROWN-IBP use a dual feasible solution like $\boldsymbol{\alpha} = \frac{\boldsymbol{u}^+}{\boldsymbol{u}^+ - \boldsymbol{l}^-}$ or $\boldsymbol{\alpha} = \boldsymbol{1}[\boldsymbol{u}^+ + \boldsymbol{l}^- > 0]$, our proposed method aims to optimize over the dual variable $\boldsymbol{\alpha}$ or equivalently optimize over $\boldsymbol{0} \leq \underline{\boldsymbol{a}} \leq \boldsymbol{1}$ to further tighten the upper bound on the loss.

C.4 Learning curves for variants of CROWN-IBP

We find that 0.5/1 and 1/1 have less smooth loss landscapes than CROWN-IBP (as shown in Figure C.1) where p/q denotes the variant with sampling $\underline{a} \in \{0, 1\}$ with $P(\underline{a} = 1 \mid |l| > |u|) = p$ and $P(\underline{a} = 1 \mid |l| \leq |u|) = q$ for unstable ReLUs. On the other hand, 0/0, 0/0.25, and 0/0.5 have more smooth loss landscape as in Figure C.1, but they have looser bounds than CROWN-IBP.

C.5 Mode Connectivity

In this section, we check the mode connectivity [25] between two models that are trained using certifiable training methods. Mode connectivity is a framework that investigates the connectedness between two models by finding a high accuracy curve between those models. It enables us to understand the

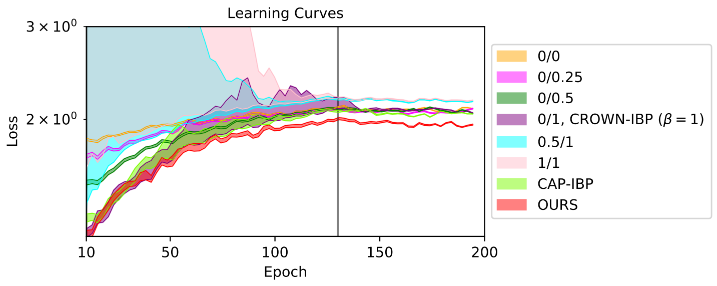


Figure C.1: The learning curves for the scheduled value of ϵ with the loss variation along gradient descent direction (equivalent to Figure 3.6). As $\underline{\boldsymbol{a}}$ becomes sparse, the loss variation is narrower.

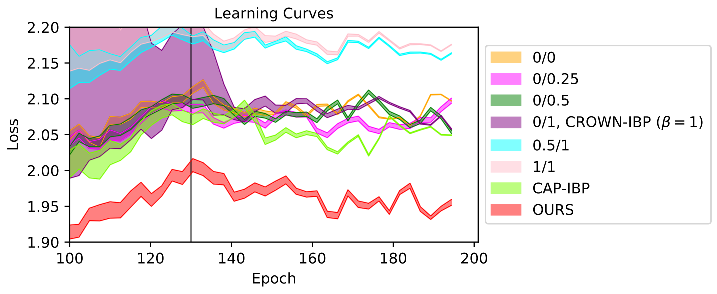


Figure C.2: A zoomed-in version of Figure C.1 for epochs 100-200.

loss surface of neural networks.

Let $\boldsymbol{w}_0$ and $\boldsymbol{w}_1$ be two sets of weight corresponding to two different well-trained neural networks. Moreover, let $\phi_{\boldsymbol{\theta}_c}(t)$ with $t \in [0,1]$ be a continuous piece-wise smooth parametric curve with parameters $\boldsymbol{\theta}_c$ such that $\phi_{\boldsymbol{\theta}_c}(0) = \boldsymbol{w}_0$ and $\phi_{\boldsymbol{\theta}_c}(1) = \boldsymbol{w}_1$. To find a low-loss path between $\boldsymbol{w}_0$ and $\boldsymbol{w}_1$, [25] suggested to find the parameter $\boldsymbol{\theta}_c$ that minimizes the expectation of a loss $\ell(\boldsymbol{w})$ over a distribution $q_{\boldsymbol{\theta}_c}(t)$ on the curve,

$$L(\boldsymbol{\theta}_c) = \mathbb{E}_{t \sim q_{\boldsymbol{\theta}_c}(t)}[\ell(\phi_{\boldsymbol{\theta}_c}(t)].$$

To optimize $L(\boldsymbol{\theta}_c)$ for $\boldsymbol{\theta}_c$, we use uniform distribution $U[0,1]$ as $q_{\boldsymbol{\theta}_c}(t)$ and Bezier curve [22] as $\phi_{\boldsymbol{\theta}_c}(t)$, which provides a convenient parameterization of smoothness on the paths connecting two end points ($\boldsymbol{w}_0$ and $\boldsymbol{w}_1$) as follows:

$$\phi_{\boldsymbol{\theta}_c}(t) = (1-t)^2\boldsymbol{w}_0 + 2t(1-t)\boldsymbol{\theta}_c + t^2\boldsymbol{w}_1, 0 \leq t \leq 1.$$

Table C.1: Performance (in terms of errors) of the variants of CROWN-IBP ($\beta = 1$). Note that 0/0.25, 0/0.5, and CAP-IBP start with looser bounds but they have more smooth landscape, which leads to a better performance than CROWN-IBP ($\beta = 1$) (highlighted with underline).

Model	0/0	0/0.25	0/0.5	0/1 CROWN-IBP ($\beta = 1$)	0.5/1	1/1	CAP-IBP	OURS
Standard	70.66	64.50	62.72	63.24	70.69	71.41	60.36	57.14
PGD	73.84	72.67	71.42	71.70	76.68	77.03	69.46	66.88
Verified	77.60	<u>74.47</u>	<u>74.92</u>	75.72	78.38	78.73	<u>74.29</u>	**71.45**

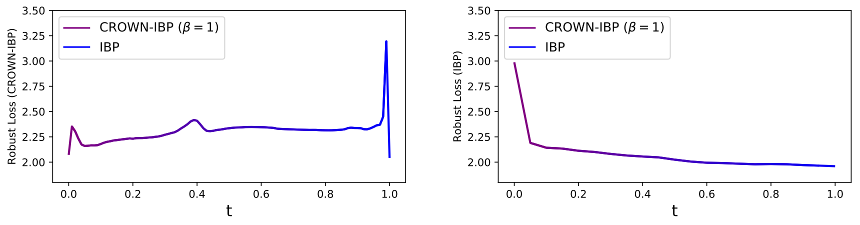
A path $\phi_{\boldsymbol{\theta}_c}$ is said to have a barrier if $\exists t$ such that $\ell(\phi_{\boldsymbol{\theta}_c}(t)) > \max\{\ell(\boldsymbol{w}_0), \ell(\boldsymbol{w}_1)\}$. The existence of a barrier suggests the modes of two well-trained models are not connected by the path in terms of the given loss function ℓ [81].

