Robustra: Training Provable Robust Neural Networks over Reference Adversarial Space

Linyi Li[*] , **Zexuan Zhong**[*] , **Bo Li** and **Tao Xie**
University of Illinois at Urbana-Champaign
{linyi2, zexuan2, lbo, taoxie}@illinois.edu

Abstract

Machine learning techniques, especially deep neural networks (DNNs), have been widely adopted in various applications. However, DNNs are recently found to be vulnerable against adversarial examples, i.e., maliciously perturbed inputs that can mislead the models to make arbitrary prediction errors. Empirical defenses have been studied, but many of them can be adaptively attacked again. Provable defenses provide provable error bound of DNNs, while such bound so far is far from satisfaction. To address this issue, in this paper, we present our approach named Robustra for effectively improving the provable error bound of DNNs. We leverage the adversarial space of a reference model as the feasible region to solve the min-max game between the attackers and defenders. We solve its dual problem by linearly approximating the attackers' best strategy and utilizing the monotonicity of the slack variables introduced by the reference model. The evaluation results show that our approach can provide significantly better provable adversarial error bounds on MNIST and CIFAR10 datasets, compared to the state-of-the-art results. In particular, bounded by ℓ_∞, with $\epsilon = 0.1$, on MNIST we reduce the error bound from 2.74% to 2.09%; with $\epsilon = 0.3$, we reduce the error bound from 24.19% to 16.91%.

1 Introduction

In recent years, machine learning techniques, especially deep neural networks (DNNs), have been widely adopted in various applications, such as image classification and machine translation. However, DNNs are recently found to be vulnerable against adversarial examples, i.e., maliciously perturbed inputs that can mislead the models to make arbitrary prediction errors [Szegedy *et al.*, 2014]. Such malicious perturbations are usually imperceptible to human eyes, but allow attackers to easily mislead the behavior of DNNs. The existence of adversarial examples impedes application of DNNs in domains where the safety is highly sensitive, such as autonomous driving and access control systems. Although a series of heuristic defenses [Papernot *et al.*, 2016; Samangouei *et al.*, 2018] have been proposed to defend against existing adversarial examples, many of these defenses can be adaptively attacked again [Carlini and Wagner, 2017b; Athalye *et al.*, 2018]. How to train DNNs that are guaranteed to be robust to any attack serves as an open problem to the community.

Recently, provable defenses [Wong and Kolter, 2018; Wong *et al.*, 2018] have been proposed to train DNNs with low *provable error bound*, which refers to the fraction of test set inputs for which there is provably no adversarial example within allowed perturbations. However, provable error bounds of DNNs so far are far from satisfaction. For example, on CIFAR10, the state-of-the-art provable error bound is still as high as 46.11%, when bounded by ℓ_∞, with $\epsilon = 2/255$ [Tjeng *et al.*, 2019].

To address the issue, in this paper, we present our approach named Robustra for effectively improving the provable error bound of ReLU-based DNNs. We formalize the target task as a min-max game between the attackers and defenders. Different from previous work [Wong and Kolter, 2018] that solves the min-max optimization over the whole norm-bounded space, we additionally leverage the adversarial space of a reference model as the feasible region. Specifically, given a reference model, our formalized problem seeks to train a model that is robust in the adversarial space of the reference model, i.e., the adversarial examples cannot transfer to the model that we train.

Solving such a non-convex problem is non-trivial. We use linear approximation for ReLU activations in the DNNs, and solve its dual problem. The constraint related to the reference adversarial space introduces a family of slack variables in the dual problem, causing no existing approach to effectively solve it. To address the problem, by leveraging the monotonicity and boundedness of the slack variables' gradients, only querying the gradients once, our approach directly calculates out the solution for slack variables. Thus, we solve the problem in the same order of time as the previous work [Wong *et al.*, 2018]. Furthermore, by using similar techniques introduced in the previous work [Wong *et al.*, 2018], Robustra is scalable to be applied to large models such as ResNet. Moreover, we train a pair of models mutually, i.e., iteratively using one as training model and the other as refer-

[*]Both authors contributed equally to this work.

ence, and thus we obtain a pair of robust models at the same time, without requiring external reference.

We evaluate Robustra on the MNIST and CIFAR10 datasets. The evaluation results show that our approach can provide significantly better provable adversarial error bounds on both datasets, compared to the state-of-the-art results. Specifically, we reduce the error bound from 24.19% to 16.91% with ℓ_∞ norm $\epsilon = 0.3$ on MNIST, and from 46.11% to 43.68% with ℓ_∞ norm $\epsilon = 2/255$ on CIFAR10.

In summary, this paper makes the following main contributions:

- We propose Robustra, a novel approach for training robust ReLU-based DNNs. Robustra formalizes the problem of robust training as a min-max optimization over the adversarial space of a reference model.
- We propose an algorithm that leverages the monotonicity and boundedness of the slack variables in the dual problem to efficiently solve the optimization problem.
- We evaluate Robustra on both MNIST and CIFAR10 datasets. The evaluation results show that Robustra produces DNNs with significantly better provable adversarial error bounds compared to the state-of-the-art results.

Our code and model weights are available at `https://github.com/llylly/Robustra`.

2 Preliminaries

In this work, we consider a pair of ReLU-based DNNs f and g, where f is the trainable model, and g is the reference model. We refer to f as f_θ in some places to explicitly emphasize the trainable parameters θ.

