Evaluating Robustness of Neural Networks

by

Tsui-Wei (Lily) Weng

B.S., National Taiwan University (2011)
S.M., National Taiwan University (2013)

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

Doctor of Philosophy in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2020

Author .....
Department of Electrical Engineering and Computer Science
August 28, 2020

Certified by .....
Luca Daniel
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

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

Evaluating Robustness of Neural Networks

by

Tsui-Wei (Lily) Weng

Submitted to the Department of Electrical Engineering and Computer Science
on August 28, 2020, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Electrical Engineering and Computer Science

Abstract

The robustness of neural networks to adversarial examples has received great attention due to security implications. Despite various attack approaches to crafting visually imperceptible adversarial examples, little has been developed towards a comprehensive metric of robustness. This thesis is dedicated to developing several robustness quantification frameworks for deep neural networks against both adversarial and non-adversarial input perturbations, including the first robustness score CLEVER, efficient certification algorithms Fast-Lin, CROWN, CNN-Cert, and probabilistic robustness verification algorithm PROVEN. Our proposed approaches are computationally efficient and provide good quality of robustness estimates and certificates as demonstrated by extensive experiments on MNIST, CIFAR and ImageNet.

Thesis Supervisor: Luca Daniel
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

During my study at MIT, I worked with many amazing people who I am very thankful and grateful. First of all, I am deeply indebted to my PhD advisor Prof. Luca Daniel for his endless patience, encouragement and mentoring throughout the years of my PhD. His continued advice, guidance and support helped me to navigate the most intricate challenges during my PhD journey. I greatly appreciate his understanding when I decided to change my research area from uncertainty quantification and photonics to machine learning; and I truly appreciate his support to let me explore different research topics and being very patient to let me figure out what my strength and potentials are. Luca also supported me to attend many conferences and workshops, which has greatly enriched my research and cultural experience. Luca is a great professor and also a great mentor.

I am also very grateful for my thesis committee members, Prof. Alexandre Megretski and Prof. Tommi Jaakkola, who have provided extensive help and guidance. I learnt many new things from every meeting with Sasha, especially the thought process, and I appreciate his patience in discussing with me on many research ideas and being very supportive to my research direction. I am also very thankful to Prof. Jaakkola for his time to discuss interesting research ideas and feedback for my research. I have been greatly inspired by his insights and learned to think out-of-box and try to approach a problem in many different view points.

I would also like to sincerely thank my research collaborators (and mentors). Dr. Zheng Zhang introduced me into the field of Uncertainty Quantification and has always been a caring mentor not only in the lab but also in my life outside of research. Dr. Pin-Yu Chen has been a long-term collaborator ever since our first MIT-IBM proposal! Dr. Sijia Liu has been a great collaborator at MIT-IBM lab and has also been very helpful and generous to share with me many his valuable experience. Dr. Naoki Abe was my manager when I interned at IBM Research and I have learned a lot from him. I appreciate his guidance and kindness to believe potentials in me even I just switched my research area at the time.

The time spent with my smart and nice students is truly amazing. I have to thank Luca for giving me the opportunities to work with and supervise many MIT undergraduate students, master and PhD students over the last two years. It was a very amazing rewarding experience and I am very lucky to work with so many brilliant and talented fellow students: Akhilan Boopathy, Shreyan Jain, Jeet Mohapatra, Irene Ko, Zhaoyang Lyu, Tuomas Oikarinen, Wonjune Kang, Alex Gu and Victor Rong. I also want to thank my lab mates including Dr. Niloofar Farnoosh, Zohaib Mahmood, Yu-Chung Hsiao, Richard Zhang, Jose Cruz Serralles, for being a great friend in my journey! I also want to thank my friends for supporting me through many ups and downs: Jiasi Shen, Debbie Yu, Tien-Ru Yang, Sumaiya Nazeen, Yen-ling Guo, Annie Chen, Hsin-Jung Yang, Charith Mendis, Omar Labbin, Ann Collin, Guan Yue, Yang Yang, Cheng Peng, Kexin Yi, Hang Zhao, Xijia Zheng, Zheng Li, Zhan Su, Penny Lee, Claire Wu, Yi-jin Wu, Hwang-Fu.

I also greatly appreciate NSF NEEDS programs and MIT-IBM Watson AI lab for supporting my research throughout my PhD journey. The revolutionary results of this work thanks to MIT-IBM Lab's funding support.

Finally, I would like to thank my family (Mom, Dad, aunts, cousins, nephews) and my fiancee Mickey for their endless support and love. I owe this thesis to them.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Deep neural networks (DNNs) have recently demonstrated superior performance in many machine learning tasks related to computer vision, natural language processing and speech, such as image classification [53], object detection [92], scene parsing [107], instance segmentation [67], machine translation [4], text classification [57], speech recognition [38], etc. In addition to the aforementioned classical machine learning and artificial intelligence domains, recently deep neural networks have also been widely deployed in the field of medicine and biology [18], security and defense [2], finance [40], and robotics [81], for the tasks including but not limited to cancer cell detection, diabetic grading, drug discovery, face detection, video surveillance, satellite imagery, fraud detection, trading, financial services, robot manipulation, motion planning and autonomous driving.

While researchers are very excited about the remarkable success and capabilities of DNNs in a large number of applications, a natural question arises:

" *are deep neural networks safe and trustworthy*? "

Unfortunately, the answer is No,

deep neural networks are not as safe and trustworthy as we expect.

As an example, DNNs are vulnerable to adversarial attacks [94], where a small and often human-imperceptible perturbation on test images can surprisingly fool the state-of-the-art image classifiers. This first discovery of adversarial examples has drawn serious attentions and raised concerns in the machine learning community.

Unfortunately, the existence of adversarial examples for image classifiers is not a single occurrence. It has been demonstrated in many different DNNs applications, including for instance object detection [104], image captioning [14], speech recognition [20], malware detection [97] and reading comprehension [46]. Moreover, beyond the white-box attack setting where the target model is entirely transparent, visually imperceptible adversarial perturbations can also be generated in the black-box setting by only using the prediction results of the target model [75, 69, 16, 10]. In addition, real-life adversarial examples have been made possible by crafting physical perturbations [55, 29, 3]. As DNNs are becoming a core technique deployed in a wide range of applications, the vulnerability to adversarial examples calls into question safety-critical applications and services deployed by neural networks, including autonomous driving systems and malware detection protocols, among others.

While there is a relentless arm race in crafting adversarial examples with stronger attacking performance, a more alarming observation is that DNNs are also susceptible to random noises – [7] showed that LeNet and AlexNet malfunctioned with additive Gaussian noises, [32] showed that random perturbations can fool VGG networks; [43] showed that the Google Cloud Vision API are vulnerable to random Gaussian noises. The above results suggested that current state-of-the-art DNNs may be over-sensitive to small input variations, regardless if the variations source are adversarial or natural. Indeed, DNNs are powerful but we need more than just superior performance. As a first step toward building safe and trustworthy deep neural networks, our goal is to assess and verify if the AI and ML system involving DNNs produce consistent answers upon perturbations.

In this thesis, we investigate the *robustness* of deep neural networks against both adversarial and non-adversarial input perturbations, hoping to propose a useful evaluation metric to efficiently assess the robustness of neural networks.

1.2 How to evaluate robustness of DNNs?

Up to the end of 2017, the machine learning community mainly assessed the robustness of neural networks via attacks algorithms. Their goal is to develop strong adversarial attacks which can craft visually imperceptible adversarial distortions with high success rate. Classical adversarial attacks in chronological order includes Fast Gradient Sign Method (FGSM) [94, 37], DeepFool attack [73], Carlini Wagner attack [13] and Projected Gradient Descent (PGD) attack [71]. The central idea of the above attack algorithms is to compute or estimate the gradient of loss functions or classifier functions to create adversarial examples. As time evolves, stronger attacks are devised to craft stronger adversarial examples with smaller adversarial distortions and higher success rate.

However, we argue that using adversarial distortions crafted by attack algorithms as a way to evaluate robustness has some limitations. The reason is that though it is easy to find *one* adversarial perturbation through attack algorithms, it is hard to find *smallest* adversarial perturbation, which is the ultimate goal of adversaries. As it is difficult to know if strong attacks could exist, it is better to have a robustness metric that is not entangled with the attack algorithms so that the metric is not limited by the attack capabilities. Hence, the focus of this thesis is to develop a metric that is independent of the attack algorithms, which is also called an *attack-agnostic* robustness metric. To achieve our goal, there are three fundamental questions to be answered:

1. How do we properly define an *attack-agnostic* robustness metric?
2. Can we develop tools and algorithms that compute the proposed metric efficiently?
3. Can we provide any guarantees on our tools and algorithms?

In the next section, we first define an metric to quantify robustness independent of attack algorithms. The rest of the thesis is dedicated to answer the 2nd and 3rd question with constructive results.

1.3 Attack-agnostic robustness metrics

Given a neural network classifier function f and let $\mathbf{x_0}$ be a natural example (e.g. a given arbitrary test image). An attacker wants to find a perturbation δ such that f gives wrong prediction results on the perturbed image $\mathbf{x_0} + \delta$. In the literature, a standard way to quantify the magnitude of the perturbation is through ℓ_p norm, $\|\cdot\|_p$, which is defined as

$$\|\delta\|_p = (\sum_{i=1}^{n} |\delta_i|^p)^{\frac{1}{p}},$$

where δ is a n-dimensional vector. Popular choices of p includes $p = 1, 2, \infty$, which corresponds to the well-known Manhattan distance, Euclidean distance and Maximum component (e.g. pixel) change. As described in the previous section, an attacker's goal is to find a perturbation δ such that $f(x + \delta)$ gives the wrong prediction, plotted as red points in Figure 1-1. The attacker's ultimate goal is to find the smallest possible δ, which is known as *minimum adversarial perturbation*; however, at least in the case when f is a ReLU network, it has been proved that this is an NP-complete problem [48].

Thus, we propose an attack-agnostic robustness metric r as follows:

> The prediction results are the same (consistent) in the local neighborhood of $\mathbf{x_0}$ for any perturbations $\|\delta\|_p \leq r$.

This local neighborhood, i.e. the ℓ_p-ball region centered at $\mathbf{x_0}$, is called a *certified robustness region*, the green region depicted in Figure 1-1. Ideally, we want to compute the *largest* r so that the output of neural network $f(\mathbf{x_0} + \delta)$ produces the same prediction as $f(\mathbf{x_0}), \forall \|\delta\|_p \leq r$. Unfortunately, computing the maximum r is equivalent to computing minimum the adversarial distortion and has been shown to be an NP-complete problem [48, 91] at least in the case of ReLU networks. Hence, formal verification methods such as Reluplex [48] and Planet [27] are computationally demanding and cannot scale to large realistic networks and practical applications. As an example, Reluplex, a formal verification technique based on Satisfactory Modulo Theory (SMT), requires 3 hours for one example $\mathbf{x_0}$ on a very small neural network

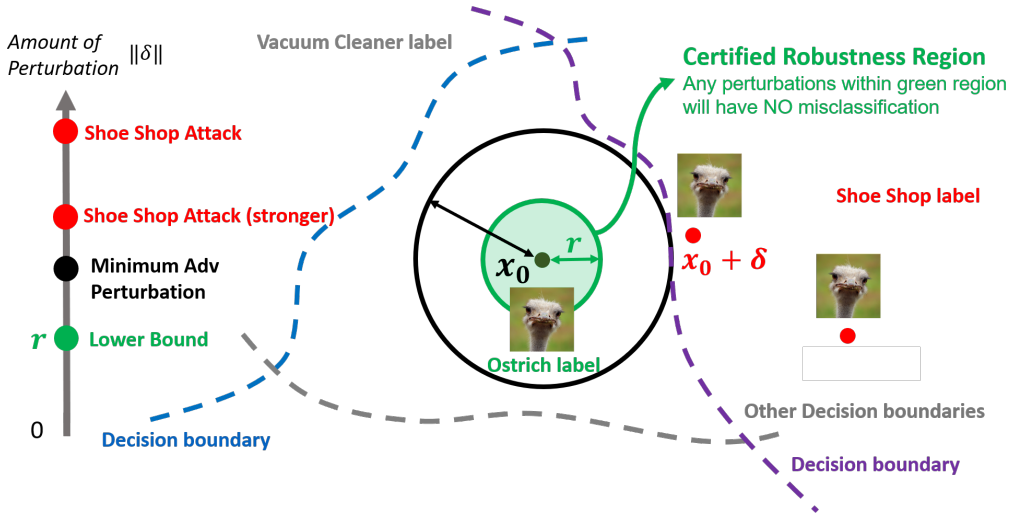


Figure 1-1: Measuring robustness of DNNs.

with only 5 inputs, 5 outputs and a total of 300 neurons (i.e. the input $\mathbf{x_0}$ has dimension equals 5, and model complexity with 300 neurons). In contrast, standard image datasets such as MNIST [58], CIFAR [54] and ImageNet [23] have much higher input dimensions: 28×28 pixels, $32 \times 32 \times 3$ pixels, $256 \times 256 \times 3$ pixels, respectively. Moreover, state-of-the-art DNNs have much higher model complexity, e.g. AlexNet and VGG-16 have more than 650,000 and 13,000,000 neurons respectively.

Therefore, this thesis is devoted to developing *efficient* and *scalable* algorithms to compute attack-agnostic robustness for DNNs. Importantly, we want our algorithms to satisfy the crucial properties:

- Scalable to the standard datasets (e.g. MNIST, CIFAR, ImageNet) and to the state-of-the-art DNNs (e.g. AlexNet, VGG-16);
- Equipped with some provable guarantees on the adversarial perturbation (worst-case guarantees) and non-adversarial perturbations (probabilistic guarantees).

1.4 Thesis Contributions and Organization

1.4.1 Contributions of this thesis

This thesis focuses on the development of efficient local robustness quantification algorithms to accelerate robustness assessment for DNNs. The novel contributions

include three parts.

In the first part, we propose a novel *efficient* adversarial local robustness algorithm that is scalable to state-of-the-art DNNs and practical datasets such as ImageNet.

- **Contribution 1.** In Chapter 3, we propose a novel and generic robustness score that we called CLEVER, which is short for **C**ross **L**ipschitz **E**xtreme **V**alue for n**E**twork **R**obustness. Compared to the existing robustness evaluation approaches, our metric has the following advantages: (i) attack-agnostic; (ii) applicable to any neural network classifier; (iii) comes with strong theoretical guarantees; and (iv) is computationally feasible for large neural networks. Our extensive experiments show that the CLEVER score is a good practical indicator for the level of local robustness of a wide range of natural and defended networks.

The second part of this thesis develops a generic verification framework to quantify neural network robustness with some *provable guarantees*.

- **Contribution 2.** In Chapter 4, we present a general framework CROWN to efficiently compute a certified lower bound of minimum distortion in neural networks for any given data point $\mathbf{x_0}$. CROWN is a generalized version of our previous robustness verification algorithm Fast-Lin [101] based on linear parallel bounding techniques. Specifically, CROWN features adaptive linear (quadratic) upper and lower bounds and works on neural networks with *general* activation functions that are not necessarily piece-wise linear. Experiments show that (1) CROWN outperforms state-of-the-art baselines on ReLU networks and (2) CROWN can efficiently certify non-trivial lower bounds for large networks with over 10K neurons and with different activation functions.

The third part of this thesis focuses on quantify the local robustness neural network robustness with provable *probabilistic* guarantees which is applicable to *non-adversarial* input perturbations.

- **Contribution 3.** In Chapter 5, we propose a novel probabilistic framework PROVEN to quantify the local robustness of neural networks and derived theoretical bounds on the robustness certificate with statistical guarantees. PROVEN is a

general tool that can build on top of existing state-of-the-art neural network local robustness level certification algorithms including Fast-Lin [101], CROWN [106], and CNN-Cert [8] and hence can be readily applied to fully-connected and convolutional neural networks with different activation functions. Experimental results on large neural networks demonstrated significant benefits of PROVEN over the standard worst-case analysis results.

1.4.2 Thesis Outline

This thesis is organized as follows:

- In Chapter 2, we give a brief introduction to deep learning and neural networks basics, as well as notations used in this thesis.
- Chapter 3 to Chapter 5 presents the details of our three novel contributions, including how to efficiently compute the attack-agnostic local robustness metric for DNNs and how to provide some guarantees for the proposed metric.
- Chapter 6 overviews a few of our extensions and follow-up works based on our proposed robustness verification framework.
- Finally, Chapter 7 summarizes the results of this thesis and discusses promising future directions in this field.

Chapter 2

Backgrounds

In this Chapter, we first give a brief introduction on the backgrounds of neural networks and image classification tasks in Section 2.1 and 2.2. We then summarize the notations that will be used frequently in the following Chapters in Section 2.3.

2.1 Neural Networks

The most basic form of a neural network model is *feed-forward fully-connected* neural network, also known as *multi-layer perceptron* (MLP), where every neuron in one layer are connected to every neuron in the next layer in a feed-forward manner, as shown in Figure 2-1. The mathematical form of an m-layer MLP with $m-1$ hidden layers, $f(\mathbf{x})$, is defined as follows.

Input and Model parameters. Let $\mathbf{x} \in \mathbb{R}^{n_0}$ be the input vector and let the number of neurons in each layer be $n_k, \forall k \in [m]$. We use $[n]$ to denote set $\{1, 2, \cdots, n\}$. The model parameters of a MLP consist of weight matrices $\mathbf{W}$ and biases $\mathbf{b}$, which are trainable and to be determined during the training processes (will be discussed more in the next section). Let the parameters of the k-th layer be $\mathbf{W}^{(k)}$ and $\mathbf{b}^{(k)}$ with dimension $n_k \times n_{k-1}$ and n_k, respectively.

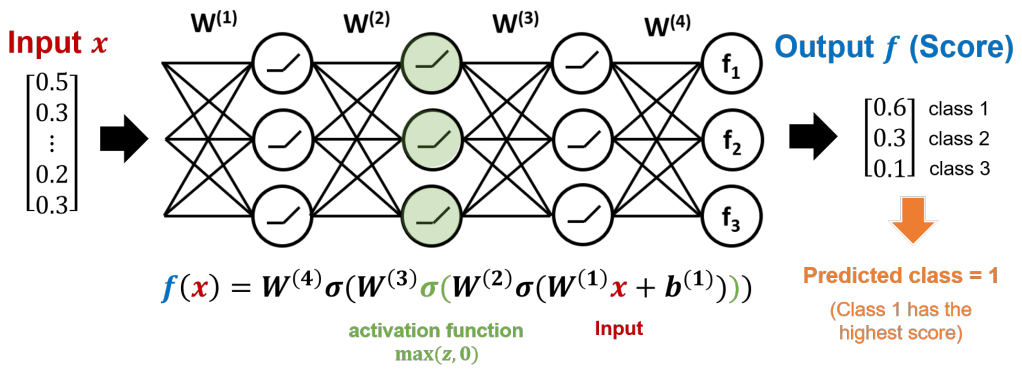


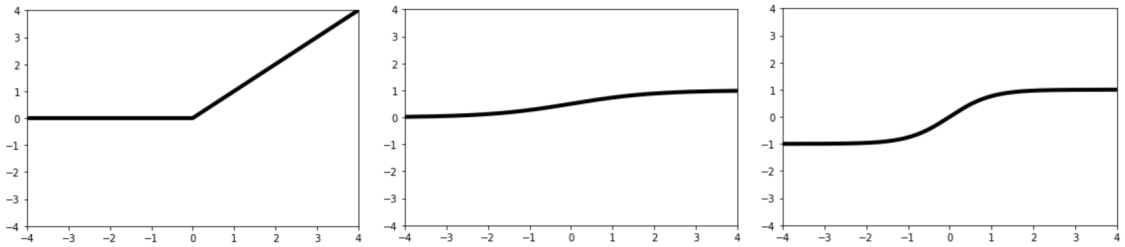
Figure 2-1: A 4-layer feed-forward neural network with 3 classes.