We test the mode connectivity between the models trained with IBP, CROWN-IBP, and OURS. For example, to check the mode connectivity between two different models trained with CROWN-IBP and IBP, we use the loss function used on each model as a user-specified loss for training the parametric curve $\phi_{\boldsymbol{\theta}_c}$. Therefore, we can obtain two curves as depicted in Figure C.3, C.4, and C.5 for each pair of models. Here, we use the identical settings in Appendix C.1.1.

Figure C.3 shows the mode-connectivity between CROWN-IBP and IBP. We use CROWN-IBP loss as user-specific loss in Figure C.3a and IBP loss in Figure C.3b. In this figure, we find that using CROWN-IBP loss (C.3a), there exists a barrier between the two models. This suggests they are not connected by the path in terms of CROWN-IBP loss. However, with IBP loss, there is no loss barrier separating the two models. This indicates that using CROWN-IBP, it is hard to optimize the parameters from $\boldsymbol{w}_0$ to $\boldsymbol{w}_1$, but IBP can.

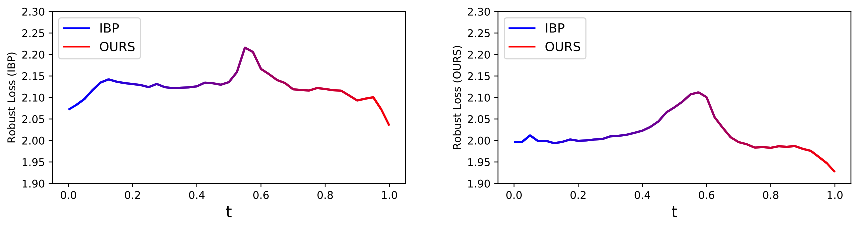
Figure C.4 shows the mode-connectivity results on IBP and OURS. We find that two models are not connected to each other using either IBP bound or OURS bound, since there exists a barrier in both curves. In this figure, we can also notify that OURS has tighter bounds than IBP because the value of the loss function using OURS is lower than that of IBP.

Finally, Figure C.5 illustrates the mode connectivity between CROWN-IBP and OURS. Using CROWN-IBP as a user-specified loss function, we can find that the robust loss on the curve is higher than that of the end points. However, when OURS is used as a loss function, the robust loss generally decreases as the t increases. It shows that OURS has much favorable loss landscape compared to CROWN-IBP. In addition, we can find that OURS has a tighter bound than CROWN-IBP, since the value of the robust loss using OURS is lower than CROWN-IBP.



(a) CROWN-IBP bound (b) IBP bound

Figure C.3: Mode connectivity between CROWN-IBP and IBP, where $\boldsymbol{w}_0$ and $\boldsymbol{w}_1$ are well-trained models using CROWN-IBP bound and IBP bound, respectively. $\boldsymbol{\theta}_c$ is trained using CROWN-IBP (C.3a) and IBP (C.3b), respectively.



(a) IBP bound (b) OURS bound

Figure C.4: Mode connectivity between IBP and OURS, where $\boldsymbol{w}_0$ and $\boldsymbol{w}_1$ are well-trained models using IBP bound and OURS bound, respectively. $\boldsymbol{\theta}_c$ is trained using IBP (C.4a) and OURS (C.4b), respectively.

C.6 ReLU

In this section, we investigate how pre-activation bounds u and l for the activation layer change during training. For each activation node, it is said to be "active" when the pre-activation bounds are both positive ($0 < l \leq u$), "unstable" when they span zero ($l \leq 0 \leq u$), and "dead" when they are both negative ($l \leq u < 0$).

Figure C.6 shows the ratios of the number of active and dead ReLUs during the ramp-up period. Notably, we find that CROWN-IBP has more active ReLUs during training compared to the other three methods. Simultaneously, CROWN-IBP has the lowest ratio of dead ReLUs.

Figure C.7 shows the numbers of active, unstable, and dead ReLUs during the ramp-up period. We find that in CROWN-IBP, the number of unstable and active ReLUs increases as the number of dead ReLUs decreases. This indicates that a number of dead ReLUs change to unstable ReLUs as the training ϵ increases. However, in the other methods, the number of unstable ReLUs is consistently small, while the number of active ReLUs decreases as the number of dead ReLUs increases.

Figure C.8 depicts the histograms of the distribution of the slope $\frac{u^+}{u^+-l^-}$ of the unstable ReLUs during the ramp-up period. In the early stages of CAP training, the slope distribution is concentrated around 0.4. However as the training progresses with a larger ϵ, the histogram distribution moves to left, which indicates unstable ReLUs change to dead ReLUs. It is consistent with the results in Figure C.7c. On the other hand, in the case of CROWN-IBP, the histogram distribution moves to right during training. It is the same with the results in Figure C.7b, which shows that number of active ReLUs increases during training.

C.7 β- and κ-schedulings

Table C.2 shows the evaluation results of the models as in Table 3.4 but trained with different κ-scheduling (from 0 to 0). Table C.3 shows the evaluation results of the proposed models trained with different κ- and β-schedulings.

Table C.2: Test errors (Standard / PGD / Verified error) of IBP, CROWN-IBP ($\beta = 1$), CAP, and OURS on MNIST, CIFAR-10, and SVHN. See Appendix C.1 for all the other settings, same as in Table 3.4. Bold and underline numbers are the first and second lowest verified error.