Let n denote the dimension of input vector, and m denote the number of classes. The ReLU function is $\mathrm{ReLU}(x) = \max(x, 0)$. f and g are formally defined as follows:

$$\begin{cases} z_0 = x \in \mathbb{R}^n, \\ z_i = \mathrm{ReLU}(W_i z_{i-1} + b_i), 1 \le i \le k_1 \\ f(x) = z_{k_1} \in \mathbb{R}^m. \end{cases}$$

$$\begin{cases} w_0 = x \in \mathbb{R}^n, \\ w_i = \mathrm{ReLU}(U_i w_{i-1} + d_i), 1 \le i \le k_2 \\ g(x) = w_{k_2} \in \mathbb{R}^m. \end{cases}$$

Namely, f (resp. g): $\mathbb{R}^n \to \mathbb{R}^m$ is a k_1 (resp. k_2)-layer neural network with parameters $W_i, b_i, 1 \le i \le k_1$ (resp. $U_i, b_i, 1 \le i \le k_2$). All activations functions are ReLU functions in f and g.

Given an input x, we have $f(x), g(x) \in \mathbb{R}^m$, representing the predicted probability for each class. Let $\big(f(x)\big)_i$ (resp. $\big(g(x)\big)_i$) represent the i^{th} dimension of $f(x)$ (resp. $g(x)$), i.e., the predicted probability of class i.

In this paper, we use ℓ_∞ norm to measure the adversarial perturbation. We also support to work on ℓ_1 norm or ℓ_2 norm by applying the techniques introduced in [Wong *et al.*, 2018].

3 Related Work and Background

Attacks and Empirical Defenses. Szegedy *et al.* [2014] discover the broad existence of adversarial examples in neural networks. From then on, various attack approaches have been proposed, such as FGSM [Goodfellow *et al.*, 2015], CW [Carlini and Wagner, 2017b; Carlini and Wagner, 2017a], and PGD [Madry *et al.*, 2018]. At the same time, heuristic defenses have been proposed, such as Distillation [Papernot *et al.*, 2016] and Defense-GAN [Samangouei *et al.*, 2018]. Although these heuristic defenses have empirically shown effectiveness against existing attacks, they can be adaptively attacked again after being proposed [Athalye *et al.*, 2018; Carlini, 2019].

Provable Defenses. Provable defenses train DNNs with low provable error bound, i.e., they can provably guarantee robustness for a fraction of test set inputs against any attacks for the trained DNNs. Existing provable defenses use the upper bound of loss function as (part or all of) the training objective. Existing upper-bound derivation approaches include SDP [Raghunathan *et al.*, 2018], LP-dual [Wong and Kolter, 2018; Wong *et al.*, 2018], and LP [Mirman *et al.*, 2018; Singh *et al.*, 2018]. Besides, concurrent work by Xiao *et al.* [2019] combines PGD adversarial training with regularization for weight sparsity and ReLU stability. However, provable error bounds of DNNs obtained from these existing approaches are far from satisfaction.

DNN Verification. DNN verification approaches determine provable error bound for trained DNNs. Reluplex [Katz *et al.*, 2017] pioneers the field but is not scalable to verify DNNs with more than a thousand neurons. By using Mixed Integer Programming, Tjeng *et al.* [2019] provide a much faster approach for DNN verification. Other approaches provide relaxed provable error bound by using LP-based relaxations [Weng *et al.*, 2018; Singh *et al.*, 2019; Salman *et al.*, 2019] or SDP-based relaxations [Fazlyab *et al.*, 2019].

4 Motivation: An Empirical Observation

Our approach is inspired by an empirical observation: during training, merely limiting overlap of the adversarial space with a reference model can make the model internally robust. This observation indicates that the adversarial space of one model should be defended against with high priority.

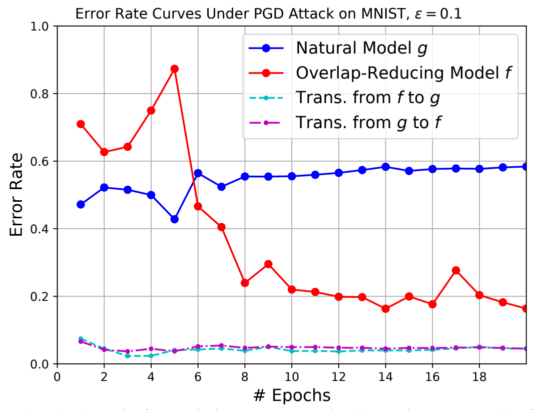


Figure 1: A heuristic training approach that aims at reducing the overlap of adversarial space to a reference model can make the model internally robust (see red curve).

Specifically, given an input x and its true label y, a naturally trained reference model $g(\cdot)$, we train a model $f(\cdot)$ by modifying the training objective from original loss function $\ell_f \overset{def}{=} \ell(f(x), y)$ to $\ell_f + \lambda|\cos\langle\nabla_x \ell_f, \nabla_x \ell_g\rangle|$, where λ is a tunable hyper parameter, being set to 0.1; g, the reference model, is a naturally trained model. Our intuitive approach aims at guiding the local gradients between two models to be diverse (orthogonal) in terms of their cosine angle, hence reducing the overlap of their adversarial spaces.