Math form and Activation functions. Let $\Phi^k : \mathbb{R}^{n_0} \to \mathbb{R}^{n_k}$ be the operator mapping from input layer to layer k and $\sigma(\mathbf{y})$ be the coordinate-wise activation function; for each $k \in [m-1]$, the relation between layer $k-1$ and layer k can be written as $\Phi^k(\mathbf{x}) = \sigma(\mathbf{W}^{(k)}\Phi^{k-1}(\mathbf{x}) + \mathbf{b}^{(k)})$, where $\mathbf{W}^{(k)} \in \mathbb{R}^{n_k \times n_{k-1}}, \mathbf{b}^{(k)} \in \mathbb{R}^{n_k}$. For the input layer and the output layer, we have $\Phi^0(\mathbf{x}) = \mathbf{x}$ and $\Phi^m(\mathbf{x}) = \mathbf{W}^{(m)}\Phi^{m-1}(\mathbf{x}) + \mathbf{b}^{(m)}$. The output of the neural network is $f(\mathbf{x}) = \Phi^m(\mathbf{x})$, which is a vector of length n_m, and the j-th output is its j-th coordinate, denoted as $f_j(\mathbf{x}) = [\Phi^m(\mathbf{x})]_j$. Popular choices of activation functions include:

- ReLU: $\sigma(\mathbf{y}) = \max(\mathbf{y}, \mathbf{0})$
- hyperbolic tangent: $\sigma(\mathbf{y}) = \tanh(\mathbf{y})$
- sigmoid: $\sigma(\mathbf{y}) = 1/(1 + e^{-\mathbf{y}})$
- inverse tangent: $\sigma(\mathbf{y}) = \tan^{-1}(\mathbf{y})$

which are element-wise operation on the input vector $\mathbf{y}$. The activation functions are plotted in Figure 2-2.

Advanced neural network architectures. The feed-forward fully-connected neural networks can be viewed as cascading the *fully-connected* layers with activation functions in sequence. In deep learning, especially for computer vision tasks, there are some popular building blocks that help to further increase the power of deep neural networks, including convolutional layers, max-pooling layers, batch-normalization



(a) ReLU (b) Sigmoid (c) Tanh

Figure 2-2: Commonly-used activation functions in neural networks.

layers, as well as residual blocks. The mathematical form of the input-output relations for these building blocks are summarized below, where Φ^{r-1} is the input and Φ^r is the output:

- Fully-connected layer:

$$\Phi^r = \sigma(\mathbf{W}^{(r)}\Phi^{r-1} + \mathbf{b}^{(r)});$$

- Convolutional layer:

$$\Phi^r = \sigma(\mathbf{W}^{(r)} * \Phi^{r-1} + \mathbf{b}^{(r)}),$$

  where $*$ is the convolutional operator and $\mathbf{W}^{(r)}, \mathbf{b}^{(r)}$ and Φ^r are three dimensional tensors;

- Max pooling layer:

$$\Phi^r_n = \max_{S_n} \Phi^{r-1}_{S_n},$$

  where S_n denotes the pooled input index set associated with the n-th output;

- Residual blocks:

$$\Phi^{r-1} = \mathbf{W}^{r-1} * \Phi^{r-2} + \mathbf{b}^{r-1},$$
$$\Phi^r = \mathbf{W}^r * \sigma(\Phi^{r-1}) + \mathbf{b}^r + \Phi^{r-2},$$

  note that in this building block, Φ^{r-2} is the input;

- Batch-normalization layer:

$$\Phi^r = \gamma_{\text{bn}} \frac{\Phi^{r-1} - \mu_{\text{bn}}}{\sqrt{\sigma_{\text{bn}}^2 + \epsilon_{\text{bn}}}} + \beta_{\text{bn}},$$

  where γ_{bn}, β_{bn} are learned training parameters and μ_{bn}, σ_{bn}^2 are the running average of the batch mean and variance of training data during training, and ϵ_{bn} is a small positive number to avoid denominator being zero.

2.2 Image Classification tasks

An image classification task is a supervised-learning task in computer vision, where a machine learning model $f(\mathbf{x})$ is used to associate an unseen image $\mathbf{x}$ with y, one out of a set of pre-defined categories such as animals, automobiles, plants, etc. There are many options for a machine learning model, for instance, $f(\mathbf{x})$ could be a decision tree, k-nearest neighbor model, support vector machine and neural network. This thesis is primarily focused on the neural network models, as recent advancement in deep learning has demonstrated unprecedented success in the image classification task, which can achieve classification accuracy close to (or even better than) human perceptions. In general, a classification task is not limited to images-based input – for example, the input could be text or audio, and the tasks could be document classification, sentiment analysis, speech classifications, just to name a few.

A standard machine learning workflow consists of 3 stages: (i) Gathering data, (ii) Training a model and (iii) Testing the learned model.

- **(i) Gathering data.** In supervised-learning task, we need to gather the data pairs $\{(\mathbf{x}_i, y_i)\}$, where $\mathbf{x}_i$ is an input feature vector (e.g. the RGB values of an image), and y_i is its corresponding class label (usually represented as an integer number). Once we gather all the available data pairs for the task, all the data is divided into *training set*, *validation set* and *test set*, often in the percentages of $70\%, 15\%, 15\%$ respectively. The training set is used to to fit the model and the validation set can then be used to select (tune) some hyper-parameters of the model. The test set, known as hold-out set, is never exposed during training process and is solely used in the testing phase in order to fairly assess the learned model performance (i.e. generalization).

- **(ii) Training a model.** To train a model, one needs to first decide which machine learning model to use. As discussed earlier, there are many different types of machine learning models spanning from a decision tree, support vector machines to neural networks, and each is useful in different applications. In this thesis, we focus on neural network model f due to its superior performance in

many standard machine learning tasks. For a basic feed-forward neural network, denoted as $f_\theta(\mathbf{x})$, described in the last section, the model parameters θ are the weight matrices $\mathbf{W}$ and biases vectors $\mathbf{b}$. The purpose of the training process is to find values $\hat{\theta}$ such that the model fits the training data well (but does not result in over-fitting) while able to generalize to unseen test data. To assess the model performance, a standard procedure is to define a loss function $\mathcal{L}(\theta; \mathbf{x}_i, y_i)$, which characterizes the prediction error of the classifier f_θ and the true label y_i on the data point $\mathbf{x}_i$. Some commonly used losses are the Mean-Square-Error, also known as L2 loss:

$$\mathcal{L}(\theta; \mathbf{x}_i, y_i) = \|f_\theta(x_i) - y_{i,\text{one-hot}}\|_2^2,$$

and cross entropy loss:

$$\mathcal{L}(\theta; \mathbf{x}_i, y_i) = -\sum_{c=1}^{M} \left[f_\theta(x_i)\right]_c \cdot \left[\log y_{i,\text{one-hot}}\right]_c,$$

where M is the total number of classes and $y_{i,\text{one-hot}} \in [0,1]^M$ is the one-hot vector of label y_i (i.e. only $[y_{i,\text{one-hot}}]_{y_i} = 1$ while other elements in the vector is 0; y_i is the class label of training data x_i and is a positive integer.) [1]. Once the loss function is decided, one can start the training process, which is an optimization process with the goal of *finding the best θ that can minimize the loss function $\mathcal{L}$ over the training data.* Some default choices of optimization solvers for Neural networks training includes Stochastic Gradient Descent (SGD) [9], accelerated Gradient Descent [86], RMSProp [95], AdaGrad [99], Adam [50] etc, some of which are useful in helping escaping from possible saddle points. Since a neural network is a highly non-convex function, the convergence guarantees of the

[1] Note that here we assume the output of $f_\theta(x_i)$ is already normalized to probability, i.e $\sum_{c=1}^{M} \left[f_\theta(x_i)\right]_c = 1$. This can be easily achieved by adding a soft-max layer before the output of the neural network. Nevertheless, for all the analysis in this thesis, we could remove the last layer of a neural network (i.e. the soft-max layer) because the soft-max function is a monotonically increasing function, and our analysis only requires relative difference between each output, e.g. $f_c(\mathbf{x}) - f_j(\mathbf{x})$, $j \neq c$.

training algorithms are still an active area of research.

- **(iii) Testing the learned model.** Once a model is trained (learned), the next immediate task is to test the generalization performance of the learned model on the un-seen test data. As illustrated in Figure 2-1, a trained classifier can be used for prediction on an unseen test sample $\mathbf{x}_{\text{test}}$ and the rule is as follows:

$$\text{predicted label } \hat{c} = \underset{i}{\text{argmax}}\, f_i(\mathbf{x}_{\text{test}}).$$

A standard metric is to use zero-one loss on the testing points:

$$\mathcal{L}(\theta; \mathbf{x}_{\text{test}}, y_{\text{test}}) = \mathbb{1}_{\hat{c} = y_{\text{test}}},$$

and the mean of the total loss on the full test set is equivalent to computing average test accuracy. Lastly, we note that in the rest of thesis, we often omit the parameter θ in f_θ because the adversarial attacks of concerns happens at testing time, hence θ is already frozen before the attack.

2.3 Notations in this thesis

In this section, we summarized the important symbols used in this thesis in the following Table 2.1.

Table 2.1: Table of Notation

Notation	Definition
$\|\cdot\|_p$	ℓ_p norm
n_0	dimensionality of the input vector
K	number of output classes
$f : \mathbb{R}^{n_0} \to \mathbb{R}^K$	neural network classifier
$c = \text{argmin}_i\, f_i(\mathbf{x_0})$	predicted class of input $\mathbf{x_0}$
$\mathbf{x_0} \in \mathbb{R}^{n_0}$	original input vector
$\mathbf{x}_a \in \mathbb{R}^{n_0}$	adversarial example
$\boldsymbol{\delta} \in \mathbb{R}^{n_0}$	input perturbation
$r_{p,\min}$	minimum ℓ_p perturbation of $\mathbf{x_0}$
$r_{p,U}$	upper bound of minimum perturbation $r_{p,\min}$
$r_{p,L}$	lower bound of minimum perturbation $r_{p,\min}$
$B_p(\mathbf{x_0}, R)$	hyper-ball with center $\mathbf{x_0}$ and radius R
L_q^j	Lipschitz constant
L_{q,x_0}^j	local Lipschitz constant
$\sigma(\mathbf{y})$	activation function
CDF	cumulative distribution function
PDF	probability density function
F_X	CDF of a random variable X
f_X	PDF of a random variable X
$g_t(\mathbf{x}) := f_c(\mathbf{x}) - f_t(\mathbf{x})$	margin function at $\mathbf{x}$ for class $t \neq c$
$\mathbb{P}[g_t(X) > a]$	probability that $g_t(X)$ is greater than $a \in \mathbb{R}$
$g_t^L(\mathbf{x}) : \mathbb{R}^{n_0} \to \mathbb{R}$	linear lower bound of $g_t(\mathbf{x})$
$g_t^U(\mathbf{x}) : \mathbb{R}^{n_0} \to \mathbb{R}$	linear upper bound of $g_t(\mathbf{x})$
γ_L	theoretical lower bound of $\mathbb{P}[g_t(X) > a]$
γ_U	theoretical upper bound of $\mathbb{P}[g_t(X) > a]$

Chapter 3

Robustness Quantification via Local Lipschitz Continuity

3.1 Introduction

Motivated by the evaluation criterion for assessing the quality of text and image generation that is completely independent of the underlying generative processes, such as the BLEU score for texts [77] and the INCEPTION score for images [87], we aim to propose a comprehensive and attack-agnostic local robustness metric for neural networks. Stemming from a perturbation analysis of an arbitrary neural network classifier, we derive a universal lower bound on the minimal perturbation required to craft an adversarial example from an original one, where the lower bound applies to any attack algorithm and any ℓ_p norm for $p \geq 1$. We show that this lower bound associates with the maximum norm of the local gradients with respect to the original example, and therefore local robustness quantification becomes a local Lipschitz constant estimation problem. To efficiently and reliably estimate the local Lipschitz constant, we propose to use *extreme value theory* [22] for local robustness quantification. In this context, the extreme value corresponds to the local Lipschitz constant of our interest, which can be inferred by a set of independently and identically sampled local gradients. With the aid of the extreme value theory, we propose a local robustness metric called CLEVER, which is short for **C**ross **L**ipschitz **E**xtreme **V**alue

for n**E**twork **R**obustness.

3.1.1 Contributions

We highlight the main contributions of this chapter as follows:

1. We propose a novel local robustness score called CLEVER, which is short for **C**ross **L**ipschitz **E**xtreme **V**alue for n**E**twork **R**obustness. CLEVER is the first local robustness score that is attack-independent and can be applied to any arbitrary neural network classifier and scales to large networks for ImageNet.

2. CLEVER score is well supported by our theoretical analysis on formal robustness guarantees and the use of extreme value theory. Our robustness analysis extends the results in [41] from continuously differentiable functions to a special class of non-differentiable functions – neural networks with ReLU activations.

3. We corroborate the effectiveness of CLEVER by conducting experiments on state-of-the-art models for ImageNet, including ResNet [39], Inception-v3 [93] and MobileNet [44]. We also use CLEVER to investigate defended networks against adversarial examples, including the use of defensive distillation [76] and bounded ReLU [105]. Experimental results show that our CLEVER score well aligns with the attack-specific robustness indicated by the ℓ_2 and ℓ_∞ perturbations of adversarial examples.

3.1.2 Related Literature

[94] computed global Lipschitz constant for each layer and used their product to explain the robustness issue in neural networks, however the global Lipschitz constant often gives a very loose bound. [41] gave a robustness lower bound via mean value theorem and derived a closed-form bound for a multi-layer perceptron (MLP) with a single hidden layer and softplus activation. Nevertheless, a closed-form bound is hard to derive for a neural network with more than one hidden layer. [96] utilized terminologies from topology to study robustness. However, no robustness bounds

or estimates were provided for neural networks. On the other hand, works done by [27, 48, 49, 45] focused on formally verifying certain properties in neural networks for any possible input, and transform this formal verification problem into satisfiability modulo theory (SMT) and large-scale linear programming (LP) problems. These SMT or LP based approaches have high computational complexity and are only plausible for very small networks.

Intuitively, we can use the distortion of adversarial examples found by a certain attack algorithm as a robustness metric. For example, [5] proposed a LP formulation to find adversarial examples and use the perturbations as the robustness metric. They observed that the LP formulation can find adversarial examples with smaller perturbations than other gradient-based attacks like L-BFGS [94]. However, the perturbation found by these algorithms is an *upper bound* of the true minimum perturbation and depends on specific attack algorithms. These methods differ from our proposed robustness measure CLEVER, because CLEVER is an estimation of the *lower bound* of the minimum perturbation and is *independent* of the attack algorithms. Additionally, unlike LP-based approaches which are impractical for large networks, CLEVER is computationally feasible for large networks like Inception-v3. The concept of minimum perturbation and upper/lower bound will be formally defined in the next section.

We note that CLEVER is an attack-independent robustness metric that applies to any neural network classifier, as long as the neural network is locally bounded. In contrast, the robustness metric proposed in [41], albeit attack-agnostic, only applies to a neural network classifier with one hidden layer and the neural network has to be continuous differentiable.

3.2 Theory on Lipschitz Continuity

In this section, we provide formal robustness guarantees of a classifier in Theorem 3.2.6. Our robustness guarantees are general since they only require a mild assumption on Lipschitz continuity of the classification function. For differentiable classification functions, our results are consistent with the main theorem in [41] but are obtained by a much simpler and more intuitive manner[1]. Furthermore, our robustness analysis can be easily extended to non-differentiable classification functions (e.g. neural networks with ReLU) as in Lemma 3.2.10, whereas the analysis in [41] is restricted to differentiable functions. Specifically, Corollary 3.2.8 shows that the robustness analysis in [41] is in fact a special case of our analysis. We start our analysis by defining the notion of adversarial examples, minimum ℓ_p perturbations, and lower/upper bounds. All the notations are summarized in Table 2.1 in Section 2.3.

Definition 3.2.1 (perturbed example and adversarial example)**.** *Let $\mathbf{x_0} \in \mathbb{R}^{n_0}$ be an input vector of a K-class classification function $f : \mathbb{R}^{n_0} \to \mathbb{R}^K$ and the prediction is given as $c(\mathbf{x_0}) = \text{argmax}_{1 \leq i \leq K} f_i(\mathbf{x_0})$. Given $\mathbf{x_0}$, we say $\mathbf{x}_a$ is a perturbed example of $\mathbf{x_0}$ with perturbation $\boldsymbol{\delta} \in \mathbb{R}^{n_0}$ and ℓ_p-norm value $\|\delta\|_p$. An adversarial example is a perturbed example $\mathbf{x}_a$ that changes $c(\mathbf{x_0})$. A successful untargeted attack is to find a $\mathbf{x}_a$ such that $c(\mathbf{x}_a) \neq c(\mathbf{x_0})$ while a successful targeted attack is to find a $\mathbf{x}_a$ such that $c(\mathbf{x}_a) = t$ given a target class $t \neq c(\mathbf{x_0})$.*

Definition 3.2.2 (minimum adversarial perturbation $r_{p,\text{min}}$)**.** *Given an input vector $\mathbf{x_0}$ of a classifier f, the minimum ℓ_p adversarial perturbation of $\mathbf{x_0}$, denoted as $r_{p,min}$, is defined as the smallest ℓ_p-norm perturbation over all adversarial examples of $\mathbf{x_0}$.*

Definition 3.2.3 (lower bound of $r_{p,\text{min}}$)**.** *Suppose $r_{p,min}$ is the minimum adversarial perturbation of $\mathbf{x_0}$. A lower bound of $r_{p,min}$, denoted by $r_{p,L}$ where $r_{p,L} \leq r_{p,min}$, is defined such that any perturbed examples of $\mathbf{x_0}$ with $\|\boldsymbol{\delta}\|_p \leq r_{p,L}$ are not adversarial examples.*

[1] The authors in [41] implicitly assume Lipschitz continuity and use Mean Value Theorem and Hölder's Inequality to prove their main theorem. Here we provide a simple and direct proof with Lipschitz continuity assumption and without involving Mean Value Theorem and Hölder's Inequality.

Definition 3.2.4 (upper bound of $r_{p,\min}$)**.** *Suppose $r_{p,min}$ is the minimum adversarial perturbation of $\mathbf{x_0}$. An upper bound of $r_{p,min}$, denoted by $r_{p,U}$ where $r_{p,U} \geq r_{p,min}$, is defined such that there exists an adversarial example of $\mathbf{x_0}$ with $\|\boldsymbol{\delta}\|_p \geq r_{p,U}$.*

The lower and upper bounds are instance-specific because they depend on the input $\mathbf{x_0}$. While $r_{p,U}$ can be easily computed by finding an adversarial example of $\mathbf{x_0}$ using any attack method, non-trivial $r_{p,L}$ is not easy to find (the trivial $r_{p,L}$ is 0). It is because $r_{p,L}$ guarantees that the classifier is robust to any perturbations with $\|\delta\|_p \leq r_{p,L}$, certifying the robustness of the classifier. Below we show how to derive a formal robustness guarantee of a classifier with Lipschitz continuity assumption. Specifically, our analysis obtains a lower bound of ℓ_p minimum adversarial perturbation, which can be derived as $r_{p,L} = \min_{j \neq c} \frac{f_c(\mathbf{x_0}) - f_j(\mathbf{x_0})}{L_q^j}$ in Theorem 3.2.6. We first start with Lemma 3.2.5 and then provide a formal guarantee of the lower bound $r_{p,L}$ in Theorem 3.2.6.

Lemma 3.2.5 (Lipschitz continuity and its relationship with gradient norm [79])**.** *Let $S \subset \mathbb{R}^{n_0}$ be a convex bounded closed set and let $h(\mathbf{x}) : S \to \mathbb{R}$ be a continuously differentiable function on an open set containing S. Then, $h(\mathbf{x})$ is a Lipschitz function with Lipschitz constant L_q if the following inequality holds for any $\mathbf{x}, \mathbf{y} \in S$:*

$$|h(\mathbf{x}) - h(\mathbf{y})| \leq L_q \|\mathbf{x} - \mathbf{y}\|_p, \tag{3.1}$$

where $L_q = \max\{\|\nabla h(\mathbf{x})\|_q : \mathbf{x} \in S\}, \nabla h(\mathbf{x}) = (\frac{\partial h(\mathbf{x})}{\partial x_1}, \cdots, \frac{\partial h(\mathbf{x})}{\partial x_d})^\top$ is the gradient of $h(\mathbf{x})$, and $\frac{1}{p} + \frac{1}{q} = 1, 1 \leq p, q \leq \infty$.