Data	$\epsilon(l_\infty)$	IBP	CROWN-IBP ($\beta = 1$)	CAP	OURS
MNIST	$\epsilon = 0.1$	1.25 / 2.31 / 3.10	1.23 / 2.19 / $\underline{2.75}$	0.80 / 1.73 / 3.19	1.09 / 1.86 / **2.28**
	$\epsilon = 0.2$	1.95 / 2.95 / $\underline{6.28}$	2.89 / 5.32 / 7.61	3.22 / 6.72 / 11.06	1.70 / 3.37 / **4.78**
	$\epsilon = 0.3$	3.67 / 5.55 / $\underline{9.74}$	6.11 / 11.33 / 17.51	19.19 / 35.84 / 47.85	3.39 / 4.85 / **9.12**
	$\epsilon = 0.4$	3.67 / 6.55 / $\underline{16.55}$	6.11 / 15.34 / 26.72	-	3.39 / 5.88 / **15.04**
CIFAR 10	$\epsilon = 2/255$	43.60 / 52.62 / 56.58	32.15 / 42.67 / 49.36	28.80 / 38.95 / **48.50**	32.04 / 43.13 / $\underline{49.62}$
	$\epsilon = 4/255$	53.89 / 62.58 / 65.14	45.05 / 56.46 / 63.04	40.78 / 52.62 / $\underline{61.88}$	43.15 / 54.85 / **61.31**
	$\epsilon = 6/255$	61.37 / 68.64 / 70.82	53.87 / 65.03 / 71.08	49.20 / 60.85 / $\underline{69.03}$	50.99 / 62.23 / **67.59**
	$\epsilon = 8/255$	64.11 / 70.68 / $\underline{72.99}$	60.96 / 70.52 / 75.68	56.77 / 66.78 / 73.02	56.35 / 67.06 / **70.56**
	$\epsilon = 16/255$	69.74 / 76.66 / $\underline{79.86}$	79.14 / 83.64 / 84.36	75.11 / 80.67 / 82.07	66.96 / 75.63 / **78.08**
SVHN	$\epsilon = 0.01$	20.19 / 34.57 / 44.25	16.66 / 30.05 / $\underline{38.15}$	16.88/ 30.16 / **37.09**	15.46 / 29.34 / 38.57

C.8 one-step vs multi-step

To get a tighter bound, we propose multi-step version of (3.3.24) as follows:

$$\underline{\boldsymbol{a}}_{t+1} = \Pi_{[0,1]^n}\left(\underline{\boldsymbol{a}}_t - \alpha \mathrm{sign}(\nabla_{\underline{\boldsymbol{a}}} \mathcal{L}(\overline{\boldsymbol{s}}(\boldsymbol{x}, y, \epsilon; \boldsymbol{\theta}, \boldsymbol{\phi}), y))\right). \tag{C.8.7}$$

We compare the original 1-step method ($\alpha \geq 1$) to 7-step ($t = 7$) method with $\alpha = 0.1$. The results are summarized in Table C.4. We found no significant difference between two methods even though multi-step takes multiple times with multi-step. Therefore, we decide to focus on one-step method.

C.9 Train with $\epsilon_{train} \geq \epsilon_{test}$

C.9.1 $\epsilon_{train} \geq \epsilon_{test}$ on MNIST

[78] and [27] observed that IBP performs better when using $\epsilon_{train} \geq \epsilon_{test}$ than $\epsilon_{train} = \epsilon_{test}$. Figure C.5 shows the results with different ϵ_{train}'s for each ϵ_{test}. The overfitting issue is more prominent in the case of IBP and

Table C.3: Test errors of OURS with different β- and κ-scheduling on MNIST and CIFAR-10.

Data	$\epsilon(l_\infty)$	**OURS**$_{1\to1}$ ($\kappa = 1 \to 0$)			**OURS**$_{1\to0}$ ($\kappa = 1 \to 0$)			**OURS**$_{1\to1}$ ($\kappa = 0 \to 0$)			**OURS**$_{1\to0}$ ($\kappa = 0 \to 0$)		
		Standard	PGD	Verfied	Standard	PGD	Verfied	Standard	PGD	Verfied	Standard	PGD	Verfied
MNIST	$\epsilon = 0.1$	1.09	1.77	2.36	1.29	2.29	3.58	1.09	1.86	2.28	1.15	2.03	3.53
	$\epsilon = 0.2$	1.70	3.44	4.34	1.61	3.09	5.71	1.70	3.37	4.78	1.64	2.57	5.43
	$\epsilon = 0.3$	3.49	5.59	9.79	2.42	4.37	7.84	3.39	4.85	9.12	2.44	4.41	8.00
	$\epsilon = 0.4$	3.49	6.77	15.42	2.42	5.68	13.72	3.39	5.88	15.04	2.44	5.29	13.84
CIFAR 10	$\epsilon = 2/255$	31.49	42.73	49.42	37.77	48.30	54.43	32.04	43.13	49.62	38.58	48.59	54.63
	$\epsilon = 8/255$	56.01	66.17	69.70	58.87	67.76	71.50	56.35	67.06	70.56	58.90	67.81	70.99
	$\epsilon = 16/255$	65.39	75.39	77.87	66.24	74.69	78.66	66.96	75.63	78.08	66.76	75.17	77.99

CROWN-IBP$_{1\to0}$ than the proposed method and CROWN-IBP$_{1\to1}$. However, using larger perturbations compromises the standard accuracy, and thus it is desirable to use smaller ϵ_{train}.

C.9.2 $\epsilon_{train} = 1.1\epsilon_{test}$ on CIFAR-10

As mentioned in [27], we also train with $\epsilon_{train} = 1.1\epsilon_{test}$ on CIFAR-10. The results are shown in Table C.6. They attain slightly improved performances in $2/255$, but not in $8/255$ and larger ϵ.

Table C.4: Test errors of OURS with different numbers of gradient update steps in (C.8.7) on CIFAR-10. Here, we use constant $\kappa = 0$.

Data	$\epsilon(l_\infty)$	**OURS (1-step)**			**OURS (7-step)**		
		Standard	PGD	Verfied	Standard	PGD	Verfied
CIFAR-10	$\epsilon = 2/255$	32.04	43.11	49.62	31.40	42.30	49.20
	$\epsilon = 8/255$	56.35	67.03	70.56	54.44	66.29	71.53

Table C.5: Comparison of the performance (Standard / PGD / Verified error) depending on various ϵ_{train}. Here, we use constant $\kappa = 0$.

Data	ϵ_{test}	ϵ_{train}	**IBP**	**CROWN-IBP$_{1\to1}$**	**OURS**	**CROWN-IBP$_{1\to0}$**
MNIST	0.2	0.2	1.25 / 3.39 / 7.77	1.23 / 3.48 / 7.64	1.09 / 3.17 / 6.29	1.13 / 2.85 / 5.89
		0.3	1.95 / 2.93 / 6.28	2.89 / 5.32 / 7.61	1.70 / 3.37 / 4.76	1.48 / 2.73 / 4.79
		0.4	3.67 / 4.77 / 6.36	6.11 / 9.08 / 12.71	3.49 / 4.72 / 6.36	2.37 / 3.26 / 4.64
	0.3	0.3	1.95 / 3.31 / 12.90	2.89 / 7.35 / 14.97	1.70 / 4.82 / 9.20	1.48 / 3.52 / 9.40
		0.4	3.67 / 5.55 / 9.74	6.11 / 11.33 / 17.51	3.49 / 5.59 / 9.79	2.37 / 3.63 / 7.22

Table C.6: Comparison of the performance (Standard / PGD / Verified error) of the models trained with ϵ_{train} and $1.1\epsilon_{train}$. Here, we use constant $\kappa = 0$.