Figure 1 shows the error curves by training epochs. The error rate is measured by the success rate of untargeted PGD attack bounded by ℓ_∞ with $\epsilon = 0.1$. The blue curve represents the natural trained model g, where as the natural training goes, the error rate slightly increases. The two dashed curves represent the transfer attack's success rate, where adversarial examples are generated from f/g model and used to attack the other model. The curves show that the transfer error rate is reduced as expected. The red curve represents the error rate of model f. We find that, after a few epochs, the error rate of model f decreases below 20% although we never deliberately train the model to be self-robust.

This observation inspires us to train a robust model constrained on the adversarial space of another model.

5 Robustra

In this section, we introduce Robustra in detail.

5.1 Min-Max Problem Formulation

We formalize our target task as a min-max game.

Given an input sample x^*, the true label y^*, and some target label $y^{\text{targ}} \neq y^*$, we define $\mathcal{A}_\epsilon(x^*)$, the adversarial space of g bounded by ℓ_∞ norm with ϵ, as follows:

$$\mathcal{A}_\epsilon(x^*) = \left\{x : (\|x - x^*\|_\infty \le \epsilon) \wedge \left((g(x))_{y^{\text{targ}}} - (g(x))_{y^*} \ge t\right)\right\}, \tag{1}$$

where t measures the margin between the decision boundary of adversarial examples in $\mathcal{A}_\epsilon(x^*)$. Particularly, when $t = 0$, $\mathcal{A}_\epsilon(x^*)$ is the exact adversarial space of g; when t decreases, $\mathcal{A}_\epsilon(x^*)$ becomes larger, vice versa.

We want the model f to be robust to adversarial examples of the reference model g. Specifically, for each input in the adversarial space $\mathcal{A}_\epsilon(x^*)$, the output of f has a larger value at y^* than at y^{targ}. We formalize the objective as the following optimization problem:

$$\begin{aligned} \forall y^{\text{targ}} \neq y^*, \quad & \min_\theta \left(\max_x (f_\theta(x))_{y^{\text{targ}}} - (f_\theta(x))_{y^*}\right), \\ \text{s.t.} \quad & x \in \mathcal{A}_\epsilon(x^*). \end{aligned} \tag{2}$$

Let $c = e_{y^{\text{targ}}} - e_{y^*}$, and let $\mathcal{F}_\mathcal{A}$ denote the value space of $\mathcal{A}_\epsilon(x^*)$, i.e.,

$$\mathcal{F}_\mathcal{A}(x^*) = \{f_\theta(x) : x \in \mathcal{A}_\epsilon(x^*)\}. \tag{3}$$

Then, the optimization (Eqn. 2) can be rewritten as follows:

$$\begin{aligned} \forall y^{\text{targ}} \neq y^*, \quad & \min_\theta \max_v c^\top v, \\ \text{s.t.} \quad & v \in \mathcal{F}_\mathcal{A}(x^*). \end{aligned} \tag{4}$$

Since the set $\mathcal{F}_\mathcal{A}(x^*)$ is consecutive and highly non-convex, solving such optimization problem is computationally intractable.

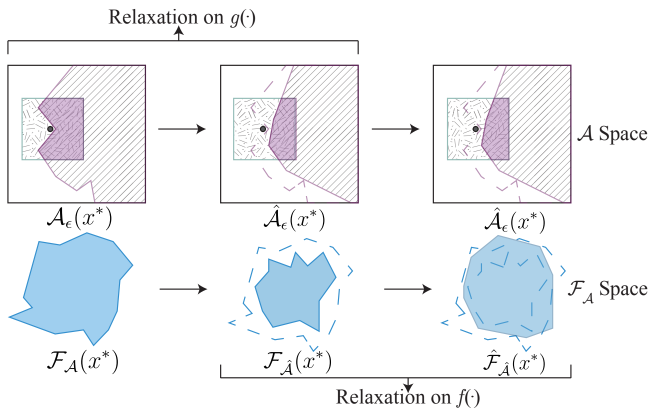

Relaxation on g(·)
A Space
$\mathcal{A}_\epsilon(x^*)$
$\hat{\mathcal{A}}_\epsilon(x^*)$
$\hat{\mathcal{A}}_\epsilon(x^*)$
$\mathcal{F}_\mathcal{A}$ Space
$\mathcal{F}_\mathcal{A}(x^*)$
$\mathcal{F}_{\hat{\mathcal{A}}}(x^*)$
$\hat{\mathcal{F}}_{\hat{\mathcal{A}}}(x^*)$
Relaxation on f(·)

Figure 2: Visual illustration of approximation by linear convex relaxation. From the exact space $\mathcal{A}_\epsilon(x^*)$, the relaxation of model g brings convex space $\hat{\mathcal{A}}_\epsilon(x^*)$. Then, the relaxation of model f brings convex space $\hat{\mathcal{F}}_{\hat{\mathcal{A}}}(x^*)$.

5.2 Linear Convex Relaxation

In order to solve the optimization problem, we approximate the optimization space $\mathcal{F}_\mathcal{A}(x^*)$ of the problem by using relaxation of the ReLU function. A convex space $\hat{\mathcal{F}}_{\hat{\mathcal{A}}}(x^*)$ is obtained from the approximation.