Proof. For any $\mathbf{x}, \mathbf{y}$, let $\boldsymbol{d} = \frac{\mathbf{y}-\mathbf{x}}{\|\mathbf{y}-\mathbf{x}\|_p}$ be the unit vector pointing from $\mathbf{x}$ to $\mathbf{y}$ and $r = \|\mathbf{y} - \mathbf{x}\|_p$. Define uni-variate function $u(z) = h(\mathbf{x} + z\boldsymbol{d})$, then $u(0) = h(\mathbf{x})$ and $u(r) = h(\mathbf{y})$ and observe that $D^+h(\mathbf{x}+z\boldsymbol{d}; \boldsymbol{d})$ and $D^+h(\mathbf{x}+z\boldsymbol{d}; -\boldsymbol{d})$ are the right-hand and left-hand derivatives of $u(z)$, we have

$$u'(z) = \begin{cases} D^+h(\mathbf{x} + z\boldsymbol{d}; \boldsymbol{d}) \leq L_q & \text{if } D^+h(\mathbf{x} + z\boldsymbol{d}; \boldsymbol{d}) = D^+h(\mathbf{x} + z\boldsymbol{d}; -\boldsymbol{d}) \\ \text{undefined} & \text{if } D^+h(\mathbf{x} + z\boldsymbol{d}; \boldsymbol{d}) \neq D^+h(\mathbf{x} + z\boldsymbol{d}; -\boldsymbol{d}) \end{cases}$$

For ReLU network, there can be at most a finite number of points in $z \in (0, r)$ such that $g'(z)$ does not exist. This can be shown because each discontinuous z is caused by some ReLU activation, and there are only finite combinations. Let $0 = z_0 < z_1 < \cdots < z_{k-1} < z_k = 1$ be those points. Then, using the fundamental theorem of calculus on each interval separately, there exists $\bar{z}_i \in (z_i, z_{i-1})$ for each i such that

$$\begin{aligned} u(r) - u(0) &\leq \sum_{i=1}^{k} |u(z_i) - u(z_{i-1})| \\ &\leq \sum_{i=1}^{k} |u'(\bar{z}_i)(z_i - z_{i-1})| && \text{(Mean value theorem)} \\ &\leq \sum_{i=1}^{k} L_q |z_i - z_{i-1}|_p \\ &= L_q \|x - y\|_p. && (z_i \text{ are in line } (x, y)) \end{aligned}$$

Theorem 3.2.6 and its corollaries remain valid after replacing Lemma 3.2.5 with Lemma 3.2.10. □

Theorem 3.2.6 (Formal guarantee on lower bound $r_{p,L}$ for untargeted attack)**.** *Let $\mathbf{x_0} \in \mathbb{R}^{n_0}$ and $f : \mathbb{R}^{n_0} \to \mathbb{R}^K$ be a multi-class classifier with continuously differentiable components f_i and let $c = \operatorname{argmax}_{1 \leq i \leq K} f_i(\mathbf{x_0})$ be the class which f predicts for $\mathbf{x_0}$. For all $\boldsymbol{\delta} \in \mathbb{R}^{n_0}$ with*

$$\|\boldsymbol{\delta}\|_p \leq \min_{j \neq c} \frac{f_c(\mathbf{x_0}) - f_j(\mathbf{x_0})}{L_q^j}, \tag{3.2}$$

$\operatorname{argmax}_{1 \leq i \leq K} f_i(\mathbf{x_0} + \boldsymbol{\delta}) = c$ *holds with* $\frac{1}{p} + \frac{1}{q} = 1, 1 \leq p, q \leq \infty$ *and L_q^j is the Lipschitz constant for the function $f_c(\mathbf{x}) - f_j(\mathbf{x})$ in ℓ_p norm. In other words, $r_{p,L} = \min_{j \neq c} \frac{f_c(\mathbf{x_0}) - f_j(\mathbf{x_0})}{L_q^j}$ is a lower bound of minimum perturbation.*

Proof. According to Lemma 3.2.5, the assumption that $g(\mathbf{x}) := f_c(\mathbf{x}) - f_j(\mathbf{x})$ is Lipschitz continuous with Lipschitz constant L_q^j gives

$$|g(\mathbf{x}) - g(\mathbf{y})| \leq L_q^j \|\mathbf{x} - \mathbf{y}\|_p. \tag{3.3}$$

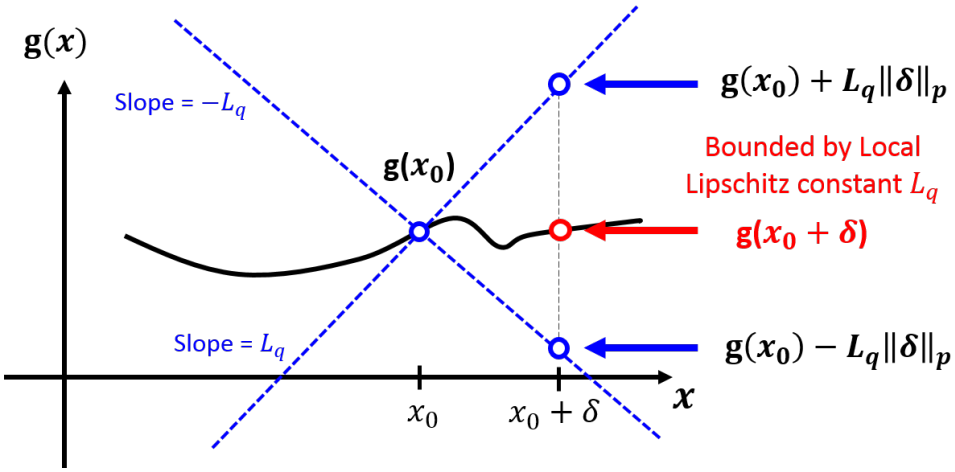


Figure 3-1: Intuitions behind Theorem 3.2.6.

Let $\mathbf{x} = \mathbf{x_0} + \boldsymbol{\delta}$ and $\mathbf{y} = \mathbf{x_0}$ in (3.3), we get

$$|g(\mathbf{x_0} + \boldsymbol{\delta}) - g(\mathbf{x_0})| \leq L_q^j \|\boldsymbol{\delta}\|_p,$$

which can be rearranged into the following form

$$g(\mathbf{x_0}) - L_q^j \|\boldsymbol{\delta}\|_p \leq g(\mathbf{x_0} + \boldsymbol{\delta}) \leq g(\mathbf{x_0}) + L_q^j \|\boldsymbol{\delta}\|_p. \tag{3.4}$$

When $g(\mathbf{x_0} + \boldsymbol{\delta}) = 0$, an adversarial example is found. As indicated by (3.4), $g(\mathbf{x_0} + \boldsymbol{\delta})$ is lower bounded by $g(\mathbf{x_0}) - L_q^j \|\boldsymbol{\delta}\|_p$. If $\|\boldsymbol{\delta}\|_p$ is small enough such that $g(\mathbf{x_0}) - L_q^j \|\boldsymbol{\delta}\|_p \geq 0$, no adversarial examples can be found:

$$g(\mathbf{x_0}) - L_q^j \|\boldsymbol{\delta}\|_p \geq 0 \Rightarrow \|\boldsymbol{\delta}\|_p \leq \frac{g(\mathbf{x_0})}{L_q^j} \Rightarrow \|\boldsymbol{\delta}\|_p \leq \frac{f_c(\mathbf{x_0}) - f_j(\mathbf{x_0})}{L_q^j},$$

Finally, to achieve $\text{argmax}_{1 \leq i \leq K} f_i(\mathbf{x_0} + \boldsymbol{\delta}) = c$, we take the minimum of the bound on $\|\boldsymbol{\delta}\|_p$ in (3.2) over $j \neq c$. I.e. if

$$\|\boldsymbol{\delta}\|_p \leq \min_{j \neq c} \frac{f_c(\mathbf{x_0}) - f_j(\mathbf{x_0})}{L_q^j},$$

the classifier decision can *never* be changed and the attack will *never* succeed. □

The intuitions behind Theorem 3.2.6 is shown in Figure 3-1 with an one-dimensional example. The function value $g(x) = f_c(x) - f_j(x)$ near point x_0 is inside a double cone

formed by two lines passing $(x_0, g(x_0))$ and with slopes equal to $\pm L_q$, where L_q is the (local) Lipschitz constant of $g(x)$ near x_0. In other words, the function value of $g(x)$ around x_0, i.e. $g(x_0 + \delta)$ can be bounded by $g(x_0)$, δ and the Lipschitz constant L_q. When $g(x_0 + \delta)$ is decreased to 0, an adversarial example is found and the minimal change of δ is $\frac{g(x_0)}{L_q}$.

Remark 3.2.7. *L_q^j is the Lipschitz constant of the function involving cross terms: $f_c(\mathbf{x}) - f_j(\mathbf{x})$, hence we also call it cross Lipschitz constant following [41].*

To distinguish our analysis from [41], we show in Corollary 3.2.8 that we can obtain the same result in [41] by Theorem 3.2.6. In fact, the analysis in [41] is a special case of our analysis because the authors implicitly assume Lipschitz continuity on $f_i(\mathbf{x})$ when requiring $f_i(\mathbf{x})$ to be continuously differentiable. They use local Lipschitz constant (L_{q,x_0}) instead of global Lipschitz constant (L_q) to obtain a tighter bound in the adversarial perturbation $\boldsymbol{\delta}$.

Corollary 3.2.8 (Formal guarantee on $r_{p,L}$ for untargeted attack)**.** *Let L_{q,x_0}^j be local Lipschitz constant of function $f_c(\mathbf{x}) - f_j(\mathbf{x})$ at $\mathbf{x_0}$ over some fixed ball $B_p(\mathbf{x_0}, R) := \{\mathbf{x} \in \mathbb{R}^{n_0} \mid \|\mathbf{x} - \mathbf{x_0}\|_p \leq R\}$ and let $\boldsymbol{\delta} \in B_p(\mathbf{0}, R)$. By Theorem 3.2.6, we obtain the bound in [41]:*

$$\|\boldsymbol{\delta}\|_p \leq \min \left\{ \min_{j \neq c} \frac{f_c(\mathbf{x_0}) - f_j(\mathbf{x_0})}{L_{q,x_0}^j}, R \right\}. \tag{3.5}$$

Proof. By Lemma 3.2.5 and let $g = f_c - f_j$, we get $L_{q,x_0}^j = \max_{y \in B_p(x_0,R)} \|\nabla g(y)\|_q = \max_{y \in B_p(x_0,R)} \|\nabla f_j(y) - \nabla f_c(y)\|_q$, which then gives the bound in Theorem 2.1 of [41]. □

An important use case of Theorem 3.2.6 and Corollary 3.2.8 is the bound for targeted attack:

Corollary 3.2.9 (Formal guarantee on $r_{p,L}$ for targeted attack)**.** *Assume the same notation as in Theorem 3.2.6 and Corollary 3.2.8. For a specified target class j, we have $\|\boldsymbol{\delta}\|_p \leq \min \left\{ \frac{f_c(\mathbf{x_0}) - f_j(\mathbf{x_0})}{L_{q,x_0}^j}, R \right\}$.*

In addition, we further extend Theorem 3.2.6 to a special case of *non-differentiable* functions – neural networks with ReLU activations. In this case the Lipchitz constant

used in Lemma 3.1 can be replaced by the maximum norm of directional derivative, and our analysis above will go through.

Lemma 3.2.10 (Formal guarantee on $r_{p,L}$ for ReLU networks)**.** *Let $h(\cdot)$ be a l-layer ReLU neural network with W_i as the weights for layer i. We ignore bias terms as they don't contribute to gradient.*

$$h(\mathbf{x}) = \sigma(W_l\sigma(W_{l-1}\dots\sigma(W_1\mathbf{x})))$$

where $\sigma(u) = \max(0, u)$. Let $S \subset \mathbb{R}^{n_0}$ be a convex bounded closed set, then equation (3.1) holds with $L_q = \sup_{\mathbf{x}\in S}\{|\sup_{\|\boldsymbol{d}\|_p=1} D^+h(\mathbf{x};\boldsymbol{d})|\}$ where we define $D^+h(\mathbf{x};\boldsymbol{d}) := \lim_{t\to 0^+}\frac{h(\mathbf{x}+t\boldsymbol{d})-h(\mathbf{x})}{t}$ to be the one-sided directional direvative, then Theorem 3.2.6, Corollary 3.2.8 and Corollary 3.2.9 still hold.

3.3 CLEVER score: a Robustness Metric via Extreme Value Theory

In this section, we provide an algorithm to estimate our derived robustness metric $r_{p,L}$ with the aid of extreme value theory. We name the estimated robustness metric of the lower bound $r_{p,L}$ as "CLEVER score", which is the the first attack-agnostic score that applies to any neural network classifiers. Recall in the earlier section, we showed that the lower bound of network robustness is associated with $g(\mathbf{x_0})$ and its cross Lipschitz constant L^j_{q,x_0}, where $g(\mathbf{x_0}) = f_c(\mathbf{x_0}) - f_j(\mathbf{x_0})$ is readily available at the output of a classifier and L^j_{q,x_0} is defined as $\max_{\mathbf{x}\in B_p(\mathbf{x_0},R)} \|\nabla g(\mathbf{x})\|_q$. Although $\nabla g(\mathbf{x})$ can be calculated easily via back propagation, computing L^j_{q,x_0} is more involved because it requires obtaining the maximum value of $\|\nabla g(\mathbf{x})\|_q$ in a ball. Exhaustive search on low dimensional $\mathbf{x}$ in $B_p(\mathbf{x}_0, R)$ seems already infeasible, not to mention the image classifiers with large feature dimensions of our interest. For instance, the input dimension $n_0 = 784$, 3072, 150528 for MNIST, CIFAR and ImageNet respectively.

One approach to compute L^j_{q,x_0} is through sampling a set of points $\mathbf{x}^{(i)}$ in a ball $B_p(\mathbf{x}_0, R)$ around $\mathbf{x}_0$ and taking the maximum value of $\|\nabla g(\mathbf{x}^{(i)})\|_q$. However, a significant amount of samples might be needed to obtain a good estimate of $\max \|\nabla g(\mathbf{x})\|_q$ and it is unknown how good the estimate is compared to the true maximum. Fortunately, Extreme Value Theory ensures that the maximum value of random variables can only follow one of the three extreme value distributions, which is useful to estimate $\max \|\nabla g(\mathbf{x})\|_q$ with only a tractable number of samples.

It is worth noting that although [103] also applied extreme value theory to estimate the Lipschitz constant, there are two main differences between their work and our proposal. First of all, the sampling methodology is entirely different. [103] calculate the slopes between pairs of sample points whereas we directly take samples on the norm of gradient as in Lemma 3.2.5. Secondly, the functions considered in [103] are only one-dimensional as opposed to the high-dimensional classification functions considered in our proposal. For comparison, we show in our experiment that the approach in [103], denoted as "SLOPE" in Table 3.2 and Figure 3-6, perform poorly

for high-dimensional classifiers such as deep neural networks.

3.3.1 Estimate Lipschitz constant via Extreme Value Theory

When sampling a point $\mathbf{x}$ uniformly in $B_p(\mathbf{x}_0, R)$, $\|\nabla g(\mathbf{x})\|_q$ can be viewed as a random variable characterized by a cumulative distribution function (CDF). For the purpose of illustration, we derived the CDF for a 2-layer neural network in Theorem 3.3.2. For any neural networks, suppose we have n samples $\{\|\nabla g(\mathbf{x}^{(i)})\|_q\}$, and denote them as a sequence of independent and identically distributed (iid) random variables $Y_1, Y_2, \cdots, Y_n$, each with CDF $F_Y(y)$. The CDF of $\max\{Y_1, \cdots, Y_n\}$, denoted as $F_Y^n(y)$, is called the limit distribution of $F_Y(y)$. Fisher-Tippett-Gnedenko theorem says that $F_Y^n(y)$, if exists, can only be one of the three family of extreme value distributions – the Gumbel class, the Fréchet class and the reverse Weibull class.

Theorem 3.3.1 (Fisher-Tippett-Gnedenko Theorem)**.** *If there exists a sequence of pairs of real numbers (a_n, b_n) such that $a_n > 0$ and $\lim_{n\to\infty} F_Y^n(a_n y + b_n) = G(y)$, where G is a non-degenerate distribution function, then G belongs to either the Gumbel class (Type I), the Fréchet class (Type II) or the Reverse Weibull class (Type III) with their CDFs as follows:*

$$\text{Gumbel class (Type I):} \quad G(y) = \exp\Big\{-\exp\Big[-\frac{y-a_W}{b_W}\Big]\Big\}, \quad y \in \mathbb{R},$$

$$\text{Fréchet class (Type II):} \quad G(y) = \begin{cases} 0, & \text{if } y < a_W, \\ \exp\{-\big(\frac{y-a_W}{b_W}\big)^{-c_W}\}, & \text{if } y \geq a_W, \end{cases}$$

$$\text{Reverse Weibull class (Type III):} \quad G(y) = \begin{cases} \exp\{-\big(\frac{a_W-y}{b_W}\big)^{c_W}\}, & \text{if } y < a_W, \\ 1, & \text{if } y \geq a_W, \end{cases}$$

where $a_W \in \mathbb{R}$, $b_W > 0$ and $c_W > 0$ are the location, scale and shape parameters, respectively.

Theorem 3.3.1 implies that the maximum values of the samples follow one of the three families of distributions. If $g(\mathbf{x})$ has a bounded Lipschitz constant, $\|\nabla g(\mathbf{x}^{(i)})\|_q$ is also bounded, thus its limit distribution must have a finite right end-point. We are

particularly interested in the reverse Weibull class, as its CDF has a finite right end-point (denoted as a_W). The right end-point reveals the upper limit of the distribution, known as the *extreme value*. The extreme value is exactly the unknown local cross Lipschitz constant $L^j_{q,\mathbf{x_0}}$ we would like to estimate here. To estimate $L^j_{q,\mathbf{x_0}}$, we first generate N_s samples of $\mathbf{x}^{(i)}$ over a fixed ball $B_p(\mathbf{x_0}, R)$ uniformly and independently in each batch with a total of N_b batches. We then compute $\|\nabla g(\mathbf{x}^{(i)})\|_q$ and store the maximum values of each batch in set S. Next, with samples in S, we perform a maximum likelihood estimation of reverse Weibull distribution parameters, and the location estimate $\hat{a}_W$ is used as an estimate of $L^j_{q,\mathbf{x_0}}$.

3.3.2 Compute CLEVER score of neural network classifiers

Given an instance $\mathbf{x_0}$, its classifier $f(\mathbf{x_0})$ and a target class j, a targeted CLEVER score of the classifier's robustness can be computed via $g(\mathbf{x_0})$ and L^j_{q,x_0}. Similarly, untargeted CLEVER scores can be computed. With the proposed procedure of estimating L^j_{q,x_0} described in the previous section, we summarize the flow of computing CLEVER score for both targeted attacks and un-targeted attacks in Algorithm 1 and 2, respectively.

Algorithm 1: CLEVER-t, compute CLEVER score for **t**argeted attack

Input: a K-class classifier $f(\mathbf{x})$, data example $\mathbf{x_0}$ with predicted class c, target class j, batch size N_b, number of samples per batch N_s, perturbation norm p, maximum perturbation R

Result: CLEVER Score $\mu \in \mathbb{R}_+$ for target class j

$S \leftarrow \{\emptyset\}$, $g(\mathbf{x}) \leftarrow f_c(\mathbf{x}) - f_j(\mathbf{x})$, $q \leftarrow \frac{p}{p-1}$.
for $i \leftarrow 1$ **to** N_b **do**
  for $k \leftarrow 1$ **to** N_s **do**
    randomly select a point $\mathbf{x}^{(i,k)} \in B_p(\mathbf{x_0}, R)$
    compute $b_{ik} \leftarrow \|\nabla g(\mathbf{x}^{(i,k)})\|_q$ via back propagation
  end
  $S \leftarrow S \cup \{\max_k\{b_{ik}\}\}$
end
$\hat{a}_W \leftarrow$ MLE of location parameter of reverse Weibull distribution on S
$\mu \leftarrow \min(\frac{g(\mathbf{x_0})}{\hat{a}}, R)$

Algorithm 2: CLEVER-u, compute CLEVER score for **u**n-targeted attack

Input: Same as Algorithm 1, but without a target class j

Result: CLEVER score $\nu \in \mathbb{R}_+$ for un-targeted attack

for $j \leftarrow 1$ **to** K, $j \neq c$ **do**
  $\mu_j \leftarrow \text{CLEVER} - \mathtt{t}(f, \mathbf{x_0}, c, j, N_b, N_s, p, R)$
end
$\nu \leftarrow \min_j\{\mu_j\}$

In the following, we derive $F_Y(y)$ for a one-hidden-layer neural network in Theorem 3.3.2 and give some insights on the gradient norm of a one-hidden-layer neural network.