Data	ϵ_{test}	ϵ_{train}	**IBP**	**CROWN-IBP$_{1\to1}$**	**OURS**	**CROWN-IBP$_{1\to0}$**
CIFAR 10	$^2/_{255}$	$^2/_{255}$	43.6 / 52.71 / 56.58	32.15 / 42.67 / 49.36	32.04 / 43.13 / 49.62	37.25 / 47.19 / 52.53
		$^{2.2}/_{255}$	44.78 / 52.62 / 55.78	33.23 / 43.11 / 49.18	33.04 / 43.70 / 48.60	38.42 / 47.80 / 52.53
	$^8/_{255}$	$^8/_{255}$	64.11 / 70.68 / 72.99	60.96 / 70.52 / 75.68	56.35 / 67.06 / 70.56	56.95 / 67.89 / 70.43
		$^{8.8}/_{255}$	64.54 / 70.30 / 72.40	61.48 / 70.58 / 75.17	58.28 / 67.50 / 70.52	59.37 / 68.51 / 70.71

C.10 Training time

All the training times are measured on a single TITAN X (Pascal) on Medium for CIFAR-10. We train with a batch size of 128 for OURS, CROWN-IBP$_{1\to1}$ and IBP, but with a batch size of 32 for CAP due to its high memory cost. For CAP, we use random projection of 50 dimensions.

- OURS: 115.9 sec / epoch
- CROWN-IBP$_{1\to1}$: 51.68 sec / epoch
- IBP: 14.85 sec / epoch
- CAP (batch size 32, 1 GPU): 751.0 sec / epoch
- CAP (batch size 64, 1 GPU): 724.6 sec / epoch
- CAP (batch size 128, 2 GPUs): 387.9 sec / epoch

C.11 Loss and Tightness violin plots

We plot the equivalent tightness violin plots in Section 3.3.4 for models trained with other methods. The proposed method achieves the best results in terms of loss and tightness followed by CROWN-IBP, CAP-IBP, and RANDOM. Figure C.9 (a)-(b), (c)-(d), and (e)-(f) show the tightness evaluated on the model trained by CROWN-IBP$_{1\to 0}$, CROWN-IBP$_{1\to 1}$ and IBP, respectively.

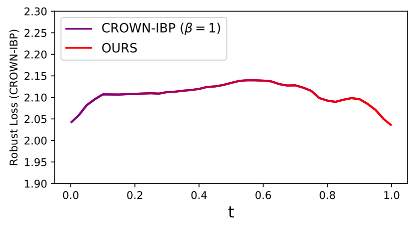
C.12 Comparison with CAP-IBP

As in section C.4, we train a model with CAP-IBP and compare with the proposed method and CROWN-IBP ($\beta = 1$). Figure C.10 shows that CAP-IBP has gradient differences larger than the proposed method and smaller than CROWN-IBP ($\beta = 1$), which leads to a performance between the proposed method and CROWN-IBP ($\beta = 1$) (see Table C.1). CAP-IBP has looser bounds than CROWN-IBP ($\beta = 1$) as shown in Figure 3.10 and Figure C.9, but with a relatively more smooth landscape, it can achieve a better performance than CROWN-IBP ($\beta = 1$).

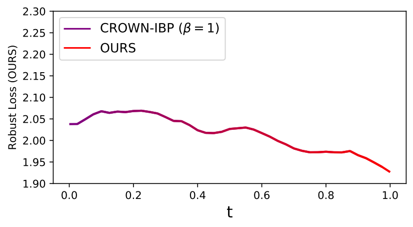
C.13 ReLU Stability

To see the effect of unstable ReLUs on smoothness, we adopt the ReLU stability loss (RS loss) $\mathcal{L}_{RS}(\boldsymbol{u}, \boldsymbol{l}) = -\tanh(1 + \boldsymbol{u} \cdot \boldsymbol{l})$ as a regularizer [74]. We use $\mathcal{L} + \lambda\mathcal{L}_{RS}$ as a loss and run CROWN-IBP ($\beta = 1$) with various λ settings. We plot the smoothness and the tightness in Figure C.11 and Figure C.12 on $\lambda = 0$, $\lambda = 0.01$, $\lambda = 10$.

We found that small λ suggested in [74] has no effect on reducing the number of unstable ReLUs, and thus not on improving the smoothness as shown in Figure C.7. By increasing λ, we observed that RS successfully reduces the number of unstable ReLUs with $\lambda = 10$. Figure C.11 shows that large λ leads to a smaller loss variation and gradient difference. This supports that unstable ReLUs are closely related to the smoothness of the loss landscape. However, as [74] mentioned ”placing too much weight on RS Loss can decrease the model capacity, potentially lowering the provable adversarial accuracy", the models trained with a large $\lambda \geq 1$ couldn't obtain a tightness of the upper bound and significant improvement on robustness as illustrated in Figure C.12. The test errors (Standard / PGD / Verified) are 0.6278 / 0.7189 / 0.7634 on $\lambda = 0.01$ and 0.6090 / 0.7085 / 0.7600 on $\lambda = 10$.



(a) CROWN-IBP bound



(b) OURS bound

Figure C.5: Mode connectivity between CROWN-IBP and OURS, where $\boldsymbol{w}_0$ and $\boldsymbol{w}_1$ are well-trained models using CROWN-IBP bound and OURS bound, respectively. $\boldsymbol{\theta}_c$ is trained using CROWN-IBP (C.5a) and OURS (C.5b), respectively.