Since both $f(\cdot)$ and $g(\cdot)$ are ReLU-based DNNs, we use the linear relaxation for ReLU function. Let $o = \text{ReLU}(i)$ denote the ReLU function and $[l, u]$ denote the range of input i, i.e., $l \le i \le u$. The approximation of ReLU function corresponds to its convex outer space:

1. When $l \le u \le 0$, $o = 0$;
2. When $0 \le l \le u$, $o = i$;
3. When $l < 0 < u$, $(u - l)o \le u(i - l) \wedge o \ge 0 \wedge o \ge i$.

We replace each ReLU function in the problem with corresponding linear constraints according to the range of the input.

We apply the relaxation to both $f(\cdot)$ and $g(\cdot)$ models, as shown in Figure 2.

First, we apply the relaxation to the ReLU neurons in model $g(\cdot)$. Let $\hat{G}(\cdot)$ denote the output space after applying the relaxation. Note that $\hat{G}(x) \subseteq \mathbb{R}^m$, while $g(x) \in \mathbb{R}^m$. Accordingly, the adversarial space $\mathcal{A}_\epsilon(x^*)$ is replaced with its approximation subspace $\hat{\mathcal{A}}_\epsilon(x^*)$:

$$\hat{\mathcal{A}}_\epsilon(x^*) = \left\{x : (\|x - x^*\|_\infty \le \epsilon) \wedge \left(\forall v' \in \hat{G}(x) : c^\top v' \ge t\right)\right\}. \tag{5}$$

Note that $\hat{\mathcal{A}}_\epsilon(x^*)$ is a convex subspace of $\mathcal{A}_\epsilon(x^*)$, because $\hat{G}(x)$ is a convex outerspace of $g(x)$. Then, we obtain an approximated optimization subspace $\mathcal{F}_{\hat{\mathcal{A}}}(x^*)$ (shown in Figure 2).

Second, we similarly apply the relaxation to the ReLU neurons in model $f(\cdot)$. Let $\hat{F}(\cdot)$ denote the output space after applying the relaxation. Then, we obtain $\hat{\mathcal{F}}_{\hat{\mathcal{A}}}(x^*) = \bigcup_{x \in \hat{\mathcal{A}}_\epsilon(x^*)} \hat{F}(x)$, which is a convex approximation space of $\mathcal{F}_\mathcal{A}(x^*)$.

After the two-step relaxation, the optimization problem that we would like to address is as follows:

$$\begin{aligned} &\forall y^{\text{targ}} \neq y^*, \quad \min_{\theta} \max_{v} c^\top v, \\ \text{s.t.} \qquad & v \in \hat{\mathcal{F}}_{\hat{\mathcal{A}}}(x^*). \end{aligned} \tag{6}$$

It is an approximation of the original optimization problem (Eqn. 4).

Figure 2 illustrates the procedure of the relaxation. At the beginning, the optimization space $\mathcal{F}_{\mathcal{A}}(x^*)$ is non-convex; after relaxation of model g, we obtain a subspace $\mathcal{F}_{\hat{\mathcal{A}}}(x^*)$; after relaxation of model f, we obtain $\hat{\mathcal{F}}_{\hat{\mathcal{A}}}(x^*)$, which is a convex approximation of the original space. We address the optimization problem over this approximation convex space.

5.3 Duality

Training on the outer minimization in Eqn. 6 requires to calculate the gradient of model parameter θ w.r.t. the inner maximization problem: $\triangledown_\theta \max_v c^\top v = \triangledown_\theta c^\top v^*$, where v^* is the solution to the inner problem. In order to calculate the gradient, efficiently solving the inner problem, i.e., finding v^*, is desirable.

Since the inner maximization is an optimization over linear convex space $\hat{\mathcal{F}}_{\hat{\mathcal{A}}}(x^*)$, the minimum of its dual problem corresponds to the solution due to strong duality. Inspired by previous work [Wong and Kolter, 2018; Wong *et al.*, 2018], we formulate the dual problem by a feedforward network as follows:

$$\max_{v} \; c^\top v \leq \min_{\eta \geq 0} \; h_c(\eta), \tag{7}$$

where $h_c(\eta)$

$$\begin{aligned} = \;& x^\top(\hat{\nu}_1 + \eta\hat{q}_1) + \epsilon\|\hat{\nu}_1 + \eta\hat{q}_1\|_1 - \eta t \\ &+ \sum_{i=1}^{k_1-1} \nu_{i+1}^\top b_i + \eta \sum_{i=1}^{k_2-1} q_{i+1}^\top d_i \\ &- \sum_{i=2}^{k_1-1} \sum_{j\in\mathcal{I}_i} l_{i,j}[\nu_{i,j}]_+ - \eta \sum_{i=2}^{k_2-1} \sum_{j\in\mathcal{Q}_i} l'_{i,j}[q_{i,j}]_+. \end{aligned} \tag{8}$$