Theorem 3.3.2 ($F_Y(y)$ of one-hidden-layer neural network)**.** *Consider a neural network $f : \mathbb{R}^d \to \mathbb{R}^K$ with input $\mathbf{x_0} \in \mathbb{R}^d$, a hidden layer with U hidden neurons, and rectified linear unit (ReLU) activation function. If we sample uniformly in a ball $B_p(\mathbf{x_0}, R)$, then the cumulative distribution function of $\|\nabla g(\mathbf{x})\|_q$, denoted as $F_Y(y)$,*

is piece-wise linear with at most $M = \sum_{i=0}^{d} \binom{U}{i}$ *pieces, where* $g(\mathbf{x}) = f_c(\mathbf{x}) - f_j(\mathbf{x})$ *for some given* c *and* j*, and* $\frac{1}{p} + \frac{1}{q} = 1, 1 \leq p, q \leq \infty$.

Proof. The j_{th} output of a one-hidden-layer neural network can be written as

$$f_j(\mathbf{x}) = \sum_{r=1}^{U} \boldsymbol{V}_{jr} \cdot \sigma\left(\sum_{i=1}^{d} \boldsymbol{W}_{ri} \cdot x_i + b_r\right) = \sum_{r=1}^{U} \boldsymbol{V}_{jr} \cdot \sigma\left(\boldsymbol{w}_r \mathbf{x} + b_r\right),$$

where $\sigma(z) = \max(z, 0)$ is ReLU activation function, $\boldsymbol{W}$ and $\boldsymbol{V}$ are the weight matrices of the first and second layer respectively, and $\boldsymbol{w}_r$ is the r_{th} row of $\boldsymbol{W}$. Thus, we can compute $g(\mathbf{x})$ and $\|\nabla g(\mathbf{x})\|_q$ below:

$$\begin{aligned} g(\mathbf{x}) = f_c(\mathbf{x}) - f_j(\mathbf{x}) &= \sum_{r=1}^{U} \boldsymbol{V}_{cr} \cdot \sigma\left(\boldsymbol{w}_r \mathbf{x} + b_r\right) - \sum_{r=1}^{U} \boldsymbol{V}_{jr} \cdot \sigma\left(\boldsymbol{w}_r \mathbf{x} + b_r\right) \\ &= \sum_{r=1}^{U} (\boldsymbol{V}_{cr} - \boldsymbol{V}_{jr}) \cdot \sigma\left(\boldsymbol{w}_r \mathbf{x} + b_r\right) \end{aligned}$$

and

$$\|\nabla g(\mathbf{x})\|_q = \left\| \sum_{r=1}^{U} \mathbb{I}(\boldsymbol{w}_r \mathbf{x} + b_r)(\boldsymbol{V}_{cr} - \boldsymbol{V}_{jr}) \boldsymbol{w}_r^{\top} \right\|_q,$$

where $\mathbb{I}(z)$ is an univariate indicator function:

$$\mathbb{I}(z) = \begin{cases} 1, & \text{if } z > 0, \\ 0, & \text{if } z \leq 0. \end{cases}$$

As illustrated in Figure 3-2, the hyperplanes $\boldsymbol{w}_r \mathbf{x} + b_r = 0, r \in \{1, \ldots, U\}$ divide the d dimensional spaces $\mathbb{R}^d$ into different regions, with the interior of each region satisfying a different set of inequality constraints, e.g. $\boldsymbol{w}_{r_+} \mathbf{x} + b_{r_+} > 0$ and $\boldsymbol{w}_{r_-} \mathbf{x} + b_{r_-} < 0$. Given $\mathbf{x}$, we can identify which region it belongs to by checking the sign of $\boldsymbol{w}_r \mathbf{x} + b_r$ for each r. Notice that the gradient norm is the same for all the points in the same region, i.e. for any $\mathbf{x}_1$, $\mathbf{x}_2$ satisfying $\mathbb{I}(\boldsymbol{w}_r \mathbf{x}_1 + b_r) = \mathbb{I}(\boldsymbol{w}_r \mathbf{x}_2 + b_r)\ \forall r$, we have $\|\nabla g(\mathbf{x}_1)\|_q = \|\nabla g(\mathbf{x}_2)\|_q$. Since there can be at most $M = \sum_{i=0}^{d} \binom{U}{i}$ different regions for a d-dimensional space with U hyperplanes, $\|\nabla g(\mathbf{x})\|_q$ can take at most M different values.

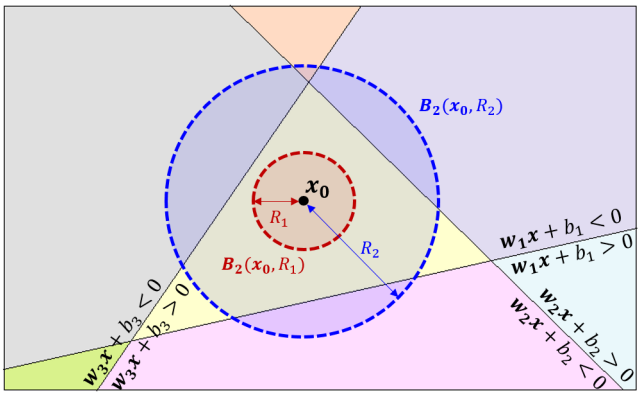


Figure 3-2: Illustration of Theorem 3.3.2 with $d = 2, q = 2$ and $U = 3$. The three hyperplanes $\boldsymbol{w}_i\mathbf{x} + b_i = 0$ divide the space into seven regions (with different colors). The red dash line encloses the ball $B_2(\mathbf{x_0}, R_1)$ and the blue dash line encloses a larger ball $B_2(\mathbf{x_0}, R_2)$. If we draw samples uniformly within the balls, the probability of $\|\nabla g(\mathbf{x})\|_2 = y$ is proportional to the intersected volumes of the ball and the regions with $\|\nabla g(\mathbf{x})\|_2 = y$.

Therefore, if we perform uniform sampling in a ball $B_p(\mathbf{x_0}, R)$ centered at $\mathbf{x_0}$ with radius R and denote $\|\nabla g(\mathbf{x})\|_q$ as a random variable Y, the probability distribution of Y is discrete and its CDF is piece-wise constant with at most M pieces. Without loss of generality, assume there are $M_0 \leq M$ distinct values for Y and denote them as $m_{(1)}, m_{(2)}, \ldots, m_{(M_0)}$ in an increasing order, the CDF of Y, denoted as $F_Y(y)$, is the following:

$$F_Y(m_{(i)}) = F_Y(m_{(i-1)}) + \frac{\mathbb{V}_d(\{\mathbf{x} \mid \|\nabla g(\mathbf{x})\|_q = m_{(i)}\}) \cap \mathbb{V}_d(B_p(\mathbf{x_0}, R)))}{\mathbb{V}_d(B_p(\mathbf{x_0}, R))}, i = 1, \ldots, M_0,$$

where $F_Y(m_{(0)}) = 0$ with $m_{(0)} < m_{(1)}$, $\mathbb{V}_d(E)$ is the volume of E in a d dimensional space. □

3.4 Experimental Results

3.4.1 Networks and Parameter Setup

We conduct experiments on CIFAR-10 (CIFAR for short), MNIST, and ImageNet data sets. For the former two smaller datasets CIFAR and MNIST, we evaluate CLEVER scores on four relatively small networks: a single hidden layer MLP with softplus activation (with the same number of hidden units as in [41]), a 7-layer AlexNet-like CNN (with the same structure as in [13]), and the 7-layer CNN with defensive distillation [76] (DD) and bounded ReLU [105] (BReLU) defense techniques employed.

For ImageNet data set, we use three popular deep network architectures: a 50-layer Residual Network [39] (ResNet-50), Inception-v3 [93] and MobileNet [44]. They were chosen for the following reasons: (i) they all yield (close to) state-of-the-art performance among equal-sized networks; and (ii) their architectures are significantly different with unique building blocks, i.e., residual block in ResNet, inception module in Inception net, and depthwise separable convolution in MobileNet. Therefore, their diversity in network architectures is appropriate to test our robustness metric. For MobileNet, we set the width multiplier to 1.0, achieving a 70.6% accuracy on ImageNet. We used public pretrained weights for all ImageNet models[2].

In all our experiments, we set the sampling parameters $N_b = 500$, $N_s = 1024$ and $R = 5$. For targeted attacks, we use 500 test-set images for CIFAR and MNIST and use 100 test-set images for ImageNet; for each image, we evaluate its targeted CLEVER score for three targets: a random target class, a least likely class (the class with lowest probability when predicting the original example), and the top-2 class (the class with largest probability except for the true class, which is usually the easiest target to attack). We also conduct untargeted attacks on MNIST and CIFAR for 100 test-set images, and evaluate their untargeted CLEVER scores. Our experiment code is publicly available[3].

[2] Pretrained models can be downloaded at https://github.com/tensorflow/models/tree/master/research/slim

[3] Source code is available at https://github.com/IBM/CLEVER-Robustness-Score

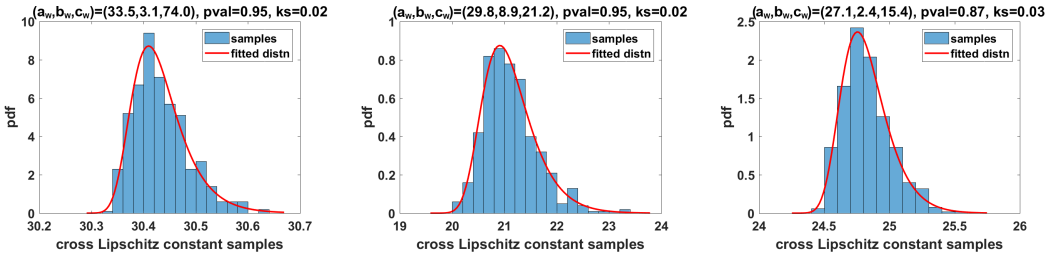
3.4.2 Fitting Gradient Norm samples with Reverse Weibull distributions

We fit the cross Lipschitz constant samples in S (see Algorithm 1) with reverse Weibull class distribution to obtain the maximum likelihood estimate of the location parameter $\hat{a}_W$, scale parameter $\hat{b}_W$ and shape parameter $\hat{c}_W$, as introduced in Theorem 3.3.1. To validate that reverse Weibull distribution is a good fit to the empirical distribution of the cross Lipschitz constant samples, we conduct Kolmogorov-Smirnov goodness-of-fit test (a.k.a. K-S test) to calculate the K-S test statistics D and corresponding p-values. The null hypothesis is that samples S follow a reverse Weibull distribution.

Figure 3-3 plots the probability distribution function of the cross Lipschitz constant samples and the fitted Reverse Weibull distribution for images from various data sets and network architectures. The estimated MLE parameters, p-values, and the K-S test statistics D are also shown. We also calculate the percentage of examples whose estimation have p-values greater than 0.05, as illustrated in Figure 3-4. If the p-value is greater than 0.05, the null hypothesis cannot be rejected, meaning that the underlying data samples fit a reverse Weibull distribution well. Figure 3-4 shows that all numbers are close to 100%, validating the use of reverse Weibull distribution as an underlying distribution of gradient norm samples empirically. Therefore, the fitted location parameter of reverse Weibull distribution (i.e., the extreme value), $\hat{a}_W$, can be used as a good estimation of local cross Lipschitz constant to calculate the CLEVER score. The exact numbers are shown in Table 3.4.

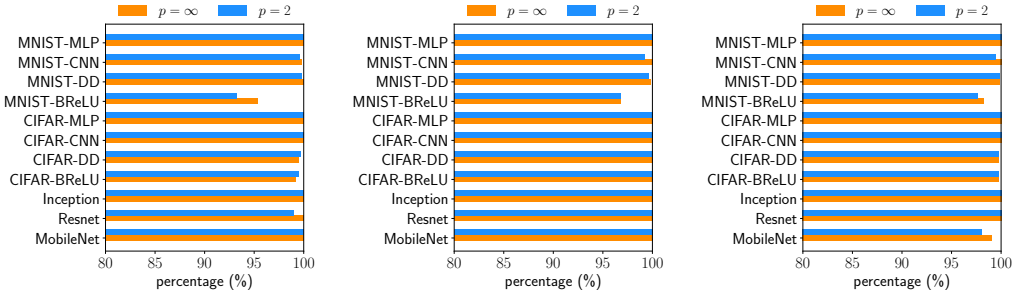
3.4.3 Comparing CLEVER Score with Attack-specific Network Robustness

We apply the state-of-the-art white-box attack methods, iterative fast gradient sign method (I-FGSM) [37, 56] and Carlini and Wagner's attack (CW) [13], to find adversarial examples for 11 networks, including 4 networks trained on CIFAR, 4 networks trained on MNIST, and 3 networks trained on ImageNet. For CW attack, we run 1000 iterations for ImageNet and CIFAR, and 2000 iterations for MNIST,



(a) CIFAR-MLP (b) MNIST-CNN (c) ImageNet-MobileNet

Figure 3-3: The cross Lipschitz constant samples for three images from CIFAR, MNIST and ImageNet datasets, and their fitted Reverse Weibull distributions with the corresponding MLE estimates of location, scale and shape parameters (a_W, b_W, c_W) shown on the top of each plot. The D-statistics of K-S test and p-values are denoted as *ks* and *pval*. With small *ks* and high p-value, the hypothesized reverse Weibull distribution fits the empirical distribution of cross Lipschitz constant samples well.



(a) Least likely target (b) Random target (c) Top 2 target

Figure 3-4: The percentage of examples whose null hypothesis (the samples S follow a reverse Weibull distribution) cannot be rejected by K-S test with a significance level of 0.05 for $p = 2$ and $p = \infty$. All numbers for each model are close to 100%, indicating S fits reverse Weibull distributions well.

as MNIST has shown to be more difficult to attack [15]. Attack learning rate is individually tuned for each model: 0.001 for Inception-v3 and ResNet-50, 0.0005 for MobileNet and 0.01 for all other networks. For I-FGSM, we run 50 iterations and choose the optimal $\epsilon \in \{0.01, 0.025, 0.05, 0.1, 0.3, 0.5, 0.8, 1.0\}$ to achieve the smallest ℓ_∞ distortion for each individual image. For defensively distilled (DD) networks, 50 iterations of I-FGSM are not sufficient; we use 250 iterations for CIFAR-DD and 500 iterations for MNIST-DD to achieve a 100% success rate. For the problem to be non-trivial, images that are classified incorrectly are skipped. We report 100% attack success rates for all the networks, and thus the average distortion of adversarial

Table 3.1: Comparison between the average untargeted CLEVER score and distortion found by CW and I-FGSM untargeted attacks. DD and BReLU represent Defensive Distillation and Bounded ReLU defending methods applied to the baseline CNN network.

	CW		I-FGSM		CLEVER	
	ℓ_2	ℓ_∞	ℓ_2	ℓ_∞	ℓ_2	ℓ_∞
MNIST-MLP	1.113	0.215	3.564	0.178	0.819	0.041
MNIST-CNN	1.500	0.455	4.439	0.288	0.721	0.057
MNIST-DD	1.548	0.409	5.617	0.283	0.865	0.063
MNIST-BReLU	1.337	0.433	3.851	0.285	0.833	0.065
CIFAR-MLP	0.253	0.018	0.885	0.016	0.219	0.005
CIFAR-CNN	0.195	0.023	0.721	0.018	0.072	0.002
CIFAR-DD	0.285	0.032	1.136	0.024	0.130	0.004
CIFAR-BReLU	0.159	0.019	0.519	0.013	0.045	0.001

examples can indicate the attack-specific robustness of each network. For comparison, we compute the CLEVER scores for the same set of images and attack targets. To the best of our knowledge, CLEVER is the first attack-independent robustness score that is capable of handling the large networks studied here, so we directly compare it with the attack-induced distortion metrics in our study.

We evaluate the effectiveness of our CLEVER score by comparing the upper bound $r_{p,U}$ (found by attacks) and CLEVER score, where CLEVER serves as an estimated lower bound, $r_{p,L}$. Table 3.2 compares the average ℓ_2 and ℓ_∞ distortions of adversarial examples found by targeted CW and I-FGSM attacks and the corresponding average *targeted* CLEVER scores for ℓ_2 and ℓ_∞ norms, and Figure 3-6 visualizes the results for ℓ_∞ norm. Similarly, Table 3.1 compares untargeted CW and I-FGSM attacks with *untargeted* CLEVER scores. As expected, CLEVER is smaller than the distortions of adversarial images in most cases. More importantly, since CLEVER is independent of attack algorithms, the reported CLEVER scores can roughly indicate the distortion of the *best possible* attack in terms of a specific ℓ_p distortion. The average ℓ_2 distortion found by CW attack is close to the ℓ_2 CLEVER score, indicating CW is a strong ℓ_2 attack. In addition, when a defense mechanism (Defensive Distillation or Bounded ReLU) is used, the corresponding CLEVER scores are consistently increased (except for CIFAR-BReLU), indicating that the network is indeed made more resilient to

adversarial perturbations. For CIFAR-BReLU, both CLEVER scores and ℓ_p norm of adversarial examples found by CW attack decrease, implying that bound ReLU is an ineffective defense for CIFAR. CLEVER scores can be seen as a security checkpoint for unseen attacks. For example, if there is a substantial gap in distortion between the CLEVER score and the considered attack algorithms, it may suggest the existence of a more effective attack that can close the gap.

Since CLEVER score is derived from an estimation of the robustness lower bound, we further verify the viability of CLEVER per each example, i.e., whether it is usually smaller than the upper bound found by attacks. Table 3.3 shows the percentage of inaccurate estimations where the CLEVER score is larger than the distortion of adversarial examples found by CW and I-FGSM attacks in three ImageNet networks. We found that CLEVER score provides an accurate estimation for most of the examples. For MobileNet and Resnet-50, our CLEVER score is a strict lower bound of these two attacks for more than 96% of tested examples. For Inception-v3, the condition of strict lower bound is worse (still more than 75%), but we found that in these cases the attack distortion only differs from our CLEVER score by a fairly small amount. In Figure 3-7 we show the empirical CDF of the gap between CLEVER score and the ℓ_2 norm of adversarial distortion generated by CW attack for the same set of images in Table 3.3. In Figure 3-8, we plot the ℓ_2 distortion and CLEVER scores for each individual image. A positive gap indicates that CLEVER (estimated lower bound) is indeed less than the upper bound found by CW attack. Most images have a small positive gap, which signifies the near-optimality of CW attack in terms of ℓ_2 distortion, as CLEVER suffices for an estimated capacity of the best possible attack.

3.4.4 Time v.s. Estimation Accuracy

In Figure 3-9, we vary the number of samples ($N_b = 50, 100, 250, 500$) and compute the ℓ_2 CLEVER scores for three large ImageNet models, Inception-v3, ResNet-50 and MobileNet. We observe that 50 or 100 samples are usually sufficient to obtain a reasonably accurate robustness estimation despite using a smaller number of samples. On a single GTX 1080 Ti GPU, the cost of 1 sample (with $N_s = 1024$) is measured

as 2.9 s for MobileNet, 5.0 s for ResNet-50 and 8.9 s for Inception-v3, thus the computational cost of CLEVER is feasible for state-of-the-art large-scale deep neural networks.

Percentage of examples having p value > 0.05

Table 3.4 shows the percentage of examples where the null hypothesis cannot be rejected by K-S test, indicating that the maximum gradient norm samples fit reverse Weibull distribution well.

Table 3.4: Percentage of estimations where the null hypothesis cannot be rejected by K-S test for a significance level of 0.05. The bar plots of this table are illustrated in Figure 3-4.

	Least Likely		Random		Top-2	
	L_2	L_∞	L_2	L_∞	L_2	L_∞
MNIST-MLP	100.0	100.0	100.0	100.0	100.0	100.0
MNIST-CNN	99.6	99.8	99.2	100.0	99.4	100.0
MNIST-DD	99.8	100.0	99.6	99.8	99.8	99.8
MNIST-BReLU	93.3	95.4	96.8	96.8	97.6	98.2
CIFAR-MLP	100.0	100.0	100.0	100.0	100.0	100.0
CIFAR-CNN	100.0	100.0	100.0	100.0	100.0	100.0
CIFAR-DD	99.7	99.5	100.0	100.0	99.7	99.7
CIFAR-BReLU	99.5	99.2	100.0	100.0	99.7	99.7
Inception-v3	100.0	100.0	100.0	100.0	100.0	100.0
Resnet-50	99.0	100.0	100.0	100.0	100.0	100.0
MobileNet	100.0	100.0	100.0	100.0	98.0	99.0

CLEVER v.s. number of samples

Figure 3-10 shows the ℓ_2 CLEVER score with different number of samples ($N_b = 50, 100, 250, 500$) for MNIST and CIFAR models. For most models except MNIST-BReLU, reducing the number of samples only change CLEVER scores very slightly. For MNIST-BReLU, increasing the number of samples improves the estimated lower bound, suggesting that a larger number of samples is preferred. In practice, we can start with a relatively small $N_b = a$, and also try $2a, 4a, \cdots$ samples to see if CLEVER scores change significantly. If CLEVER scores stay roughly the same despite increasing

N_b, we can conclude that using $N_b = a$ is sufficient.