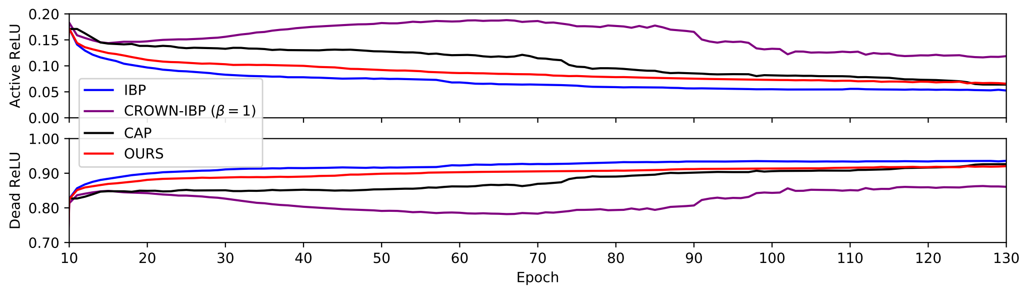
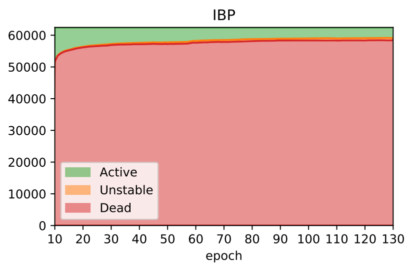
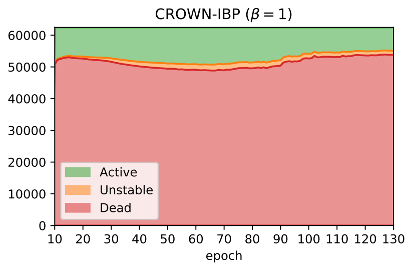


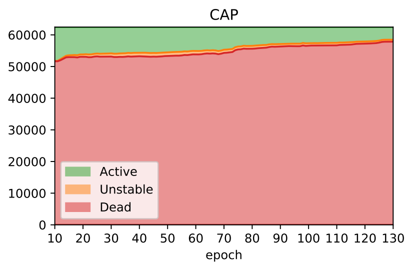
Figure C.6: The ratio of the number of active (*top*) and dead (*bottom*) ReLUs during the ramp-up period.



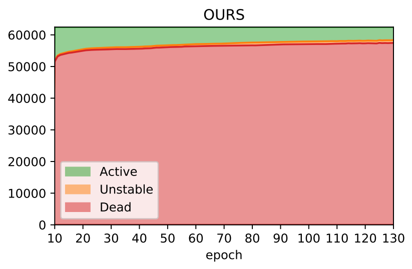
(a) IBP



(b) CROWN-IBP ($\beta = 1$)

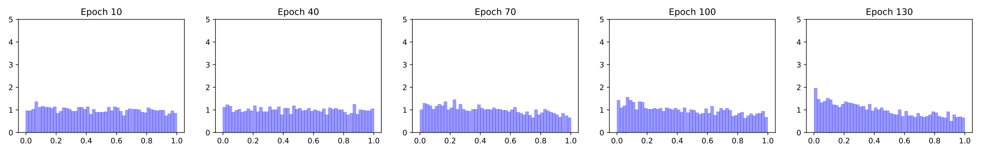


(c) CAP

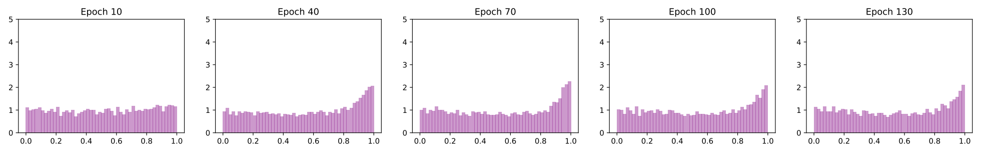


(d) OURS

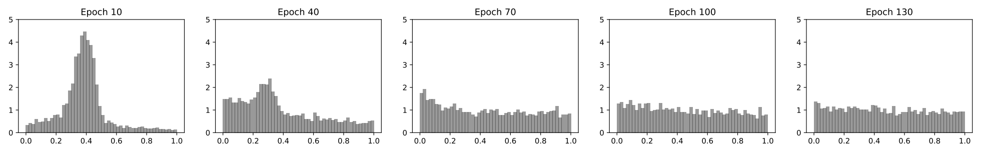
Figure C.7: Number of active (Green), unstable (Orange), and dead (Red) ReLUs.



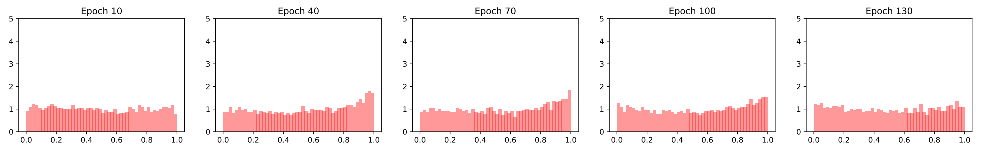
(a) IBP



(b) CROWN-IBP

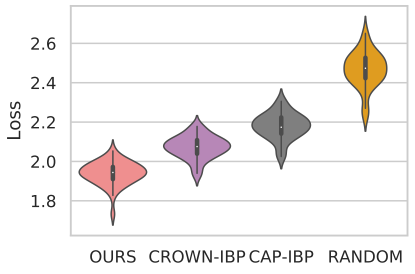


(c) CAP

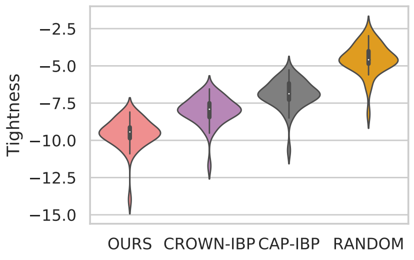


(d) OURS

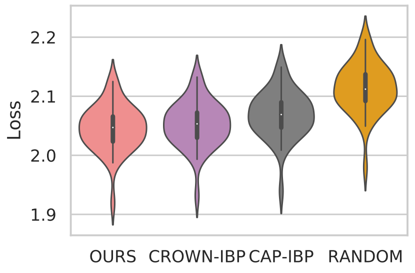
Figure C.8: Histograms of the distribution of the slope $\frac{u^+}{u^+ - l^-}$ when $l \leq 0 \leq u$ during the ramp-up period.