ν and q are defined as follows:

$$\begin{aligned} \nu_{k_1} &= -c \\ \hat{\nu}_i &= W_i^\top \nu_{i+1}, \quad \text{for } i = k_1-1,\ldots,1 \\ \nu_{i,j} &= \begin{cases} 0 & j \in \mathcal{I}_i^- \\ \hat{\nu}_{i,j} & j \in \mathcal{I}_i^+ \\ (u_{i,j}/(u_{i,j}-l_{i,j}))\hat{\nu}_{i,j} & j \in \mathcal{I}_i, \end{cases} \\ &\qquad \text{for } i = k_1-1,\ldots,2 \\ q_{k_2} &= -c \\ \hat{q}_i &= U_i^\top q_{i+1}, \quad \text{for } i = k_2-1,\ldots,1 \\ q_{i,j} &= \begin{cases} 0 & j \in \mathcal{Q}_i^- \\ \hat{q}_{i,j} & j \in \mathcal{Q}_i^+ \\ (u'_{i,j}/(u'_{i,j}-l'_{i,j}))\hat{q}_{i,j} & j \in \mathcal{Q}_i, \end{cases} \\ &\qquad \text{for } i = k_2-1,\ldots,2 \end{aligned} \tag{9}$$

In Eqn. 8 and 9, $u_{i,j}$ and $l_{i,j}$ represent the upper and lower bound of the ReLU function input for the j^{th} neuron in layer i of network $f(\cdot)$. Similarly, $u'_{i,j}$ and $l'_{i,j}$ represent the upper and lower bounds of the ReLU function input for the j^{th} neuron in layer i of network $g(\cdot)$.

For layer i in model f and g, $\mathcal{I}_i^+$ and $\mathcal{Q}_i^+$ represent the sets of neurons where the inputs are always positive, respectively ($0 \leq l \leq u$); similarly, $\mathcal{I}_i^-$ and $\mathcal{Q}_i^-$ are the sets of neurons where the inputs are always negative ($l \leq u \leq 0$); and $\mathcal{I}_i$ and $\mathcal{Q}_i$ are the sets of neurons spanning zero ($l < 0 < u$).

$\nu, \hat{\nu}, q, \hat{q}$, and η are newly introduced dual variables. We transform the original problem to an optimization problem over the new variable η.

5.4 Gradient Monotonicity and t selection

In order to solve the dual problem $\min_{\eta\geq 0} \; h_c(\eta)$, we turn to the gradient $\triangledown_\eta h_c(\eta)$, which can be written as follows:

$$\begin{aligned} & \triangledown_\eta \; h_c(\eta) \\ = \;& \epsilon \triangledown_\eta \|\hat{\nu}_1 + \eta\hat{q}_1\|_1 - t \\ & + x^\top \hat{q}_1 + \sum_{i=1}^{k_2-1} q_{i+1}^\top d_i - \sum_{i=2}^{k_2-1} \sum_{j\in\mathcal{Q}_i} l'_{i,j}[q_{i,j}]_+ \\ \overset{def}{=} \;& \epsilon\gamma(\eta) - t + C, \end{aligned} \tag{10}$$

where $\gamma(\eta) = \sum_{j=1}^{n} \text{sgn}(\hat{v}_{1,j} + \eta\hat{q}_{1,j}) \cdot \hat{q}_{1,j}$ and C is a constant independent of η and t. Note that $\gamma(\eta)$ is the analytic form of $\triangledown_\eta\|\hat{\nu}_1 + \eta\hat{q}_1\|_1$.

Theorem 1 (Property of $\gamma(\eta)$) *$\gamma(\eta)$ is a bounded non-decreasing step-function, and the transition points are $\{-\hat{\nu}_{1,j}/\hat{q}_{1,j} : 1 \leq j \leq n\}$.*

The theorem can be proved by case discussion over sign of $\hat{q}_{1,j}$. Since $\gamma(\eta)$ is the only η-dependent term in Eqn. 10, $\triangledown_\eta h_c(\eta)$ has the same property, i.e., $\triangledown_\eta h_c(\eta)$ is a bounded non-decreasing step-function. Thus, we can obtain the value range of the gradient of $h_c(\eta)$. Specifically, we have

Theorem 2 (Property of $h_c(\eta)$) *When the domain of $h_c(\eta)$ is $[0,\infty)$, let $\eta_l = 0, \eta_r = \max_j(-\hat{\nu}_{1,j}/\hat{q}_{1,j})$, the gradient value $\triangledown_\eta h_c(\eta) \in [l,r] \overset{def}{=} [\triangledown_\eta h_c(\eta)|_{\eta=\eta_l}, \triangledown_\eta h_c(\eta)|_{\eta=\eta_r}]$.*

Let us define *adversarial space constraints* to be constraints imposed by the reference model, i.e., those corresponding to $\Big(\big(g(x)\big)_{y^{\text{targ}}} - \big(g(x)\big)_{y^*} \geq t\Big)$ in Eqn. 1. We can relate the solution for $\min_{\eta\geq 0} h_c(\eta)$ to sign of l and r:

- If $l \leq r \leq 0$, when $\eta \to +\infty$, $h_c(\eta) \to -\infty$. According to the duality theorem, there is no solution to the original problem $\max_v \; c^\top v$, indicating that the space $\hat{\mathcal{F}}_{\hat{\mathcal{A}}}(x^*)$ is empty. In this situation, we regard adversarial space constraints to be *infeasible*. We need to decrease t, i.e., enlarge adversarial space $\hat{\mathcal{A}}_\epsilon(x^*)$.
- If $l < 0 < r$, a positive η solution exists. In this situation, we regard adversarial space constraints to be *tight*.
- If $0 \leq l \leq r$, the solution is $\eta = 0$. As seen from Eqn. 8, reference-model-related variables are all eliminated

from objective function $h_c(\eta)$, indicating that the adversarial space constraints are *loose*. In this situation, we would like to increase t, i.e., narrow adversarial space $\hat{\mathcal{A}}_\epsilon(x^*)$.