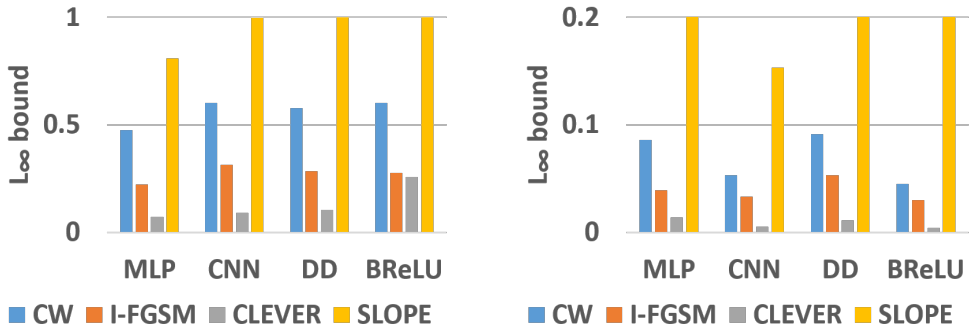
Table 3.2: Comparison of the average targeted CLEVER scores with average ℓ_∞ and ℓ_2 distortions found by CW, I-FSGM attacks and SLOPE [103]. DD and BReLU denote CNN networks trained with Defensive Distillation and Bounded ReLU defending methods. SLOPE in ImageNet networks is not included because it has been shown ineffective even for smaller networks.

(a) avergage ℓ_∞ distortion of CW and I-FGSM targeted attacks, and CLEVER and SLOPE estimation. Some very large SLOPE estimates (in parentheses) exceeding the maximum possible ℓ_∞ distortion are reported as 1.

	Least Likely Target				Random Target				Top-2 Target			
	CW	I-FGSM	CLEVER	SLOPE	CW	I-FGSM	CLEVER	SLOPE	CW	I-FGSM	CLEVER	SLOPE
MNIST-MLP	0.475	0.223	0.071	0.808	0.337	0.173	0.072	0.813	0.218	0.119	0.069	0.786
MNIST-CNN	0.601	0.313	0.090	0.996	0.550	0.264	0.088	0.982	0.451	0.211	0.070	0.826
MNIST-DD	0.578	0.283	0.103	1 (1.090)	0.531	0.238	0.091	0.953	0.412	0.165	0.091	0.984
MNIST-BReLU	0.601	0.276	0.257	1 (5.327)	0.544	0.238	0.187	3.907	0.442	0.196	0.117	1 (2.470)
CIFAR-MLP	0.086	0.039	0.014	0.294	0.051	0.024	0.014	0.284	0.019	0.013	0.014	0.286
CIFAR-CNN	0.053	0.033	0.005	0.153	0.042	0.023	0.005	0.148	0.022	0.013	0.004	0.129
CIFAR-DD	0.091	0.053	0.011	0.278	0.066	0.032	0.010	0.255	0.033	0.014	0.007	0.184
CIFAR-BReLU	0.045	0.030	0.004	0.250	0.034	0.022	0.003	0.173	0.018	0.012	0.002	0.095
Inception-v3	0.023	0.011	0.002	-	0.021	0.012	0.002	-	0.010	0.011	0.001	-
Resnet-50	0.031	0.015	0.002	-	0.025	0.012	0.002	-	0.010	0.010	0.001	-
MobileNet	0.025	0.010	0.003	-	0.018	0.010	0.002	-	0.006	0.010	0.001	-

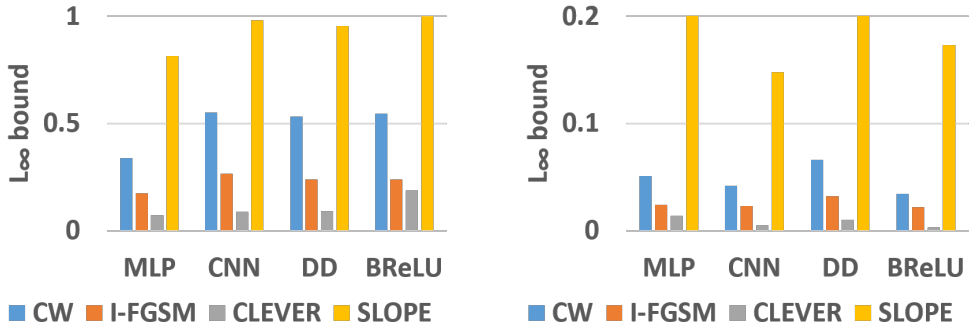
(b) average ℓ_2 distortion of CW and I-FGSM targeted attacks, and CLEVER and SLOPE estimation. Some very large SLOPE estimates (in parentheses) exceeding the sampling radius $R = 5$ are reported as 5.

	Least Likely Target				Random Target				Top-2 Target			
	CW	I-FGSM	CLEVER	SLOPE	CW	I-FGSM	CLEVER	SLOPE	CW	I-FGSM	CLEVER	SLOPE
MNIST-MLP	2.575	4.273	1.409	5 (8.028)	1.833	3.369	1.432	5 (8.102)	1.128	2.374	1.383	5 (7.853)
MNIST-CNN	2.377	4.417	1.257	5 (9.947)	2.005	3.902	1.227	5 (9.619)	1.504	3.242	0.987	5 (7.921)
MNIST-DD	2.644	4.957	1.532	5 (10.628)	2.240	4.253	1.340	5 (9.493)	1.542	3.010	1.330	5 (9.646)
MNIST-BReLU	2.349	5.170	3.312	5 (52.058)	1.923	4.544	2.565	5 (37.531)	1.404	3.778	1.583	5 (23.548)
CIFAR-MLP	1.123	1.896	0.620	5 (5.013)	0.673	1.214	0.597	4.806	0.262	0.689	0.599	4.949
CIFAR-CNN	0.836	1.067	0.156	2.630	0.372	0.837	0.146	2.497	0.188	0.552	0.123	2.195
CIFAR-DD	2.065	1.540	0.347	4.735	0.624	1.097	0.307	4.279	0.296	0.582	0.220	3.083
CIFAR-BReLU	0.407	0.928	0.140	4.125	0.303	0.732	0.103	2.944	0.152	0.494	0.052	1.564
Inception-v3	0.628	2.244	0.524	-	0.595	2.261	0.466	-	0.287	2.073	0.234	-
Resnet-50	0.767	2.410	0.357	-	0.647	2.098	0.299	-	0.212	1.682	0.134	-
MobileNet	0.837	2.195	0.617	-	0.603	2.066	0.439	-	0.190	1.771	0.144	-



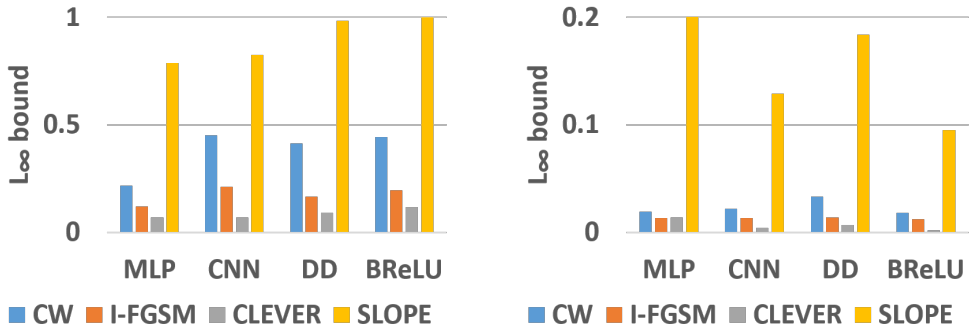
(a) MNIST: Least likely target

(b) CIFAR: Least likely target



(c) MNIST: Random target

(d) CIFAR: Random target



(e) MNIST: Top 2 target

(f) CIFAR: Top 2 target

Figure 3-6: Comparison of ℓ_∞ distortion obtained by CW and I-FGSM attacks, CLEVER score and the slope based Lipschitz constant estimation (SLOPE) by [103]. SLOPE significantly exceeds the distortions found by attacks, thus it is an inappropriate estimation of lower bound β_L.

Table 3.3: Percentage of images in ImageNet where the CLEVER score for that image is greater than the adversarial distortion found by different attacks.

	Least Likely Target				Random Target				Top-2 Target			
	CW		I-FGSM		CW		I-FGSM		CW		I-FGSM	
	L_2	L_∞	L_2	L_∞	L_2	L_∞	L_2	L_∞	L_2	L_∞	L_2	L_∞
MobileNet	4%	0%	0%	0%	2%	0%	0%	0%	0%	0%	0%	0%
Resnet-50	4%	0%	0%	0%	2%	0%	0%	0%	1%	0%	0%	0%
Inception-v3	25%	0%	0%	0%	23%	0%	0%	0%	15%	0%	0%	0%

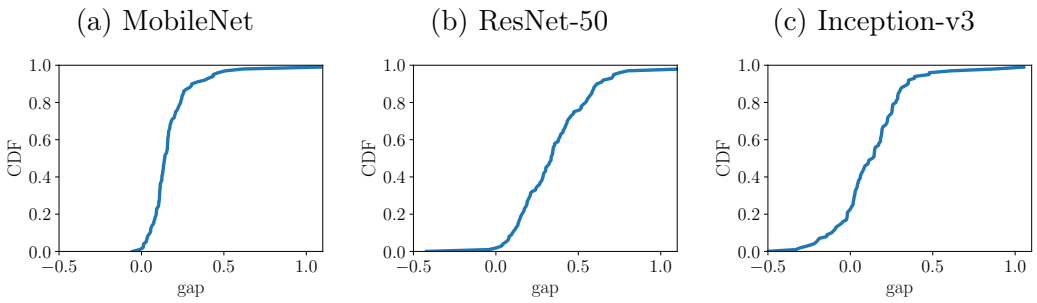


Figure 3-7: The empirical CDF of the gap between CLEVER score and the ℓ_2 norm of adversarial distortion generated by CW attack with random targets for 100 images on 3 ImageNet networks.

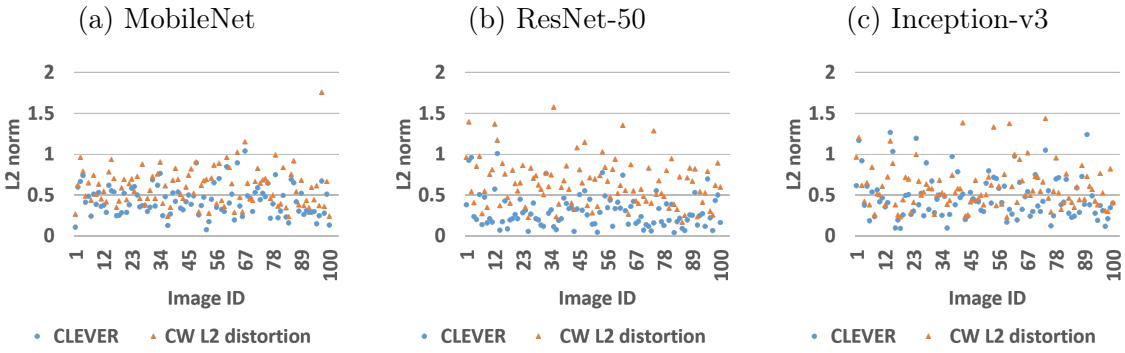


Figure 3-8: Comparison of the CLEVER scores (circle) and the ℓ_2 norm of adversarial distortion generated by CW attack (triangle) with random targets for 100 images. The x-axis is image ID and the y-axis is the ℓ_2 distortion metric.

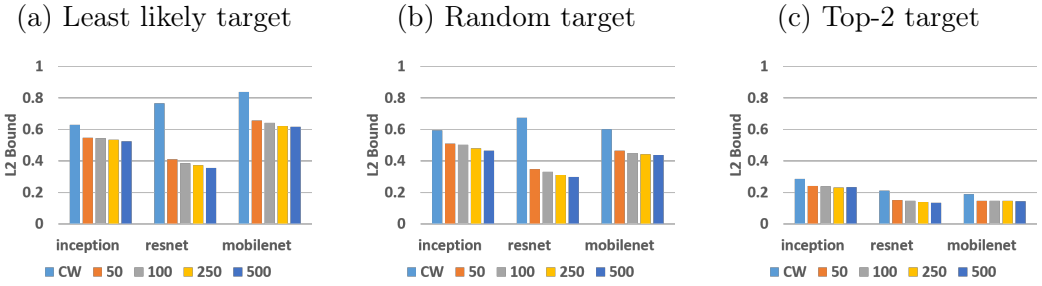
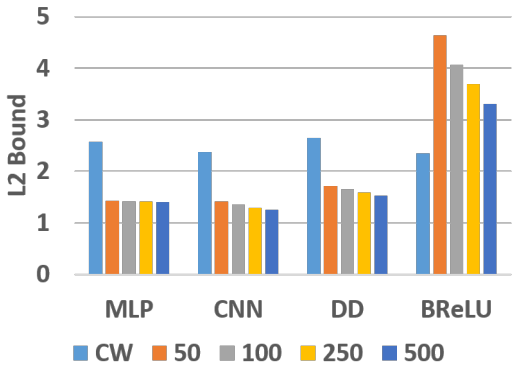
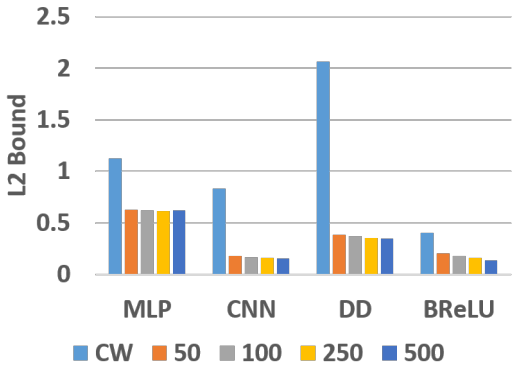


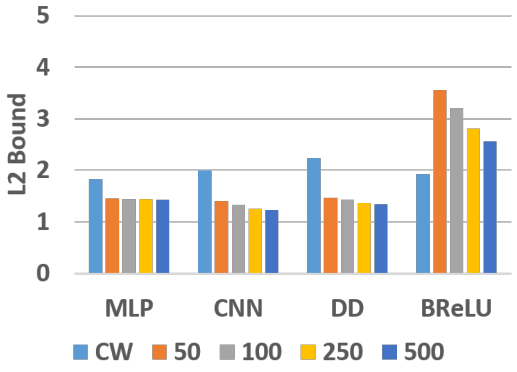
Figure 3-9: Comparison of the CLEVER score calculated by $N_b = \{50, 100, 250, 500\}$ and the ℓ_2 norm of adversarial distortion found by CW attack (CW) on 3 ImageNet models and 3 target types.



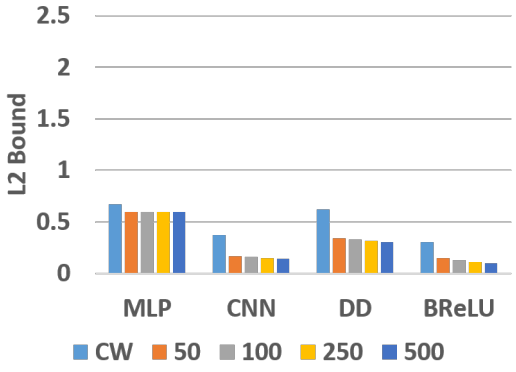
(a) MNIST, Least likely target



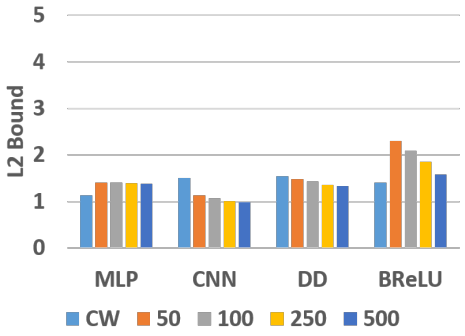
(b) CIFAR, Least likely target



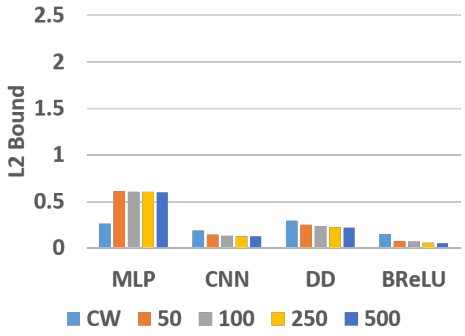
(c) MNIST, Random target



(d) CIFAR, Random target



(e) MNIST, Top-2 target



(f) CIFAR, Top2 target

Figure 3-10: Comparison of the CLEVER score calculated by $N_b = \{50, 100, 250, 500\}$ and the ℓ_2 norm of adversarial distortion found by CW attack (CW) on MNIST and CIFAR models with 3 target types.

3.5 Conclusion

In this Chapter, we propose the CLEVER score, a novel and generic score to evaluate the robustness of a neural network classifier against adversarial examples. Compared to the existing robustness evaluation approaches, our metric has the following advantages: (i) attack-agnostic; (ii) applicable to any neural network classifier; (iii) comes with strong theoretical guarantees; and (iv) is computationally feasible for large neural networks. Our extensive experiments show that the CLEVER score well matches the practical robustness indication of a wide range of natural and defended networks.

Chapter 4

Robustness Quantification via Adaptive Bounding Framework

4.1 Introduction

In the last Chapter, we have developed a robustness score, CLEVER score, to *estimate* the theoretical lower bound $r_{p,L}$ of the minimum adversarial perturbation $r_{p,\min}$. In this Chapter, we show that it is possible to *certify* the robustness level of an arbitrary natural example $\mathbf{x}_0$ by ensuring *all* its neighborhood has the same inference outcome (e.g., consistent top-1 prediction) given an arbitrary NN. In other words, it is possible to numerically compute $r_{p,L}$ with provable guarantees as opposed to estimation. This line of research is known as *local robustness level certification*, and $r_{p,L}$ is known as *certified* robustness lower bound [1].

To briefly recap, in this thesis, the neighborhood of $\mathbf{x_0}$ is characterized by an ℓ_p ball centered at $\mathbf{x_0}$, for any $p \geq 1$. Geometrically speaking, the minimum distance of a misclassified nearby example to $\mathbf{x_0}$ is the least adversary strength (a.k.a. minimum adversarial perturbation $r_{p,\min}$) required to alter the target model's prediction, which is also the largest possible robustness certificate for $\mathbf{x_0}$. Unfortunately, finding the minimum perturbation of adversarial examples in NNs with Rectified Linear Unit

[1] We note that people usually call this line of research as *robustness certification*, but we think it should not be misleading and thus in this thesis, we call it *local robustness level certification*.

(ReLU) activations, which is one of the most widely used activation functions, is known to be an NP-complete problem [48, 91]. This makes formal verification techniques such as Reluplex [48] computationally demanding even for small-sized NNs and suffer from scalability issues.

Although certifying the largest possible robustness radius is challenging for ReLU networks, the piece-wise linear nature of ReLUs can be exploited to efficiently compute a non-trivial *certified robustness lower bound* of the minimum perturbation [80, 52, 82, 101]. Beyond ReLU, one fundamental problem that remains largely unexplored is how to generalize the robustness certification technique to other popular activation functions that are not piece-wise linear, such as tanh and sigmoid, and how to motivate and certify the design of other activation functions towards improved robustness. In this Chapter, we tackle the preceding problem by proposing an efficient robustness certification framework for NNs with general activation functions.

4.1.1 Contributions

Our main contributions in this chapter are summarized as follows:

1. We propose a generic analysis framework CROWN for certifying NNs using linear or quadratic upper and lower bounds for *general* activation functions that are not necessarily piece-wise linear.