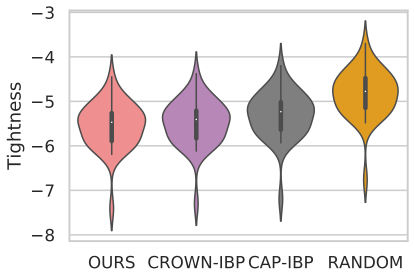
(a) the loss of CROWN-IBP$_{1\to 0}$



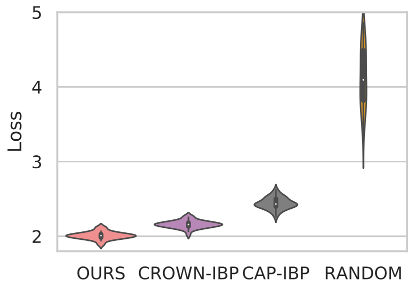
(b) the tightness of CROWN-IBP$_{1\to 0}$



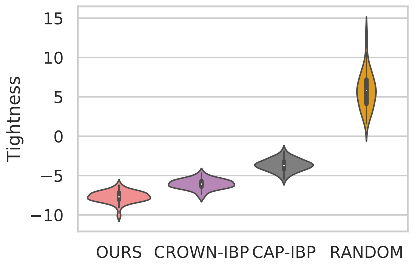
(c) the loss of CROWN-IBP$_{1\to 1}$



(d) the tightness of CROWN-IBP$_{1\to 1}$



(e) the loss of IBP



(f) the tightness of IBP

Figure C.9: Violin plots of the test loss (*Left Column*) and of tightness (*Right Column*) for various linear relaxations same as in Section 3.3.4. Lower is better.

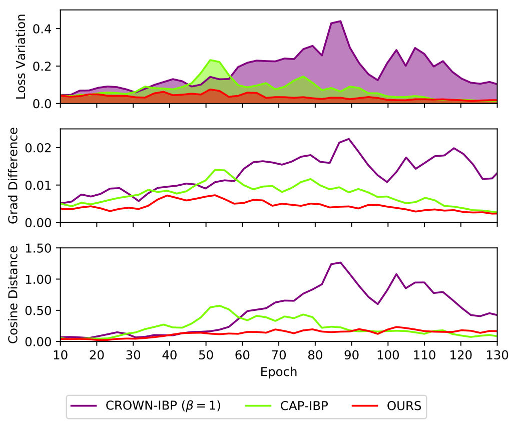


Figure C.10: (Top) Loss variations along the gradient descent direction, (Middle) ℓ_2-distance between two consecutive loss gradients and (Bottom) the cosine distance between them during the ramp-up phase.

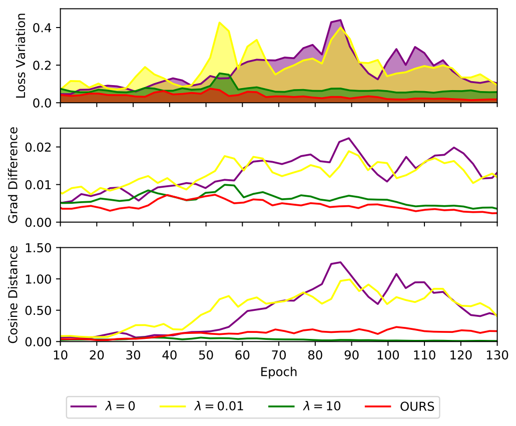


Figure C.11: (Top) Loss variations along the gradient descent direction, (Middle) ℓ_2-distance between two consecutive loss gradients and (Bottom) the cosine distance between them during the ramp-up phase on CROWN-IBP ($\beta = 1$) with $\lambda = 0$, $\lambda = 0.01$, $\lambda = 10$ and OURS.

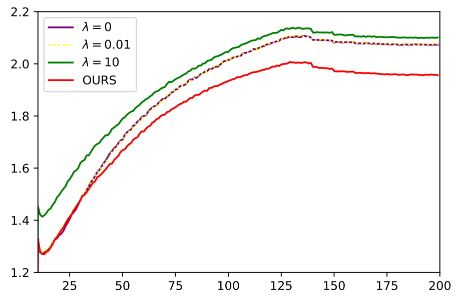


Figure C.12: Robust loss of CROWN-IBP ($\beta = 1$) with $\lambda = 0$, $\lambda = 0.01$, $\lambda = 10$ and OURS during training.

Bibliography

[1] Athalye, A., Carlini, N., and Wagner, D. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *International Conference on Machine Learning* (2018), PMLR, pp. 274–283.

[2] Athalye, A., Engstrom, L., Ilyas, A., and Kwok, K. Synthesizing robust adversarial examples. *arXiv preprint arXiv:1707.07397* (2017).

[3] Balunovic, M., and Vechev, M. Adversarial training and provable defenses: Bridging the gap. In *International Conference on Learning Representations* (2019).

[4] Banerjee, A., Dhillon, I. S., Ghosh, J., and Sra, S. Clustering on the unit hypersphere using von mises-fisher distributions. *Journal of Machine Learning Research 6*, Sep (2005), 1345–1382.

[5] Bartlett, P. L., Foster, D. J., and Telgarsky, M. J. Spectrally-normalized margin bounds for neural networks. In *Advances in Neural Information Processing Systems* (2017), pp. 6240–6249.

[6] Bassily, R., Belkin, M., and Ma, S. On exponential convergence of sgd in non-convex over-parametrized learning. *arXiv preprint arXiv:1811.02564* (2018).

[7] Bhagoji, A. N., He, W., Li, B., and Song, D. Exploring the space of black-box attacks on deep neural networks. *arXiv preprint arXiv:1712.09491* (2017).

[8] Biggio, B., Corona, I., Maiorca, D., Nelson, B., Šrndić, N., Laskov, P., Giacinto, G., and Roli, F. Evasion attacks against machine learning at test time. In *Joint European conference on machine learning and knowledge discovery in databases* (2013), Springer, pp. 387–402.

[9] Borodin, A., and Rains, E. M. Eynard–mehta theorem, schur process, and their pfaffian analogs. *Journal of statistical physics 121*, 3-4 (2005), 291–317.

[10] Carlini, N., Athalye, A., Papernot, N., Brendel, W., Rauber, J., Tsipras, D., Goodfellow, I., Madry, A., and Kurakin, A. On evaluating adversarial robustness. *arXiv preprint arXiv:1902.06705* (2019).

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

[12] Chen, P.-Y., Zhang, H., Sharma, Y., Yi, J., and Hsieh, C.-J. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security* (2017), ACM, pp. 15–26.

[13] Coates, A., Ng, A., and Lee, H. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics* (2011), pp. 215–223.

[14] Cohen, J., Rosenfeld, E., and Kolter, Z. Certified adversarial robustness via randomized smoothing. In *International Conference on Machine Learning* (2019), pp. 1310–1320.

[15] Croce, F., Andriushchenko, M., Sehwag, V., Flammarion, N., Chiang, M., Mittal, P., and Hein, M. Robustbench: a standardized adversarial robustness benchmark. *arXiv preprint arXiv:2010.09670* (2020).

[16] Croce, F., and Hein, M. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. *arXiv preprint arXiv:2003.01690* (2020).