The ideal situation is that for all optimization objectives $h_c(\eta)$, the adversarial space constraints are tight and the solution exists, i.e., $l < 0 < r$. To guarantee so, we fix a reasonable η as the solution and inversely obtain the corresponding t.

Specifically, we set η to be the median number of $\gamma(\eta)$ transition points. Combining Theorem 1, Theorem 2, and constraint $\eta \geq 0$, we pick the median number in $\{-\hat{\nu}_{1,j}/\hat{q}_{1,j} : 1 \leq j \leq n, -\hat{\nu}_{1,j}/\hat{q}_{1,j} > 0\}$ as the solution for η, denoted as η_{med}.

η_{med} being the solution indicates $\nabla_\eta h_c(\eta)|_{\eta=\eta_{\text{med}}} = 0$, which yields $t = t_0 = \epsilon\gamma(\eta_{\text{med}}) + C$. Thus, the hyperparameter t is set to t_0 according to η_{med}.

As result, we effectively solve the inner maximization problem $\min_{\eta \geq 0} \; h_c(\eta)$ by forward passing c to the network defined in Eqn. 9 and extracting the median number using $\hat{v}_1$ and $\hat{q}_1$. After that, we use t_0 and η_{med} to directly calculate $h_c(\eta)$. Formally,

$$\max_v c^\top v \leq \min_{\eta \geq 0} h_c(\eta) = h_c(\eta)|_{\eta=\eta_{\text{med}}, t=t_0}. \tag{11}$$

5.5 Scaling

To scale up the training process, we find that Cauchy random projection estimation of ℓ_1 norm [Wong *et al.*, 2018] can be directly applied in our approach. When a vector multiplies a standard Cauchy random matrix, the median number of the result vector is an estimation of ℓ_1 norm of the vector. Using this property, we can approximately calculate $\sum_{i=2}^{k_1-1} \sum_{j \in \mathcal{I}_i} l_{i,j}[\nu_{i,j}]_+$ and $\sum_{i=2}^{k_2-1} \sum_{j \in \mathcal{Q}_i} l'_{i,j}[q_{i,j}]_+$ of Eqn. 8 in linear complexity w.r.t neuron numbers [2018].

With this calculation, our approach has the same order of complexity with [Wong *et al.*, 2018]. This calculation enables our approach to scale up to large datasets such as CIFAR10 and large DNN models such as ResNet.

5.6 Training

During the training process, we use the solution for each y^{targ} as the probability of that class. Then we use cross-entropy with softmax as the training loss to jointly minimize errors for all $y^{\text{targ}} \neq y^*$.

We adopt a manual training scheme to optimize a pair of models. Initially, $f(\cdot)$ and $g(\cdot)$ are randomly initialized independently. When the model size is small, in each epoch, we train model $f(\cdot)$ where model $g(\cdot)$ is the reference, and then train model $g(\cdot)$ where model $f(\cdot)$ is the reference. When the model size is large, we accelerate the process by distributing model $f(\cdot)$ and model $g(\cdot)$ to two GPUs. By loading the last epoch weights of the other model as the reference, two models are trained parallelly, both using the other model as the reference at the same time. This training scheme optimizes both models efficiently.

Dataset	**Model**	**# Layers**	**# Hidden Units**	**# Parameters**
MNIST	Small	4	$4,804$	$166,406$
	Large	7	$28,064$	$1,974,762$
CIFAR10	Small	4	$6,244$	$214,918$
	Large	7	$62,464$	$2,466,858$
	ResNet	11	$107,496$	$4,214,850$

Table 1: Statistics of the models used in our experiments.

6 Experiments

We evaluate Robustra on image classification tasks with two datasets: MNIST [LeCun *et al.*, 1998] and CIFAR10 [Krizhevsky and Hinton, 2009].

We use the same DNNs as used in [Wong *et al.*, 2018] for comparison. All DNNs are convolutional neural networks with multiple layers, using only ReLU activations. Table 1 shows the statistics of DNNs that we use.

For each model and dataset, we run 100 epochs on the training set. Adam optimizer is used for MNIST models, and SGD optimizer (0.9 momentum, 5×10^{-4} weight decay) is used for CIFAR10 models. The ℓ_∞ norm ϵ is initialized by 0.01, and then it linearly increases to the configured ϵ (0.1 or 0.3 for MNIST, $2/255$ or $8/255$ for CIFAR10) in the first 20 epochs. In the first 20 epochs, the learning rate is set to be 0.001; then, it decades by half every 10 epochs. The batch size is set to 50. The scaling approximation (Section 5.5) is applied for training 'Large' and 'ResNet' models, where the projection dimension is 50. Note that no hyperparameter selection or tuning is applied to guarantee the soundness of the results. All experiments are run on Geforce GTX 1080 Ti GPUs.

6.1 Provable Error Bound

We use two provable error bounds to evaluate the models trained by Robustra: *LP bound* [Wong and Kolter, 2018] and *MILP bound* [Tjeng *et al.*, 2019]. The full test set is used in our evaluation. We compare the error bounds of Robustra to the state of the art on the same model structures [Wong *et al.*, 2018; Wong and Kolter, 2018].