2. Unlike previous work [101], CROWN allows flexible selections of upper/lower bounds for activation functions, enabling an *adaptive* scheme that helps to reduce approximation errors. Our experiments show that CROWN achieves up to 26% improvements in certified local robustness radius compared to [101].

3. Our algorithm is efficient and can scale to large NNs with various activation functions. For a NN with over 10,000 neurons, we can give a certified local robustness radius in about 1 minute on 1 CPU core.

4.1.2 Related Work

For ReLU networks, finding the minimum adversarial perturbation for a given input data point $\mathbf{x_0}$ can be cast as a mixed integer linear programming (MILP) problem [70, 17, 33]. Reluplex [48, 12] uses a satisfiable modulo theory (SMT) to encode ReLU activations into linear constraints. Similarly, Planet [27] uses satisfiability (SAT) solvers. However, due to the NP-completeness for solving such a problem [48], these methods can only find minimum perturbation for very small networks. It can take Reluplex several hours to find the minimum perturbation of an example for a ReLU network with 5 inputs, 5 outputs and 300 neurons [48].

On the other hand, a computationally feasible alternative of local robustness radius certificate is to provide a non-trivial and *certified robustness lower bound* of minimum perturbation. Some analytical lower bounds based on operator norms on the weight matrices [94] or the Jacobian matrix in NNs [80] do not exploit special property of ReLU and thus can be very loose [101]. The bounds in [41, 102] are based on the local Lipschitz constant. [41] assumes a continuous differentiable NN and hence excludes ReLU networks; a closed form lower-bound is also hard to derive for networks beyond 2 layers. The CLEVER score [102] applies to ReLU networks and uses Extreme Value Theory to provide an estimated lower bound. Although the CLEVER score is capable of reflecting the level of robustness in different NNs and is scalable to large networks, it is not a certified robustness lower bound. On the other hand, [52] use the idea of a convex outer adversarial polytope in ReLU networks to compute a certified robustness lower bound by relaxing the MILP certification problem to linear programing (LP). [101] exploit the ReLU property to bound the activation function (or the local Lipschitz constant) and provide efficient algorithms (Fast-Lin and Fast-Lip) for computing a certified local robustness lower bound, achieving state-of-the-art performance. A recent work [36] uses abstract transformations to zonotopes for proving robustness property for ReLU networks. Nonetheless, there are still some applications demand non-ReLU activations, e.g. RNN and LSTM, thus a general framework that can efficiently compute non-trivial and certified local robustness lower bounds for NNs

Table 4.1: Comparison of methods for providing local adversarial robustness radius certification in NNs.

Method	Non-trivial bound	Multi-layer	Scalability	Beyond ReLU
Szegedy et al. [94]	×	✓	✓	✓
Reluplex [48], Planet [27]	✓	✓	×	×
Hein & Andriushchenko [41]	✓	×	✓	differentiable*
Raghunathan et al. [82]	✓	×	×	✓
Kolter and Wong [52]	✓	✓	✓	×
Fast-lin / Fast-lip [101]	✓	✓	✓	×
CROWN (This Chapter)	✓	✓	✓	✓ (general)

* Continuously differentiable activation function required (soft-plus is demonstrated in [41])

with general activation functions is of great importance. We aim at filling this gap and propose CROWN that can perform efficient robustness certification to NNs with general activation functions. Table 4.1 summarizes the differences of other approaches and CROWN. Note that the semidefinite programming approach proposed in [82] and a recent work [26] based on solving Lagrangian dual can both handle general activation functions, but [82] is limited to NNs with one hidden layer and [26] trades off the quality of robustness bound with scalability.

Some recent works (such as robust optimization based adversarial training [71] or region-based classification [11]) *empirically* exhibited strong robustness properties against several adversarial attacks, which is beyond the scope of *provable* local robustness radius certification. In addition, Sinha et al. [91] provided distributional local robustness level certification based on Wasserstein distance between data distributions, which is different from the local ℓ_p ball robustness model considered in this thesis.

4.2 CROWN: A General Framework for Certifying the Local Robustness Radius of Neural Networks

In this section, we present a general framework CROWN for efficiently computing a certified lower bound of minimum adversarial distortion given any input data point x_0 with general activation functions in larger NNs. We first provide principles in Section 4.2.1 to derive output bounds of NNs when the inputs are perturbed within an ℓ_p ball and each neuron has different (adaptive) linear approximation bounds on its activation function. The proofs of our certification framework are presented in Section 4.2.2 and 4.2.3. Next, in Section 4.2.4, we demonstrate how to provide robustness certification for four widely-used activation functions (ReLU, tanh, sigmoid and arctan) using CROWN. In particular, we show that the state-of-the-art Fast-Lin algorithm [101] is a special case under the CROWN framework and that the adaptive selections of approximation bounds allow CROWN to achieve a tighter (larger) certified lower bound (see Experimental results in Section 4.3). Finally, in Section 4.2.5, we further highlight the flexibility of CROWN to incorporate quadratic approximations on the activation functions in addition to the linear approximations described in Section 4.2.1.

4.2.1 General framework

Input perturbation and pre-activation bounds. Let $\mathbf{x_0} \in \mathbb{R}^{n_0}$ be a given data point, and let the perturbed input vector $\mathbf{x}$ be within an ϵ-bounded ℓ_p-ball centered at $\mathbf{x_0}$, i.e., $\mathbf{x} \in B_p(\mathbf{x_0}, \epsilon)$, where $B_p(\mathbf{x_0}, \epsilon) := \{\mathbf{x} \mid \|\mathbf{x} - \mathbf{x_0}\|_p \leq \epsilon\}$. For the r-th neuron in k-th layer, let its *pre-activation* input be $\mathbf{y}_r^{(k)}$, where $\mathbf{y}_r^{(k)} = \mathbf{W}_{r,:}^{(k)} \Phi^{k-1}(\mathbf{x}) + \mathbf{b}_r^{(k)}$ and $\mathbf{W}_{r,:}^{(k)}$ denotes the r-th row of matrix $\mathbf{W}^{(k)}$. When $\mathbf{x_0}$ is perturbed within an ϵ-bounded ℓ_p-ball, let $\mathbf{l}_r^{(k)}, \mathbf{u}_r^{(k)} \in \mathbb{R}$ be the pre-activation lower bound and upper bound of $\mathbf{y}_r^{(k)}$, i.e. $\mathbf{l}_r^{(k)} \leq \mathbf{y}_r^{(k)} \leq \mathbf{u}_r^{(k)}$.

Below, we first define the linear upper bounds and lower bounds of activation functions in Definition 4.2.1, which are the key to derive explicit output bounds for

an m-layer neural network with general activation functions. The formal statement of the explicit output bounds is shown in Theorem 4.2.2.

Definition 4.2.1 (Linear bounds on activation function)**.** *For the r-th neuron in k-th layer with pre-activation bounds $\mathbf{l}_r^{(k)}$,$\mathbf{u}_r^{(k)}$ and the activation function $\sigma(y)$, define two linear functions $h_{U,r}^{(k)}, h_{L,r}^{(k)} : \mathbb{R} \to \mathbb{R}$, $h_{U,r}^{(k)}(y) = \alpha_{U,r}^{(k)}(y + \beta_{U,r}^{(k)})$, $h_{L,r}^{(k)}(y) = \alpha_{L,r}^{(k)}(y + \beta_{L,r}^{(k)})$, such that $h_{L,r}^{(k)}(y) \le \sigma(y) \le h_{U,r}^{(k)}(y)$, $y \in [\mathbf{l}_r^{(k)}, \mathbf{u}_r^{(k)}]$, $\forall k \in [m-1], r \in [n_k]$ and $\alpha_{U,r}^{(k)}, \alpha_{L,r}^{(k)} \in \mathbb{R}^+, \beta_{U,r}^{(k)}, \beta_{L,r}^{(k)} \in \mathbb{R}$.*

Note that the parameters $\alpha_{U,r}^{(k)}, \alpha_{L,r}^{(k)}, \beta_{U,r}^{(k)}, \beta_{L,r}^{(k)}$ depend on $\mathbf{l}_r^{(k)}$ and $\mathbf{u}_r^{(k)}$, i.e. for different $\mathbf{l}_r^{(k)}$ and $\mathbf{u}_r^{(k)}$ we may choose different parameters. Also, for ease of exposition, here we restrict $\alpha_{U,r}^{(k)}, \alpha_{L,r}^{(k)} \ge 0$. However, Theorem 4.2.2 can be easily generalized to the case of negative $\alpha_{U,r}^{(k)}, \alpha_{L,r}^{(k)}$.

Theorem 4.2.2 (Explicit output bounds of neural network f)**.** *Given an m-layer neural network function $f : \mathbb{R}^{n_0} \to \mathbb{R}^{n_m}$, there exists two explicit functions $f_j^L : \mathbb{R}^{n_0} \to \mathbb{R}$ and $f_j^U : \mathbb{R}^{n_0} \to \mathbb{R}$ such that $\forall j \in [n_m]$, $\forall \mathbf{x} \in B_p(\mathbf{x_0}, \epsilon)$, the inequality $f_j^L(\mathbf{x}) \le f_j(\mathbf{x}) \le f_j^U(\mathbf{x})$ holds true, where*

$$f_j^U(\mathbf{x}) = \mathbf{\Lambda}_{j,:}^{(0)}\mathbf{x} + \sum_{k=1}^{m} \mathbf{\Lambda}_{j,:}^{(k)}(\mathbf{b}^{(k)} + \mathbf{\Delta}_{:,j}^{(k)}), \quad f_j^L(\mathbf{x}) = \mathbf{\Omega}_{j,:}^{(0)}\mathbf{x} + \sum_{k=1}^{m} \mathbf{\Omega}_{j,:}^{(k)}(\mathbf{b}^{(k)} + \mathbf{\Theta}_{:,j}^{(k)}), \tag{4.1}$$

$$\mathbf{\Lambda}_{j,:}^{(k-1)} = \begin{cases} \mathbf{e}_j^\top & \text{if } k = m+1; \\ (\mathbf{\Lambda}_{j,:}^{(k)}\mathbf{W}^{(k)}) \odot \lambda_{j,:}^{(k-1)} & \text{if } k \in [m]. \end{cases} \quad \mathbf{\Omega}_{j,:}^{(k-1)} = \begin{cases} \mathbf{e}_j^\top & \text{if } k = m+1; \\ (\mathbf{\Omega}_{j,:}^{(k)}\mathbf{W}^{(k)}) \odot \omega_{j,:}^{(k-1)} & \text{if } k \in [m]. \end{cases}$$

and $\forall i \in [n_k]$, we define four matrices $\lambda^{(k)}, \omega^{(k)}, \mathbf{\Delta}^{(k)}, \mathbf{\Theta}^{(k)} \in \mathbb{R}^{n_m \times n_k}$:

$$\lambda_{j,i}^{(k)} = \begin{cases} \alpha_{U,i}^{(k)} & \text{if } k \neq 0,\ \mathbf{\Lambda}_{j,:}^{(k+1)}\mathbf{W}_{:,i}^{(k+1)} \ge 0; \\ \alpha_{L,i}^{(k)} & \text{if } k \neq 0,\ \mathbf{\Lambda}_{j,:}^{(k+1)}\mathbf{W}_{:,i}^{(k+1)} < 0; \\ 1 & \text{if } k = 0. \end{cases} \quad \omega_{j,i}^{(k)} = \begin{cases} \alpha_{L,i}^{(k)} & \text{if } k \neq 0,\ \mathbf{\Omega}_{j,:}^{(k+1)}\mathbf{W}_{:,i}^{(k+1)} \ge 0; \\ \alpha_{U,i}^{(k)} & \text{if } k \neq 0,\ \mathbf{\Omega}_{j,:}^{(k+1)}\mathbf{W}_{:,i}^{(k+1)} < 0; \\ 1 & \text{if } k = 0. \end{cases}$$

$$\boldsymbol{\Delta}_{i,j}^{(k)} = \begin{cases} \beta_{U,i}^{(k)} & \text{if } k \neq m,\ \boldsymbol{\Lambda}_{j,:}^{(k+1)} \mathbf{W}_{:,i}^{(k+1)} \geq 0; \\ \beta_{L,i}^{(k)} & \text{if } k \neq m,\ \boldsymbol{\Lambda}_{j,:}^{(k+1)} \mathbf{W}_{:,i}^{(k+1)} < 0; \\ 0 & \text{if } k = m. \end{cases} \quad \boldsymbol{\Theta}_{i,j}^{(k)} = \begin{cases} \beta_{L,i}^{(k)} & \text{if } k \neq m,\ \boldsymbol{\Omega}_{j,:}^{(k+1)} \mathbf{W}_{:,i}^{(k+1)} \geq 0; \\ \beta_{U,i}^{(k)} & \text{if } k \neq m,\ \boldsymbol{\Omega}_{j,:}^{(k+1)} \mathbf{W}_{:,i}^{(k+1)} < 0; \\ 0 & \text{if } k = m. \end{cases}$$

and $\odot$ *is the Hadamard product and* $\mathbf{e}_j \in \mathbb{R}^{n_m}$ *is a standard unit vector at* j*th coordinate.*

Theorem 4.2.2 illustrates how the output of a NN function $f_j(\mathbf{x})$ can be bounded by two linear functions $f_j^U(\mathbf{x})$ and $f_j^L(\mathbf{x})$ when the activation function of each neuron is bounded by two linear functions $h_{U,r}^{(k)}$ and $h_{L,r}^{(k)}$ in Definition 4.2.1. The central idea is to unwrap the activation functions layer by layer by considering the signs of the associated (equivalent) weights of each neuron and apply the two linear bounds $h_{U,r}^{(k)}$ and $h_{L,r}^{(k)}$. As we demonstrate in the proof in Section 4.2.2, when we replace the activation functions with the corresponding linear upper bounds and lower bounds at the layer $m-1$, we can then define equivalent weights and biases based on the parameters of $h_{U,r}^{(m-1)}$ and $h_{L,r}^{(m-1)}$ (e.g. $\boldsymbol{\Lambda}^{(k)}, \boldsymbol{\Delta}^{(k)}, \boldsymbol{\Omega}^{(k)}, \boldsymbol{\Theta}^{(k)}$ are related to the terms $\alpha_{U,r}^{(k)}, \beta_{U,r}^{(k)}, \alpha_{L,r}^{(k)}, \beta_{L,r}^{(k)}$, respectively) and then repeat the procedure to "back-propagate" to the input layer. This allows us to obtain $f_j^U(\mathbf{x})$ and $f_j^L(\mathbf{x})$ in (4.1). The formal proof of Theorem 4.2.2 is in Section 4.2.2. Note that for a neuron r in layer k, the slopes of its linear upper and lower bounds $\alpha_{U,r}^{(k)}, \alpha_{L,r}^{(k)}$ can be different. This implies:

1. Fast-Lin [101] is a special case of our framework as they require the slopes $\alpha_{U,r}^{(k)}, \alpha_{L,r}^{(k)}$ to be the same; and it only applies to ReLU networks (cf. Sec. 4.2.4). In Fast-Lin, $\boldsymbol{\Lambda}^{(0)}$ and $\boldsymbol{\Omega}^{(0)}$ are identical.

2. Our CROWN framework allows *adaptive selections* on the linear approximation when computing certified lower bounds of minimum adversarial distortion, which is the main contributor to improve the certified lower bound as demonstrated in the experiments in Section 4.3.

Global bounds. More importantly, since the input $\mathbf{x} \in B_p(\mathbf{x_0}, \epsilon)$, we can take the maximum, i.e. $\max_{\mathbf{x} \in B_p(\mathbf{x_0}, \epsilon)} f_j^U(\mathbf{x})$, and minimum, i.e. $\min_{\mathbf{x} \in B_p(\mathbf{x_0}, \epsilon)} f_j^L(\mathbf{x})$, as a pair of global upper and lower bound of $f_j(\mathbf{x})$ – which in fact has *closed-form* solutions

because $f_j^U(\mathbf{x})$ and $f_j^L(\mathbf{x})$ are two linear functions and $\mathbf{x} \in B_p(\mathbf{x_0}, \epsilon)$ is a convex norm constraint. This result is formally presented below and its proof is given in Section 4.2.3.

Corollary 4.2.3 (Closed-form global bounds)**.** *Given a data point* $\mathbf{x_0} \in \mathbb{R}^{n_0}$*,* ℓ_p *ball parameters* $p \geq 1$ *and* $\epsilon > 0$*. For an* m*-layer neural network function* $f : \mathbb{R}^{n_0} \to \mathbb{R}^{n_m}$*, there exists two fixed values* γ_j^L *and* γ_j^U *such that* $\forall \mathbf{x} \in B_p(\mathbf{x_0}, \epsilon)$ *and* $\forall j \in [n_m]$*,* $1/q = 1 - 1/p$*, the inequality* $\gamma_j^L \leq f_j(\mathbf{x}) \leq \gamma_j^U$ *holds true, where*

$$\begin{aligned} \gamma_j^U &= \epsilon \|\mathbf{\Lambda}_{j,:}^{(0)}\|_q + \mathbf{\Lambda}_{j,:}^{(0)} \mathbf{x_0} + \sum_{k=1}^{m} \mathbf{\Lambda}_{j,:}^{(k)} (\mathbf{b}^{(k)} + \mathbf{\Delta}_{:,j}^{(k)}), \\ \gamma_j^L &= -\epsilon \|\mathbf{\Omega}_{j,:}^{(0)}\|_q + \mathbf{\Omega}_{j,:}^{(0)} \mathbf{x_0} + \sum_{k=1}^{m} \mathbf{\Omega}_{j,:}^{(k)} (\mathbf{b}^{(k)} + \mathbf{\Theta}_{:,j}^{(k)}). \end{aligned} \tag{4.2}$$

Certified lower bound of minimum distortion. Given an input example $\mathbf{x_0}$ and an m-layer NN, let c be the predicted class of $\mathbf{x_0}$ and $t \neq c$ be the targeted attack class. We aim to use the uniform bounds established in Corollary 4.2.3 to obtain the largest possible lower bound $r_{p,L,t}$ and $r_{p,L}$ of targeted and untargeted attacks respectively, which can be formulated as follows:

$$r_{p,L,t} = \max_{\epsilon} \epsilon \text{ s.t. } \gamma_c^L(\epsilon) - \gamma_t^U(\epsilon) > 0 \quad \text{and} \quad r_{p,L} = \min_{t \neq c} r_{p,L,t}.$$

We note that although there is a linear ϵ term in (4.2), other terms such as $\mathbf{\Lambda}^{(k)}, \mathbf{\Delta}^{(k)}$ and $\mathbf{\Omega}^{(k)}, \mathbf{\Theta}^{(k)}$ also implicitly depend on ϵ. This is because the parameters $\alpha_{U,i}^{(k)}, \beta_{U,i}^{(k)}$, $\alpha_{L,i}^{(k)}, \beta_{L,i}^{(k)}$ depend on $\mathbf{l}_i^{(k)}, \mathbf{u}_i^{(k)}$, which may vary with ϵ; thus the values in $\mathbf{\Lambda}^{(k)}, \mathbf{\Delta}^{(k)}$, $\mathbf{\Omega}^{(k)}, \mathbf{\Theta}^{(k)}$ depend on ϵ. It is therefore difficult to obtain an explicit expression of $\gamma_c^L(\epsilon) - \gamma_t^U(\epsilon)$ in terms of ϵ. Fortunately, we can still perform a binary search to obtain $r_{p,L,t}$ with Corollary 4.2.3. More precisely, we first initialize ϵ at some fixed positive value and apply Corollary 4.2.3 repeatedly to obtain $\mathbf{l}_r^{(k)}$ and $\mathbf{u}_r^{(k)}$ from $k = 1$ to m and $r \in [n_k]$. We then check if the condition $\gamma_c^L - \gamma_t^U > 0$ is satisfied. If so, we increase ϵ; otherwise, we decrease ϵ; and we repeat the procedure until a given tolerance level is

met.[2]

Time Complexity. With Corollary 4.2.3, we can compute analytic output bounds efficiently without resorting to any optimization solvers for general ℓ_p distortion, and the time complexity for an m-layer ReLU network is polynomial time in contrast to Reluplex or Mixed-Integer Optimization-based approach [17, 33] where SMT and MIO solvers are exponential-time. For an m layer network with n neurons per layer and n outputs, time complexity of CROWN is $O(m^2n^3)$. Forming $\mathbf{\Lambda}^{(0)}$ and $\mathbf{\Omega}^{(0)}$ for the m-th layer involves multiplications of layer weights in a similar cost of forward propagation in $O(mn^3)$ time. Also, the bounds for all previous $k \in [m-1]$ layers need to be computed beforehand in $O(kn^3)$ time; thus the total time complexity is $O(m^2n^3)$.