[17] Dhillon, G. S., Azizzadenesheli, K., Lipton, Z. C., Bernstein, J., Kossaifi, J., Khanna, A., and Anandkumar, A. Stochastic activation pruning for robust adversarial defense. *arXiv preprint arXiv:1803.01442* (2018).

[18] Dvijotham, K., Gowal, S., Stanforth, R., Arandjelovic, R., O'Donoghue, B., Uesato, J., and Kohli, P. Training verified learners with learned verifiers. *arXiv preprint arXiv:1805.10265* (2018).

[19] Dvijotham, K., Stanforth, R., Gowal, S., Mann, T. A., and Kohli, P. A dual approach to scalable verification of deep networks. In *UAI* (2018), pp. 550–559.

[20] Eykholt, K., Evtimov, I., Fernandes, E., Li, B., Rahmati, A., Xiao, C., Prakash, A., Kohno, T., and Song, D. Robust physical-world attacks on deep learning models. *arXiv preprint arXiv:1707.08945* (2017).

[21] Farnia, F., Zhang, J. M., and Tse, D. Generalizable adversarial training via spectral normalization. *arXiv preprint arXiv:1811.07457* (2018).

[22] Farouki, R. T. The bernstein polynomial basis: A centennial retrospective. *Computer Aided Geometric Design 29*, 6 (2012), 379–419.

[23] Fazlyab, M., Robey, A., Hassani, H., Morari, M., and Pappas, G. J. Efficient and accurate estimation of lipschitz constants for deep neural networks. *arXiv preprint arXiv:1906.04893* (2019).

[24] Finlayson, S. G., Chung, H. W., Kohane, I. S., and Beam, A. L. Adversarial attacks against medical deep learning systems. *arXiv preprint arXiv:1804.05296* (2018).

[25] Garipov, T., Izmailov, P., Podoprikhin, D., Vetrov, D. P., and Wilson, A. G. Loss surfaces, mode connectivity, and fast ensembling of dnns. In *Advances in Neural Information Processing Systems* (2018), pp. 8789–8798.

[26] Goodfellow, I. J., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2014).

[27] Gowal, S., Dvijotham, K., Stanforth, R., Bunel, R., Qin, C., Uesato, J., Mann, T., and Kohli, P. On the effectiveness of interval bound propagation for training verifiably robust models. *arXiv preprint arXiv:1810.12715* (2018).

[28] Gowal, S., Qin, C., Uesato, J., Mann, T., and Kohli, P. Uncovering the limits of adversarial training against norm-bounded adversarial examples. *arXiv preprint arXiv:2010.03593* (2020).

[29] Guo, C., Rana, M., Cisse, M., and van der Maaten, L. Countering adversarial images using input transformations. *arXiv preprint arXiv:1711.00117* (2017).

[30] Hein, M., and Andriushchenko, M. Formal guarantees on the robustness of a classifier against adversarial manipulation. In *Advances in Neural Information Processing Systems* (2017), pp. 2266–2276.

[31] Huster, T., Chiang, C.-Y. J., and Chadha, R. Limitations of the lipschitz constant as a defense against adversarial examples. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases* (2018), Springer, pp. 16–29.

[32] Ilyas, A., Engstrom, L., Athalye, A., and Lin, J. Black-box adversarial attacks with limited queries and information. *arXiv preprint arXiv:1804.08598* (2018).

[33] Ilyas, A., Engstrom, L., and Madry, A. Prior convictions: Black-box adversarial attacks with bandits and priors. *arXiv preprint arXiv:1807.07978* (2018).

[34] Ioffe, S., and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167* (2015).

[35] Katz, G., Barrett, C., Dill, D. L., Julian, K., and Kochenderfer, M. J. Reluplex: An efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification* (2017), Springer, pp. 97–117.

[36] Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. Tech. rep., Citeseer, 2009.

[37] Kulesza, A., Taskar, B., et al. Determinantal point processes for machine learning. *Foundations and Trends® in Machine Learning 5*, 2–3 (2012), 123–286.

[38] Kurakin, A., Goodfellow, I., and Bengio, S. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533* (2016).

[39] Laidlaw, C., Singla, S., and Feizi, S. Perceptual adversarial robustness: Defense against unseen threat models. *arXiv preprint arXiv:2006.12655* (2020).

[40] Latorre, F., Rolland, P., and Cevher, V. Lipschitz constant estimation of neural networks via sparse polynomial optimization. *arXiv preprint arXiv:2004.08688* (2020).

[41] LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86*, 11 (1998), 2278–2324.

[42] Lecuyer, M., Atlidakis, V., Geambasu, R., Hsu, D., and Jana, S. Certified robustness to adversarial examples with differential privacy. In *2019 IEEE Symposium on Security and Privacy (SP)* (2019), IEEE, pp. 656–672.

[43] Lee, S., Kim, H., and Lee, J. Graddiv: Adversarial robustness of randomized neural networks via gradient diversity regularization. *arXiv preprint arXiv:2107.02425* (2021).

[44] Lee, S., Lee, J., and Park, S. Lipschitz-certifiable training with a tight outer bound. *Advances in Neural Information Processing Systems 33* (2020).

[45] Lee, S., Lee, W., Park, J., and Lee, J. Loss landscape matters: Training certifiably robust models with favorable loss landscape.

[46] Li, B., Chen, C., Wang, W., and Carin, L. Second-order adversarial attack and certifiable robustness. *arXiv preprint arXiv:1809.03113* (2018).

[47] Liu, C., Zhu, L., and Belkin, M. Toward a theory of optimization for over-parameterized systems of non-linear equations: the lessons of deep learning. *arXiv preprint arXiv:2003.00307* (2020).

[48] Liu, X., Cheng, M., Zhang, H., and Hsieh, C.-J. Towards robust neural networks via random self-ensemble. In *Proceedings of the European Conference on Computer Vision (ECCV)* (2018), pp. 369–385.

[49] Liu, X., Li, Y., Wu, C., and Hsieh, C.-J. Adv-bnn: Improved adversarial defense through robust bayesian neural network. *arXiv preprint arXiv:1810.01279* (2018).

[50] Liu, Y., Chen, X., Liu, C., and Song, D. Delving into transferable adversarial examples and black-box attacks. *arXiv preprint arXiv:1611.02770* (2016).

[51] Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083* (2017).

[52] Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations* (2018).

[53] Mardia, K. V., and Jupp, P. E. *Directional statistics*, vol. 494. John Wiley & Sons, 2009.

[54] Mirman, M., Gehr, T., and Vechev, M. Differentiable abstract interpretation for provably robust neural networks. In *International Conference on Machine Learning* (2018), pp. 3575–3583.