The results in Table 2 show that on both the MNIST dataset and CIFAR10 dataset, Robustra outperforms the state of the art in most of the settings. Specifically, on the MNIST dataset, with $\epsilon = 0.1$, Robustra significantly improves the *MILP bound* on 'Large' model from 2.74% to 2.09%; with $\epsilon = 0.3$, Robustra achieves 16.91% provable test error using 'Small' model and improves *LP bound*, from previous $43.10\%/45.66\%$ to $31.42\%/34.59\%$ for 'Small' and 'Large' models, respectively. On the CIFAR10 dataset, with $\epsilon = 2/255$, Robustra exceeds the state of the art on all models. It reduces the previous bound from 46.11% to 43.68%. With $\epsilon = 8/255$, Robustra reduces the error bound from 77.60% (on 'ResNet', *MILP bound*) to 74.87% (on 'Small', *MILP bound*).

Clean error represents the error rate of a model on a clean (i.e., unperturbed) test set. The model trained by Robustra has similar or smaller clean error rate compared to state-of-the-art models. Furthermore, we find that a model with less clean error usually has a tighter provable error bound.

Dataset	ϵ	Model	*LP bound*		*MILP bound*		Clean Err.	
			SOA	**Robustra**	**SOA**	**Robustra**	**SOA**	**Robustra**
MNIST	0.1	Small	$\mathbf{4.48\%}$	4.84%	4.38%	$\mathbf{2.55\%}$	1.26%	$\mathbf{1.01\%}$
		Large	$\mathbf{3.67\%}$	3.93%	2.74%	$\mathbf{2.09\%}$	1.08%	$\mathbf{0.83\%}$
	0.3	Small	43.10%	$\mathbf{31.42\%}$	25.79%	$\mathbf{16.91\%}$	11.40%	$\mathbf{7.37\%}$
		Large	45.66%	$\mathbf{34.59\%}$	$\mathbf{24.19\%}$	24.43%	$\mathbf{11.16\%}$	11.61%
CIFAR10	$\frac{2}{255}$	Small	52.75%	$\mathbf{51.47\%}$	50.20%	$\mathbf{47.93\%}$	38.91%	$\mathbf{36.03\%}$
		Large	46.59%	$\mathbf{43.68\%}$	-	/	31.28%	$\mathbf{28.48\%}$
		ResNet	46.11%	$\mathbf{44.43\%}$	-	/	31.72%	$\mathbf{29.51\%}$
	$\frac{8}{255}$	Small	79.25%	$\mathbf{76.29\%}$	-	$\mathbf{74.87\%}$	72.24%	$\mathbf{66.34\%}$
		Large	83.43%	$\mathbf{78.32\%}$	-	$\mathbf{77.56\%}$	80.56%	$\mathbf{71.79\%}$
		ResNet	$\mathbf{78.22\%}$	79.57%	$\mathbf{77.60\%}$	78.10%	$\mathbf{71.33\%}$	72.10%

Table 2: Provable error bound of Robustra and the state of the art (**SOA**) [Wong *et al.*, 2018; Wong and Kolter, 2018]. Two bounds are used for evaluation: *LP bound* [Wong and Kolter, 2018] and *MILP bound* [Tjeng *et al.*, 2019]. For SOA models, we use the error bounds reported in their papers. We mark a '-' on an entry if there are no reported results in the corresponding paper. For our models that are too large for *MILP bound* to be calculated in feasible time (i.e., 24 hours), we mark '/' on the entry. In addition, we compare clean example test error rate (Clean Err.) of SOA models and our models.

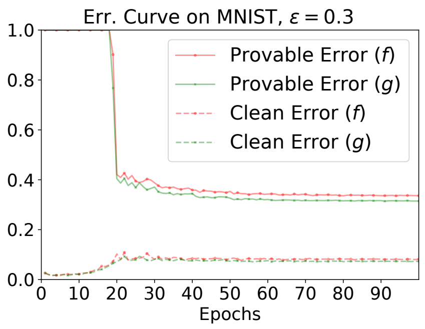


Figure 3: Provable robust error bound and clean error curves of each epoch, evaluated on the MNIST test set, with $\epsilon = 0.3$. Provable robust error bound represents *LP bound*. Clean error represents the error rate on clean examples. $f(\cdot)$ and $g(\cdot)$ stand for the two mutually trained models.

6.2 Training Progress

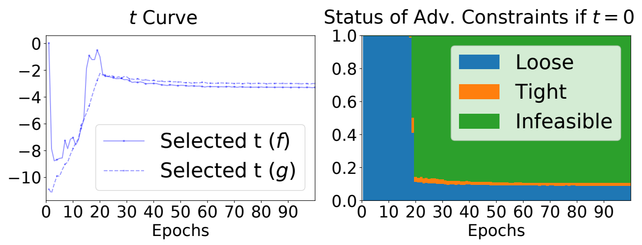
The training curves are shown in Figure 3. Provable error is measured by *LP bound*. Clean error is measured by error rate on clean (i.e., unperturbed) examples. Both are evaluated on the test set.