[2] The bound can be further improved by considering $g(\mathbf{x}) := f_c(\mathbf{x}) - f_t(\mathbf{x})$ and replacing the last layer's weights by $\mathbf{W}^{(m)}_{c,:} - \mathbf{W}^{(m)}_{t,:}$. This is also used by [101].

4.2.2 Proof of Theorem 4.2.2

Given an m-layer neural network function $f : \mathbb{R}^{n_0} \to \mathbb{R}^{n_m}$ with pre-activation bounds $\mathbf{l}^{(k)}$ and $\mathbf{u}^{(k)}$ for $\mathbf{x} \in B_p(\mathbf{x_0}, \epsilon)$ and $\forall k \in [m-1]$, let the pre-activation inputs for the i-th neuron at layer $m-1$ be $\mathbf{y}_i^{(m-1)} := \mathbf{W}_{i,:}^{(m-1)} \Phi^{m-2}(\mathbf{x}) + \mathbf{b}_i^{(m-1)}$. The j-th output of the neural network is the following:

$$
\begin{aligned}
f_j(\mathbf{x}) &= \sum_{i=1}^{n_{m-1}} \mathbf{W}_{j,i}^{(m)} [\Phi^{m-1}(\mathbf{x})]_i + \mathbf{b}_j^{(m)}, && (4.3)\\
&= \sum_{i=1}^{n_{m-1}} \mathbf{W}_{j,i}^{(m)} \sigma(\mathbf{y}_i^{(m-1)}) + \mathbf{b}_j^{(m)}, \\
&= \underbrace{\sum_{\mathbf{W}_{j,i}^{(m)} \geq 0} \mathbf{W}_{j,i}^{(m)} \sigma(\mathbf{y}_i^{(m-1)})}_{F_1} + \underbrace{\sum_{\mathbf{W}_{j,i}^{(m)} < 0} \mathbf{W}_{j,i}^{(m)} \sigma(\mathbf{y}_i^{(m-1)})}_{F_2} + \mathbf{b}_j^{(m)}. && (4.4)
\end{aligned}
$$

Assume the activation function $\sigma(y)$ is bounded by two linear functions $h_{U,i}^{(m-1)}, h_{L,i}^{(m-1)}$ in Definition 4.2.1, we have

$$
h_{L,i}^{(m-1)}(\mathbf{y}_i^{(m-1)}) \leq \sigma(\mathbf{y}_i^{(m-1)}) \leq h_{U,i}^{(m-1)}(\mathbf{y}_i^{(m-1)}).
$$

Thus, if the associated weight $\mathbf{W}_{j,i}^{(m)}$ to the i-th neuron is non-negative (the terms in F_1 bracket), we have

$$
\mathbf{W}_{j,i}^{(m)} \cdot h_{L,i}^{(m-1)}(\mathbf{y}_i^{(m-1)}) \leq \mathbf{W}_{j,i}^{(m)} \sigma(\mathbf{y}_i^{(m-1)}) \leq \mathbf{W}_{j,i}^{(m)} \cdot h_{U,i}^{(m-1)}(\mathbf{y}_i^{(m-1)}); \tag{4.5}
$$

otherwise (the terms in F_2 bracket), we have

$$
\mathbf{W}_{j,i}^{(m)} \cdot h_{U,i}^{(m-1)}(\mathbf{y}_i^{(m-1)}) \leq \mathbf{W}_{j,i}^{(m)} \sigma(\mathbf{y}_i^{(m-1)}) \leq \mathbf{W}_{j,i}^{(m)} \cdot h_{L,i}^{(m-1)}(\mathbf{y}_i^{(m-1)}). \tag{4.6}
$$

Upper bound. Let $f_j^{U,m-1}(\mathbf{x})$ be an upper bound of $f_j(\mathbf{x})$. To compute $f_j^{U,m-1}(\mathbf{x})$, (4.4), (4.5) and (4.6) are the key equations. Precisely, for the $\mathbf{W}_{j,i}^{(m)} \geq 0$ terms in (4.4), the upper bound is the right-hand-side (RHS) in (4.5); and for the $\mathbf{W}_{j,i}^{(m)} < 0$ terms

in (4.4), the upper bound is the RHS in (4.6). Thus, we obtain:

$$
\begin{aligned}
& f_j^{U,m-1}(\mathbf{x}) \\
&= \sum_{\mathbf{W}_{j,i}^{(m)} \geq 0} \mathbf{W}_{j,i}^{(m)} \cdot h_{U,i}^{(m-1)}(\mathbf{y}_i^{(m-1)}) + \sum_{\mathbf{W}_{j,i}^{(m)} < 0} \mathbf{W}_{j,i}^{(m)} \cdot h_{L,i}^{(m-1)}(\mathbf{y}_i^{(m-1)}) + \mathbf{b}_j^{(m)}, && (4.7) \\
&= \sum_{\mathbf{W}_{j,i}^{(m)} \geq 0} \mathbf{W}_{j,i}^{(m)} \alpha_{U,i}^{(m-1)}(\mathbf{y}_i^{(m-1)} + \beta_{U,i}^{(m-1)}) + \sum_{\mathbf{W}_{j,i}^{(m)} < 0} \mathbf{W}_{j,i}^{(m)} \alpha_{L,i}^{(m-1)}(\mathbf{y}_i^{(m-1)} + \beta_{L,i}^{(m-1)}) + \mathbf{b}_j^{(m)}, && (4.8) \\
&= \sum_{i=1}^{n_{m-1}} \mathbf{W}_{j,i}^{(m)} \lambda_{j,i}^{(m-1)}(\mathbf{y}_i^{(m-1)} + \mathbf{\Delta}_{i,j}^{(m-1)}) + \mathbf{b}_j^{(m)}, && (4.9) \\
&= \sum_{i=1}^{n_{m-1}} \mathbf{\Lambda}_{j,i}^{(m-1)}(\sum_{r=1}^{n_{m-2}} \mathbf{W}_{i,r}^{(m-1)}[\Phi^{m-2}(\mathbf{x})]_r + \mathbf{b}_i^{(m-1)} + \mathbf{\Delta}_{i,j}^{(m-1)}) + \mathbf{b}_j^{(m)}, && (4.10) \\
&= \sum_{i=1}^{n_{m-1}} \mathbf{\Lambda}_{j,i}^{(m-1)}(\sum_{r=1}^{n_{m-2}} \mathbf{W}_{i,r}^{(m-1)}[\Phi^{m-2}(\mathbf{x})]_r) + \sum_{i=1}^{n_{m-1}} \mathbf{\Lambda}_{j,i}^{(m-1)}(\mathbf{b}_i^{(m-1)} + \mathbf{\Delta}_{i,j}^{(m-1)}) + \mathbf{b}_j^{(m)}, && (4.11) \\
&= \sum_{r=1}^{n_{m-2}} \left(\sum_{i=1}^{n_{m-1}} \mathbf{\Lambda}_{j,i}^{(m-1)} \mathbf{W}_{i,r}^{(m-1)} \right) [\Phi^{m-2}(\mathbf{x})]_r + \left(\sum_{i=1}^{n_{m-1}} \mathbf{\Lambda}_{j,i}^{(m-1)}(\mathbf{b}_i^{(m-1)} + \mathbf{\Delta}_{i,j}^{(m-1)}) + \mathbf{b}_j^{(m)} \right), && (4.12) \\
&= \sum_{r=1}^{n_{m-2}} \tilde{\mathbf{W}}_{j,r}^{(m-1)}[\Phi^{m-2}(\mathbf{x})]_r + \tilde{\mathbf{b}}_j^{(m-1)}. && (4.13)
\end{aligned}
$$

From (4.7) to (4.8), we replace $h_{U,i}^{(m-1)}(\mathbf{y}_i^{(m-1)})$ and $h_{L,i}^{(m-1)}(\mathbf{y}_i^{(m-1)})$ by their definitions; from (4.8) to (4.9), we use variables $\lambda_{j,i}^{(m-1)}$ and $\mathbf{\Delta}_{j,i}^{(m-1)}$ to denote the slopes in front of $\mathbf{y}_i^{(m-1)}$ and the intercepts in the parentheses:

$$
\lambda_{j,i}^{(m-1)} = \begin{cases} \alpha_{U,i}^{(m-1)} & \text{if } \mathbf{W}_{j,i}^{(m)} \geq 0 \quad (\iff \mathbf{\Lambda}_{j,:}^{(m)} \mathbf{W}_{:,i}^{(m)} \geq 0); \\ \alpha_{L,i}^{(m-1)} & \text{if } \mathbf{W}_{j,i}^{(m)} < 0 \quad (\iff \mathbf{\Lambda}_{j,:}^{(m)} \mathbf{W}_{:,i}^{(m)} < 0); \end{cases} \tag{4.14}
$$

$$
\mathbf{\Delta}_{i,j}^{(m-1)} = \begin{cases} \beta_{U,i}^{(m-1)} & \text{if } \mathbf{W}_{j,i}^{(m)} \geq 0 \quad (\iff \mathbf{\Lambda}_{j,:}^{(m)} \mathbf{W}_{:,i}^{(m)} \geq 0); \\ \beta_{L,i}^{(m-1)} & \text{if } \mathbf{W}_{j,i}^{(m)} < 0 \quad (\iff \mathbf{\Lambda}_{j,:}^{(m)} \mathbf{W}_{:,i}^{(m)} < 0). \end{cases} \tag{4.15}
$$

From (4.9) to (4.10), we replace $\mathbf{y}_i^{(m-1)}$ with its definition and let $\mathbf{\Lambda}_{j,i}^{(m-1)} \coloneqq \mathbf{W}_{j,i}^{(m)} \lambda_{j,i}^{(m-1)}$. We further let $\mathbf{\Lambda}_{j,:}^{(m)} = \mathbf{e}_j^\top$ (the standard unit vector with the only non-zero jth element equal to 1), and thus we can rewrite the conditions of $\mathbf{W}_{j,i}^{(m)}$ in (4.14) and (4.15) as $\mathbf{\Lambda}_{j,:}^{(m)} \mathbf{W}_{:,i}^{(m)}$. From (4.10) to (4.11), we collect the constant terms that are not related to $\mathbf{x}$. From (4.11) to (4.12), we swap the summation order of i and r, and the coefficients in front of $[\Phi^{m-2}(x)]_r$ can be combined into a new equivalent weight $\tilde{\mathbf{W}}_{j,r}^{(m-1)}$ and the constant term can combined into a new equivalent bias $\tilde{\mathbf{b}}_j^{(m-1)}$ in (4.13):

$$
\begin{aligned}
\tilde{\mathbf{W}}_{j,r}^{(m-1)} &= \sum_{i=1}^{n_{m-1}} \mathbf{\Lambda}_{j,i}^{(m-1)} \mathbf{W}_{i,r}^{(m-1)} = \mathbf{\Lambda}_{j,:}^{(m-1)} \mathbf{W}_{:,r}^{(m-1)}, \\
\tilde{\mathbf{b}}_j^{(m-1)} &= \sum_{i=1}^{n_{m-1}} \mathbf{\Lambda}_{j,i}^{(m-1)} (\mathbf{b}_i^{(m-1)} + \mathbf{\Delta}_{i,j}^{(m-1)}) + \mathbf{b}_j^{(m)} = \mathbf{\Lambda}_{j,:}^{(m-1)} (\mathbf{b}^{(m-1)} + \mathbf{\Delta}_{:,j}^{(m-1)}) + \mathbf{b}_j^{(m)}.
\end{aligned}
$$

Notice that after defining the new equivalent weight $\tilde{\mathbf{W}}_{j,r}^{(m-1)}$ and equivalent bias $\tilde{\mathbf{b}}_j^{(m-1)}$, $f_j^{U,m-1}(\mathbf{x})$ in (4.13) and $f_j(\mathbf{x})$ in (4.3) are in the same form. Thus, we can repeat the above procedure again to obtain an upper bound of $f_j^{U,m-1}(\mathbf{x})$, i.e. $f_j^{U,m-2}(\mathbf{x})$:

$$
\begin{aligned}
\mathbf{\Lambda}_{j,i}^{(m-2)} &= \tilde{\mathbf{W}}_{j,i}^{(m-1)} \lambda_{j,i}^{(m-2)} \\
&= \mathbf{\Lambda}_{j,:}^{(m-1)} \mathbf{W}_{:,i}^{(m-1)} \lambda_{j,i}^{(m-2)} \\
\tilde{\mathbf{W}}_{j,r}^{(m-2)} &= \mathbf{\Lambda}_{j,:}^{(m-2)} \mathbf{W}_{:,r}^{(m-2)} \\
\tilde{\mathbf{b}}_j^{(m-2)} &= \mathbf{\Lambda}_{j,:}^{(m-2)} (\mathbf{b}^{(m-2)} + \mathbf{\Delta}_{:,j}^{(m-2)}) + \tilde{\mathbf{b}}_j^{(m-1)}
\end{aligned}
$$

$$
\lambda_{j,i}^{(m-2)} = \begin{cases} \alpha_{U,i}^{(m-2)} & \text{if } \tilde{\mathbf{W}}_{j,i}^{(m-1)} \geq 0 \quad (\iff \mathbf{\Lambda}_{j,:}^{(m-1)} \mathbf{W}_{:,i}^{(m-1)} \geq 0); \\ \alpha_{L,i}^{(m-2)} & \text{if } \tilde{\mathbf{W}}_{j,i}^{(m-1)} < 0 \quad (\iff \mathbf{\Lambda}_{j,:}^{(m-1)} \mathbf{W}_{:,i}^{(m-1)} < 0); \end{cases}
$$

$$
\mathbf{\Delta}_{i,j}^{(m-2)} = \begin{cases} \beta_{U,i}^{(m-2)} & \text{if } \tilde{\mathbf{W}}_{j,i}^{(m-1)} \geq 0 \quad (\iff \mathbf{\Lambda}_{j,:}^{(m-1)} \mathbf{W}_{:,i}^{(m-1)} \geq 0); \\ \beta_{L,i}^{(m-2)} & \text{if } \tilde{\mathbf{W}}_{j,i}^{(m-1)} < 0 \quad (\iff \mathbf{\Lambda}_{j,:}^{(m-1)} \mathbf{W}_{:,i}^{(m-1)} < 0). \end{cases}
$$

and repeat again iteratively until obtain the final upper bound $f_j^{U,1}(\mathbf{x})$, where $f_j(\mathbf{x}) \leq f_j^{U,m-1}(\mathbf{x}) \leq f_j^{U,m-2}(\mathbf{x}) \leq \ldots \leq f_j^{U,1}(\mathbf{x})$. We let $f_j(\mathbf{x})$ denote the final upper bound

$f_j^{U,1}(\mathbf{x})$, and we have

$$f_j^U(\mathbf{x}) = \mathbf{\Lambda}_{j,:}^{(0)}\mathbf{x} + \sum_{k=1}^{m} \mathbf{\Lambda}_{j,:}^{(k)}(\mathbf{b}^{(k)} + \mathbf{\Delta}_{:,j}^{(k)})$$

and ($\odot$ is the Hadamard product)

$$\mathbf{\Lambda}_{j,:}^{(k-1)} = \begin{cases} \mathbf{e}_j^\top & \text{if } k = m+1; \\ (\mathbf{\Lambda}_{j,:}^{(k)}\mathbf{W}^{(k)}) \odot \lambda_{j,:}^{(k-1)} & \text{if } k \in [m]. \end{cases}$$

and $\forall i \in [n_k]$,

$$\lambda_{j,i}^{(k)} = \begin{cases} \alpha_{U,i}^{(k)} & \text{if } k \in [m-1],\ \mathbf{\Lambda}_{j,:}^{(k+1)}\mathbf{W}_{:,i}^{(k+1)} \geq 0; \\ \alpha_{L,i}^{(k)} & \text{if } k \in [m-1],\ \mathbf{\Lambda}_{j,:}^{(k+1)}\mathbf{W}_{:,i}^{(k+1)} < 0; \\ 1 & \text{if } k = 0. \end{cases}$$

$$\mathbf{\Delta}_{i,j}^{(k)} = \begin{cases} \beta_{U,i}^{(k)} & \text{if } k \in [m-1],\ \mathbf{\Lambda}_{j,:}^{(k+1)}\mathbf{W}_{:,i}^{(k+1)} \geq 0; \\ \beta_{L,i}^{(k)} & \text{if } k \in [m-1],\ \mathbf{\Lambda}_{j,:}^{(k+1)}\mathbf{W}_{:,i}^{(k+1)} < 0; \\ 0 & \text{if } k = m. \end{cases}$$

Lower bound. The above derivations of upper bound can be applied similarly to deriving lower bounds of $f_j(\mathbf{x})$, and the only difference is now we need to use the LHS of (4.5) and (4.6) (rather than RHS when deriving upper bound) to bound the two terms in (4.4). Thus, following the same procedure in deriving the upper bounds, we can iteratively unwrap the activation functions and obtain a final lower bound $f_j^{L,1}(\mathbf{x})$, where $f_j(\mathbf{x}) \geq f_j^{L,m-1}(\mathbf{x}) \geq f_j^{L,m-2}(\mathbf{x}) \geq \ldots \geq f_j^{L,1}(\mathbf{x})$. Let $f_j^L(\mathbf{x}) = f_j^{L,1}(\mathbf{x})$, we have:

$$f_j^L(\mathbf{x}) = \mathbf{\Omega}_{j,:}^{(0)}\mathbf{x} + \sum_{k=1}^{m} \mathbf{\Omega}_{j,:}^{(k)}(\mathbf{b}^{(k)} + \mathbf{\Theta}_{:,j}^{(k)})$$

$$\mathbf{\Omega}_{j,:}^{(k-1)} = \begin{cases} \mathbf{e}_j^\top & \text{if } k = m+1; \\ (\mathbf{\Omega}_{j,:}^{(k)} \mathbf{W}^{(k)}) \odot \omega_{j,:}^{(k-1)} & \text{if } k \in [m]. \end{cases}$$

and $\forall i \in [n_k]$,

$$\omega_{j,i}^{(k)} = \begin{cases} \alpha_{L,i}^{(k)} & \text{if } k \in [m-1],\ \mathbf{\Omega}_{j,:}^{(k+1)} \mathbf{W}_{:,i}^{(k+1)} \geq 0; \\ \alpha_{U,i}^{(k)} & \text{if } k \in [m-1],\ \mathbf{\Omega}_{j,:}^{(k+1)} \mathbf{W}_{:,i}^{(k+1)} < 0; \\ 1 & \text{if } k = 0. \end{cases}$$

$$\mathbf{\Theta}_{i,j}^{(k)} = \begin{cases} \beta_{L,i}^{(k)} & \text{if } k \in [m-1],\ \mathbf{\Omega}_{j,:}^{(k+1)} \mathbf{W}_{:,i}^{(k+1)} \geq 0; \\ \beta_{U,i}^{(k)} & \text{if } k \in [m-1],\ \mathbf{\Omega}_{j,:}^{(k+1)} \mathbf{W}_{:,i}^{(k+1)} < 0; \\ 0 & \text{if } k = m. \end{cases}$$

Indeed, $\lambda_{j,i}^{(k)}$ and $\omega_{j,i}^{(k)}$ only differs in the conditions of selecting $\alpha_{U,i}^{(k)}$ or $\alpha_{L,i}^{(k)}$; similarly for $\mathbf{\Delta}_{i,j}^{(k)}$ and $\mathbf{\Theta}_{i,j}^{(k)}$.