[55] Moore, R. E., Kearfott, R. B., and Cloud, M. J. *Introduction to interval analysis*, vol. 110. Siam, 2009.

[56] Papernot, N., McDaniel, P., and Goodfellow, I. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *arXiv preprint arXiv:1605.07277* (2016).

[57] Papernot, N., McDaniel, P., Goodfellow, I., Jha, S., Celik, Z. B., and Swami, A. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security* (2017), ACM, pp. 506–519.

[58] Papernot, N., McDaniel, P., Wu, X., Jha, S., and Swami, A. Distillation as a defense to adversarial perturbations against deep neural networks. In *2016 IEEE Symposium on Security and Privacy (SP)* (2016), IEEE, pp. 582–597.

[59] Raghunathan, A., Steinhardt, J., and Liang, P. S. Semidefinite relaxations for certifying robustness to adversarial examples. In *Advances in Neural Information Processing Systems* (2018), pp. 10877–10887.

[60] Rebuffi, S.-A., Gowal, S., Calian, D. A., Stimberg, F., Wiles, O., and Mann, T. Fixing data augmentation to improve adversarial robustness. *arXiv preprint arXiv:2103.01946* (2021).

[61] Salman, H., Li, J., Razenshteyn, I., Zhang, P., Zhang, H., Bubeck, S., and Yang, G. Provably robust deep learning via adversarially trained smoothed classifiers. In *Advances in Neural Information Processing Systems* (2019), pp. 11289–11300.

[62] Santurkar, S., Tsipras, D., Ilyas, A., and Madry, A. How does batch normalization help optimization? In *Advances in Neural Information Processing Systems* (2018), pp. 2483–2493.

[63] Sharif, M., Bhagavatula, S., Bauer, L., and Reiter, M. K. Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (2016), ACM, pp. 1528–1540.

[64] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199* (2013).

[65] Tjeng, V., Xiao, K., and Tedrake, R. Evaluating robustness of neural networks with mixed integer programming. *arXiv preprint arXiv:1711.07356* (2017).

[66] Tramer, F., Carlini, N., Brendel, W., and Madry, A. On adaptive attacks to adversarial example defenses. *arXiv preprint arXiv:2002.08347* (2020).

[67] Tramèr, F., Kurakin, A., Papernot, N., Goodfellow, I., Boneh, D., and McDaniel, P. Ensemble adversarial training: Attacks and defenses. *arXiv preprint arXiv:1705.07204* (2017).

[68] Tramèr, F., Papernot, N., Goodfellow, I., Boneh, D., and McDaniel, P. The space of transferable adversarial examples. *arXiv preprint arXiv:1704.03453* (2017).

[69] Tsuzuku, Y., Sato, I., and Sugiyama, M. Lipschitz-margin training: Scalable certification of perturbation invariance for deep neural networks. In *Advances in Neural Information Processing Systems* (2018), pp. 6541–6550.

[70] Wong, E., and Kolter, J. Z. Provable defenses against adversarial examples via the convex outer adversarial polytope. *arXiv preprint arXiv:1711.00851* (2017).

[71] Wong, E., and Kolter, Z. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International Conference on Machine Learning* (2018), PMLR, pp. 5286–5295.

[72] Wong, E., Schmidt, F., Metzen, J. H., and Kolter, J. Z. Scaling provable adversarial defenses. In *Advances in Neural Information Processing Systems* (2018), pp. 8400–8409.

[73] Wu, D., Xia, S.-T., and Wang, Y. Adversarial weight perturbation helps robust generalization. *Advances in Neural Information Processing Systems 33* (2020).

[74] Xiao, K. Y., Tjeng, V., Shafiullah, N. M., and Madry, A. Training for faster adversarial robustness verification via inducing relu stability. *arXiv preprint arXiv:1809.03008* (2018).

[75] Xie, C., Wang, J., Zhang, Z., Ren, Z., and Yuille, A. Mitigating adversarial effects through randomization. *arXiv preprint arXiv:1711.01991* (2017).

[76] Xie, C., Wu, Y., Maaten, L. v. d., Yuille, A. L., and He, K. Feature denoising for improving adversarial robustness. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2019), pp. 501–509.

[77] Zagoruyko, S., and Komodakis, N. Wide residual networks. *arXiv preprint arXiv:1605.07146* (2016).

[78] Zhang, H., Chen, H., Xiao, C., Gowal, S., Stanforth, R., Li, B., Boning, D., and Hsieh, C.-J. Towards stable and efficient training of verifiably robust neural networks. In *International Conference on Learning Representations* (2019).

[79] Zhang, H., Weng, T.-W., Chen, P.-Y., Hsieh, C.-J., and Daniel, L. Efficient neural network robustness certification with general activation functions. In *Advances in neural information processing systems* (2018), pp. 4939–4948.

[80] Zhang, H., Yu, Y., Jiao, J., Xing, E. P., Ghaoui, L. E., and Jordan, M. I. Theoretically principled trade-off between robustness and accuracy. *arXiv preprint arXiv:1901.08573* (2019).

[81] Zhao, P., Chen, P.-Y., Das, P., Ramamurthy, K. N., and Lin, X. Bridging mode connectivity in loss landscapes and adversarial robustness. *arXiv preprint arXiv:2005.00060* (2020).

국문초록

딥러닝은 다양한 분야에서 성공적인 성능를 보여주고 있다. 그러나 심층 신경망은 적대적 공격이라 불리우는, 입력값에 작은 섭동을 주어 신경망을 사용자가 원치 않는 방향으로 행동하도록 하는 공격에 취약하다. 적대적 공격의 발견 이후로, 다양한 적대적 공격과 이에 대한 방어 방법론과 관련하여 많은 연구들이 진행되었다. 그러나 Athalye *et al.* [1] 에서 대부분의 기존 방어 방법론들이 특정 적대적 공격만을 가정하고 설계되어 더 강한 적응가능한 적대적 공격에 의해 공격 가능하다는 문제점이 밝혀졌다. 따라서 입력값에 대해 섭동가능한 영역내에서 안정적인 행동을 보증할 수 있는 검증가능한 방법론이 제안되어왔다. 본 학위 논문에서는, 휴리스틱 방법론과 검증가능한 방법론에 대해 알아보고, 검증가능한 방법론에서 중요한 요소인 *상한의 밀착성과 목적 함수의 매끄러움*에 대해서 분석한다.

주요어휘: 딥러닝, 적대적 강건성, 검증가능한 방어
학번: 2016-28750