In the first 20 epochs, ϵ used for training is linearly increased to desired ϵ (0.3). The provable robust error keeps being 100% for these first 20 epochs, and then a significant drop occurs as the training ϵ reaches 0.3. The result reveals that the model is not robust to larger perturbations unless it is trained at that perturbation level. During these first 20 epochs, the clean error increases while the provable robust error bound decreases. Then both provable errors and clean errors decrease.

At each epoch, both models are trained mutually: we first regard $g(\cdot)$ as the reference model and train the model $f(\cdot)$; then, we set the trained $f(\cdot)$ as the reference model and train the model $g(\cdot)$. Thus, at each epoch, the test error of $g(\cdot)$ is slightly smaller than the test error of $f(\cdot)$, as shown in Figure 3.

6.3 t Selection Effectiveness

In our approach, we select some t to solve the inner maximization problem, as discussed in Section 5.4. Figure 4a plots



(a) (b)

Figure 4: On MNIST with $\epsilon = 0.3$: (a) curve of selected t; and (b) tightness status of adversarial space constraints of each epoch when $t = 0$.

the curve of selected t. Larger t corresponds tighter reference space, as shown in Eqn. 1. In the first 20 epochs, t is small and sharply increases to around -3. This result indicates that our approach dynamically tightens these constraints along with increasing training ϵ. After 20 epochs, as the reference model is trained more robust, its adversarial space becomes smaller. The approach gradually decreases t to loosen the constraints.

In Figure 4b, on MNIST with $\epsilon = 0.3$, we study the solution status on test dataset if fixing $t = 0$. In the first 20 epochs, non-robust models make almost all constraints loose. After 20 epochs, the robust models make almost all constraints infeasible. Thus, when $t = 0$, we are not able to effectively solve the problem. Instead, we need to dynamically select t as we propose in Section 5.4.

7 Conclusion

We have presented Robustra, an approach for training provable robust neural networks. In particular, we formalize the training procedure as a min-max game over the adversarial space of the reference model, and effectively solve the min-max game. The experimental results have shown that Robustra significantly outperforms the state-of-the-art approach.

Acknowledgments

This work was supported in part by NSF under grants no. CNS-1513939, CNS-1564274, CCF-1816615.

References

[Athalye *et al.*, 2018] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *International Conference on Machine Learning*, pages 274–283, 2018.

[Carlini and Wagner, 2017a] Nicholas Carlini and David Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In *ACM Workshop on Artificial Intelligence and Security*, pages 3–14, 2017.

[Carlini and Wagner, 2017b] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *IEEE Symposium on Security and Privacy*, pages 39–57, 2017.

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

[Fazlyab *et al.*, 2019] Mahyar Fazlyab, Manfred Morari, and George J. Pappas. Safety verification and robustness analysis of neural networks via quadratic constraints and semidefinite programming. *arXiv preprint arXiv:1903.01287*, 2019.

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

[Katz *et al.*, 2017] Guy Katz, Clark Barrett, David Dill, Kyle Julian, and Mykel Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pages 97–117, 2017.

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

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

[Madry *et al.*, 2018] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018.

[Mirman *et al.*, 2018] Matthew Mirman, Timon Gehr, and Martin Vechev. Differentiable abstract interpretation for provably robust neural networks. In *International Conference on Machine Learning*, pages 3578–3586, 2018.

[Papernot *et al.*, 2016] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *IEEE Symposium on Security and Privacy*, pages 582–597, 2016.

[Raghunathan *et al.*, 2018] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Certified defenses against adversarial examples. In *International Conference on Learning Representations*, 2018.

[Salman *et al.*, 2019] Hadi Salman, Greg Yang, Huan Zhang, Cho-Jui Hsieh, and Pengchuan Zhang. A convex relaxation barrier to tight robust verification of neural networks. *arXiv preprint arXiv:1902.08722*, 2019.

[Samangouei *et al.*, 2018] Pouya Samangouei, Maya Kabkab, and Rama Chellappa. Defense-GAN: Protecting classifiers against adversarial attacks using generative models. In *International Conference on Learning Representations*, 2018.

[Singh *et al.*, 2018] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. Fast and effective robustness certification. In *Advances in Neural Information Processing Systems*, pages 10802–10813, 2018.

[Singh *et al.*, 2019] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. In *ACM SIGPLAN Symposium on Principles of Programming Languages*, volume 3, page 41, 2019.

[Szegedy *et al.*, 2014] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014.

[Tjeng *et al.*, 2019] Vincent Tjeng, Kai Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. In *International Conference on Learning Representations*, 2019.

[Weng *et al.*, 2018] Tsui-Wei Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Duane Boning, Inderjit S Dhillon, and Luca Daniel. Towards fast computation of certified robustness for ReLU networks. In *International Conference on Machine Learning*, pages 5273–5282, 2018.

[Wong and Kolter, 2018] Eric Wong and J. Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International Conference on Machine Learning*, pages 5283–5292, 2018.

[Wong *et al.*, 2018] Eric Wong, Frank Schmidt, Jan Hendrik Metzen, and J. Zico Kolter. Scaling provable adversarial defenses. In *Advances in Neural Information Processing Systems*, pages 8400–8409, 2018.

[Xiao *et al.*, 2019] Kai Xiao, Vincent Tjeng, Nur Muhammad Shafiullah, and Aleksander Madry. Training for faster adversarial robustness verification via inducing ReLU stability. In *International Conference on Learning Representations*, 2019.