4.2.3 Proof of Corollary 4.2.3

Definition 4.2.4 (Dual norm)**.** *Let $\|\cdot\|$ be a norm on $\mathbb{R}^n$. The associated dual norm, denoted as $\|\cdot\|_*$, is defined as*

$$\|\mathbf{a}\|_* = \{\sup_{\mathbf{y}} \mathbf{a}^\top \mathbf{y} \mid \|\mathbf{y}\| \leq 1\}.$$

Global upper bound. Our goal is to find a *global* upper and lower bound for the m-th layer network output $f_j(\mathbf{x}), \forall \mathbf{x} \in B_p(\mathbf{x_0}, \epsilon)$. By Theorem 4.2.2, for $\mathbf{x} \in B_p(\mathbf{x_0}, \epsilon)$, we have $f_j^L(\mathbf{x}) \leq f_j(\mathbf{x}) \leq f_j^U(\mathbf{x})$ and $f_j^U(\mathbf{x}) = \mathbf{\Lambda}_{j,:}^{(0)}\mathbf{x} + \sum_{k=1}^m \mathbf{\Lambda}_{j,:}^{(k)}(\mathbf{b}^{(k)} + \mathbf{\Delta}_{:,j}^{(k)})$. Thus define $\gamma_j^U := \max_{\mathbf{x}\in B_p(\mathbf{x_0},\epsilon)} f_j^U(\mathbf{x})$, and we have

$$f_j(\mathbf{x}) \leq f_j^U(\mathbf{x}) \leq \max_{\mathbf{x}\in B_p(\mathbf{x_0},\epsilon)} f_j^U(\mathbf{x}) = \gamma_j^U,$$

since $\forall \mathbf{x} \in B_p(\mathbf{x_0}, \epsilon)$. In particular,

$$\begin{aligned}
\max_{\mathbf{x}\in B_p(\mathbf{x_0},\epsilon)} f_j^U(\mathbf{x}) &= \max_{\mathbf{x}\in B_p(\mathbf{x_0},\epsilon)} \left[\mathbf{\Lambda}_{j,:}^{(0)}\mathbf{x} + \sum_{k=1}^m \mathbf{\Lambda}_{j,:}^{(k)}(\mathbf{b}^{(k)} + \mathbf{\Delta}_{:,j}^{(k)})\right] \\
&= \left[\max_{\mathbf{x}\in B_p(\mathbf{x_0},\epsilon)} \mathbf{\Lambda}_{j,:}^{(0)}\mathbf{x}\right] + \sum_{k=1}^m \mathbf{\Lambda}_{j,:}^{(k)}(\mathbf{b}^{(k)} + \mathbf{\Delta}_{:,j}^{(k)}) \qquad (4.16) \\
&= \epsilon\left[\max_{\mathbf{y}\in\mathbb{B}_p(\mathbf{0},1)} \mathbf{\Lambda}_{j,:}^{(0)}\mathbf{y}\right] + \mathbf{\Lambda}_{j,:}^{(0)}\mathbf{x_0} + \sum_{k=1}^m \mathbf{\Lambda}_{j,:}^{(k)}(\mathbf{b}^{(k)} + \mathbf{\Delta}_{:,j}^{(k)}) \qquad (4.17) \\
&= \epsilon\|\mathbf{\Lambda}_{j,:}^{(0)}\|_q + \mathbf{\Lambda}_{j,:}^{(0)}\mathbf{x_0} + \sum_{k=1}^m \mathbf{\Lambda}_{j,:}^{(k)}(\mathbf{b}^{(k)} + \mathbf{\Delta}_{:,j}^{(k)}). \qquad (4.18)
\end{aligned}$$

From (4.16) to (4.17), let $\mathbf{y} := \frac{\mathbf{x}-\mathbf{x_0}}{\epsilon}$, and thus $\mathbf{y} \in \mathbb{B}_p(\mathbf{0}, 1)$. From (4.17) to (4.18), apply Definition 4.2.4 and use the fact that ℓ_q norm is dual of ℓ_p norm for $p, q \in [1, \infty]$.

Global lower bound. Similarly, let $\gamma_j^L := \min_{\mathbf{x}\in B_p(\mathbf{x_0},\epsilon)} f_j^L(\mathbf{x})$, we have

$$f_j(\mathbf{x}) \geq f_j^L(\mathbf{x}) \geq \min_{\mathbf{x}\in B_p(\mathbf{x_0},\epsilon)} f_j^L(\mathbf{x}) = \gamma_j^L.$$

Since $f_j^L(\mathbf{x}) = \mathbf{\Omega}_{j,:}^{(0)}\mathbf{x} + \sum_{k=1}^{m} \mathbf{\Omega}_{j,:}^{(k)}(\mathbf{b}^{(k)} + \mathbf{\Theta}_{:,j}^{(k)})$, we can derive γ_j^L (similar to the derivation of γ_j^U) below:

$$\begin{aligned}
\min_{\mathbf{x}\in B_p(\mathbf{x_0},\epsilon)} f_j^L(\mathbf{x}) &= \min_{\mathbf{x}\in B_p(\mathbf{x_0},\epsilon)} \left[\mathbf{\Omega}_{j,:}^{(0)}\mathbf{x} + \sum_{k=1}^{m} \mathbf{\Omega}_{j,:}^{(k)}(\mathbf{b}^{(k)} + \mathbf{\Theta}_{:,j}^{(k)})\right] \\
&= \left[\min_{\mathbf{x}\in B_p(\mathbf{x_0},\epsilon)} \mathbf{\Omega}_{j,:}^{(0)}\mathbf{x}\right] + \sum_{k=1}^{m} \mathbf{\Omega}_{j,:}^{(k)}(\mathbf{b}^{(k)} + \mathbf{\Theta}_{:,j}^{(k)}) \\
&= -\epsilon\left[\max_{\mathbf{y}\in \mathbb{B}_p(\mathbf{0},1)} -\mathbf{\Omega}_{j,:}^{(0)}\mathbf{y}\right] + \mathbf{\Omega}_{j,:}^{(0)}\mathbf{x_0} + \sum_{k=1}^{m} \mathbf{\Omega}_{j,:}^{(k)}(\mathbf{b}^{(k)} + \mathbf{\Theta}_{:,j}^{(k)}) \\
&= -\epsilon\|\mathbf{\Omega}_{j,:}^{(0)}\|_q + \mathbf{\Omega}_{j,:}^{(0)}\mathbf{x_0} + \sum_{k=1}^{m} \mathbf{\Omega}_{j,:}^{(k)}(\mathbf{b}^{(k)} + \mathbf{\Theta}_{:,j}^{(k)}).
\end{aligned}$$

Thus, we have

$$\text{(global upper bound)} \quad \gamma_j^U = \epsilon\|\mathbf{\Lambda}_{j,:}^{(0)}\|_q + \mathbf{\Lambda}_{j,:}^{(0)}\mathbf{x_0} + \sum_{k=1}^{m} \mathbf{\Lambda}_{j,:}^{(k)}(\mathbf{b}^{(k)} + \mathbf{\Delta}_{:,j}^{(k)}),$$

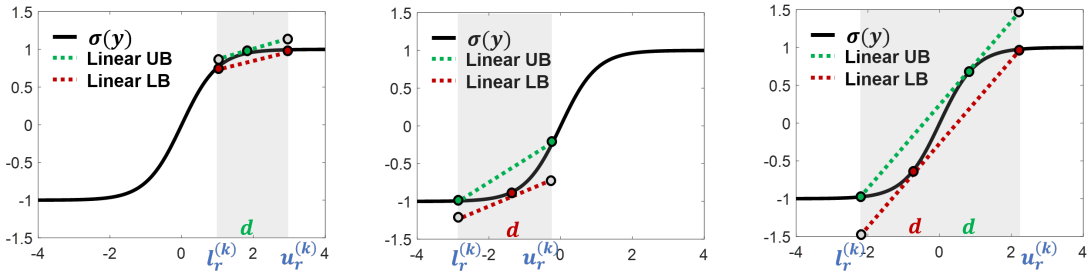
$$\text{(global lower bound)} \quad \gamma_j^L = -\epsilon\|\mathbf{\Omega}_{j,:}^{(0)}\|_q + \mathbf{\Omega}_{j,:}^{(0)}\mathbf{x_0} + \sum_{k=1}^{m} \mathbf{\Omega}_{j,:}^{(k)}(\mathbf{b}^{(k)} + \mathbf{\Theta}_{:,j}^{(k)}),$$

4.2.4 Case studies: CROWN for ReLU, tanh, sigmoid and arctan activations

In Section 4.2.1 we showed that as long as one can identify two linear functions $h_U(y), h_L(y)$ to bound a general activation function $\sigma(y)$ for each neuron, we can use Corollary 4.2.3 with a binary search to obtain certified lower bounds of minimum distortion. In this section, we illustrate how to find parameters $\alpha_{U,r}^{(k)}, \alpha_{L,r}^{(k)}$ and $\beta_{U,r}^{(k)}, \beta_{L,r}^{(k)}$ of $h_U(y), h_L(y)$ for four most widely used activation functions: ReLU, tanh, sigmoid and arctan. Other activations, including but not limited to leaky ReLU, ELU and softplus, can be easily incorporated into our CROWN framework following a similar procedure.

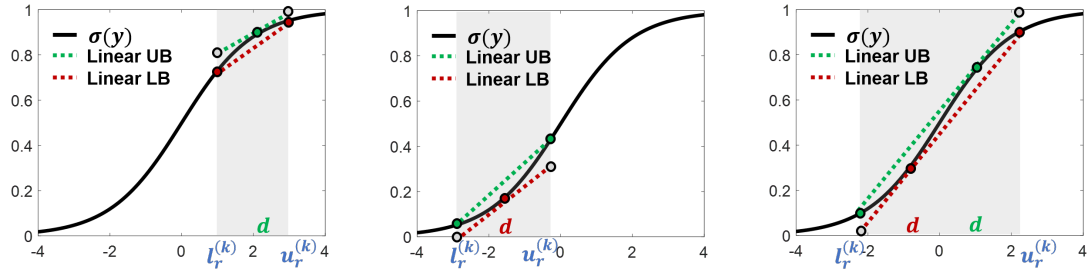
Segmenting activation functions. Based on the signs of $\mathbf{l}_r^{(k)}$ and $\mathbf{u}_r^{(k)}$, we define a partition $\{\mathcal{S}_k^+, \mathcal{S}_k^\pm, \mathcal{S}_k^-\}$ of set $[n_k]$ such that every neuron in k-th layer belongs to exactly one of the three sets. The formal definition of $\mathcal{S}_k^+$, $\mathcal{S}_k^\pm$ and $\mathcal{S}_k^-$ is $\mathcal{S}_k^+ = \{r \in [n_k] \mid 0 \leq \mathbf{l}_r^{(k)} \leq \mathbf{u}_r^{(k)}\}$, $\mathcal{S}_k^\pm = \{r \in [n_k] \mid \mathbf{l}_r^{(k)} < 0 < \mathbf{u}_r^{(k)}\}$, and $\mathcal{S}_k^- = \{r \in [n_k] \mid \mathbf{l}_r^{(k)} \leq \mathbf{u}_r^{(k)} \leq 0\}$. For neurons in each partitioned set, we define corresponding upper bound $h_{U,r}^{(k)}$ and lower bound $h_{L,r}^{(k)}$ in terms of $\mathbf{l}_r^{(k)}$ and $\mathbf{u}_r^{(k)}$. As we will see shortly, segmenting the activation functions based on $\mathbf{l}_r^{(k)}$ and $\mathbf{u}_r^{(k)}$ is useful to bound a given activation function. We note there are multiple ways of segmenting the activation functions and defining the partitioned sets (e.g. based on the values of $\mathbf{l}_r^{(k)}, \mathbf{u}_r^{(k)}$ rather than their signs), and we can easily incorporate this into our framework to provide the corresponding explicit output bounds for the new partition sets. In the case study, we consider $\mathcal{S}_k^+$, $\mathcal{S}_k^\pm$ and $\mathcal{S}_k^-$ for the four activations, as this partition reflects the curvature of tanh, sigmoid and arctan functions and activation states of ReLU.

Bounding tanh/sigmoid/arctan. For tanh activation, $\sigma(y) = \frac{1-e^{-2y}}{1+e^{-2y}}$; for sigmoid activation, $\sigma(y) = \frac{1}{1+e^{-y}}$; for arctan activation, $\sigma(y) = \arctan(y)$. All functions are convex on one side ($y < 0$) and concave on the other side ($y > 0$), thus the same rules can be used to find $h_{U,r}^{(k)}$ and $h_{L,r}^{(k)}$. Below we call $(\mathbf{l}_r^{(k)}, \sigma(\mathbf{l}_r^{(k)}))$ as left end-point and $(\mathbf{u}_r^{(k)}, \sigma(\mathbf{u}_r^{(k)}))$ as right end-point. For $r \in \mathcal{S}_k^+$, since $\sigma(y)$ is concave, we can let $h_{U,r}^{(k)}$ be



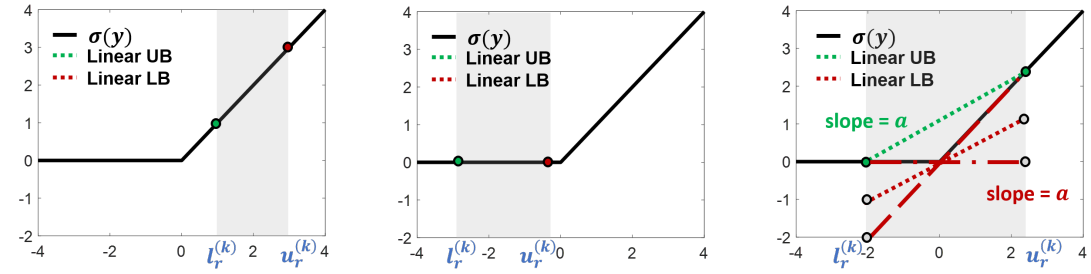
(a) $r \in \mathcal{S}_k^+$ (b) $r \in \mathcal{S}_k^-$ (c) $r \in \mathcal{S}_k^\pm$

Figure 4-1: $\sigma(y) = \tanh$. Green lines are the upper bounds $h_{U,r}^{(k)}$; red lines are the lower bounds $h_{L,r}^{(k)}$.



(a) $r \in \mathcal{S}_k^+$ (b) $r \in \mathcal{S}_k^-$ (c) $r \in \mathcal{S}_k^\pm$

Figure 4-2: The linear upper and lower bounds for $\sigma(y) = \text{sigmoid}$



(a) $r \in \mathcal{S}_k^+$ (b) $r \in \mathcal{S}_k^-$ (c) $r \in \mathcal{S}_k^\pm$

Figure 4-3: The linear upper and lower bounds for $\sigma(y) = \text{ReLU}$. For the cases (a) and (b), the linear upper bound and lower bound are exactly the function $\sigma(y)$ in the region (grey-shaded). For (c), we plot three out of many choices of lower bound, and they are $h_{L,r}^{(k)}(y) = 0$ (dashed-dotted), $h_{L,r}^{(k)}(y) = y$ (dashed), and $h_{L,r}^{(k)}(y) = \frac{\mathbf{u}_r^{(k)}}{\mathbf{u}_r^{(k)} - \mathbf{l}_r^{(k)}} y$ (dotted).

any tangent line of $\sigma(y)$ at point $d \in [\mathbf{l}_r^{(k)}, \mathbf{u}_r^{(k)}]$, and let $h_{L,r}^{(k)}$ pass the two end-points. Similarly, $\sigma(y)$ is concave for $r \in \mathcal{S}_k^+$, thus we can let $h_{L,r}^{(k)}$ be any tangent line of $\sigma(y)$ at point $d \in [\mathbf{l}_r^{(k)}, \mathbf{u}_r^{(k)}]$ and let $h_{U,r}^{(k)}$ pass the two end-points. Lastly, for $r \in \mathcal{S}_k^\pm$, we can let $h_{U,r}^{(k)}$ be the tangent line that passes the left end-point and $(d, \sigma(d))$ where $d \geq 0$

and $h_{U,r}^{(k)}$ be the tangent line that passes the right end-point and $(d, \sigma(d))$ where $d \leq 0$. The value of d for transcendental functions can be found using a binary search. The plots of upper and lower bounds for tanh and sigmoid are in Figure 4-1 and 4-2. Plots for arctan are similar and so omitted.

Bounding ReLU. For ReLU activation, $\sigma(y) = \max(0, y)$. If $r \in \mathcal{S}_k^+$, we have $\sigma(y) = y$ and so we can set $h_{U,r}^{(k)} = h_{L,r}^{(k)} = y$; if $r \in \mathcal{S}_k^-$, we have $\sigma(y) = 0$, and thus we can set $h_{U,r}^{(k)} = h_{L,r}^{(k)} = 0$; if $r \in \mathcal{S}_k^{\pm}$, we can set $h_{U,r}^{(k)} = \frac{\mathbf{u}_r^{(k)}}{\mathbf{u}_r^{(k)} - \mathbf{l}_r^{(k)}}(y - \mathbf{l}_r^{(k)})$ and $h_{L,r}^{(k)} = ay$, $0 \leq a \leq 1$. Setting $a = \frac{\mathbf{u}_r^{(k)}}{\mathbf{u}_r^{(k)} - \mathbf{l}_r^{(k)}}$ leads to the linear lower bound used in Fast-Lin [101]. Thus, Fast-Lin is a special case under our framework. We propose to *adaptively* choose a, where we set $a = 1$ when $\mathbf{u}_r^{(k)} \geq |\mathbf{l}_r^{(k)}|$ and $a = 0$ when $\mathbf{u}_r^{(k)} < |\mathbf{l}_r^{(k)}|$. In this way, the area between the lower bound $h_{L,r}^{(k)} = ay$ and $\sigma(y)$ (which reflects the gap between the lower bound and the ReLU function) is always minimized. As shown in our experiments, the adaptive selection of $h_{L,r}^{(k)}$ based on the value of $\mathbf{u}_r^{(k)}$ and $\mathbf{l}_r^{(k)}$ helps to achieve a tighter certified lower bound. Figure 4-3 illustrates the idea discussed here.

Summary. We summarized the above analysis on choosing valid linear functions $h_{U,r}^{(k)}$ and $h_{L,r}^{(k)}$ in Table 4.2 and 4.3. In general, as long as $h_{U,r}^{(k)}$ and $h_{L,r}^{(k)}$ are identified for the activation functions, we can use Corollary 4.2.3 to compute certified lower bounds for general activation functions. Note that there remain many other choices of $h_{U,r}^{(k)}$ and $h_{L,r}^{(k)}$ as valid upper/lower bounds of $\sigma(y)$, but ideally, we would like them to be close to $\sigma(y)$ in order to achieve a tighter lower bound of minimum distortion.

Table 4.2: Linear upper bound parameters of various activation functions: $h_{U,r}^{(k)}(y) = \alpha_{U,r}^{(k)}(y + \beta_{U,r}^{(k)})$

Upper bound $h_{U,r}^{(k)}$ for activation function	$r \in \mathcal{S}_k^+$		$r \in \mathcal{S}_k^-$		$r \in \mathcal{S}_k^\pm$	
	$\alpha_{U,r}^{(k)}$	$\beta_{U,r}^{(k)}$	$\alpha_{U,r}^{(k)}$	$\beta_{U,r}^{(k)}$	$\alpha_{U,r}^{(k)}$	$\beta_{U,r}^{(k)}$
ReLU	1	0	0	0	a ($a \geq \frac{\mathbf{u}_r^{(k)}}{\mathbf{u}_r^{(k)} - \mathbf{l}_r^{(k)}}$, e.g. $a = \frac{\mathbf{u}_r^{(k)}}{\mathbf{u}_r^{(k)} - \mathbf{l}_r^{(k)}}$)	$-\mathbf{l}_r^{(k)}$
Sigmoid, tanh (denoted as $\sigma(y)$)	$\sigma'(d)$ ($\mathbf{l}_r^{(k)} \leq d \leq \mathbf{u}_r^{(k)}$)	$\frac{\sigma(d)}{\alpha_{U,r}^{(k)}} - d$ *	$\frac{\sigma(\mathbf{u}_r^{(k)}) - \sigma(\mathbf{l}_r^{(k)})}{\mathbf{u}_r^{(k)} - \mathbf{l}_r^{(k)}}$	$\frac{\sigma(\mathbf{l}_r^{(k)})}{\alpha_{U,r}^{(k)}} - \mathbf{l}_r^{(k)}$	$\sigma'(d)$ ($\frac{\sigma(d) - \sigma(\mathbf{l}_r^{(k)})}{d - \mathbf{l}_r^{(k)}} - \sigma'(d) = 0,\ d \geq 0$) ◇	$\frac{\sigma(\mathbf{l}_r^{(k)})}{\alpha_{U,r}^{(k)}} - \mathbf{l}_r^{(k)}$

* If $\alpha_{U,r}^{(k)}$ is close to 0, we suggest to calculate the intercept directly, $\alpha_{U,r}^{(k)} \cdot \beta_{U,r}^{(k)} = \sigma(d) - \alpha_{U,r}^{(k)} d$, to avoid numerical issues in implementation. Same for other similar cases.

◇ Alternatively, if the solution $d \geq \mathbf{u}_r^{(k)}$, then we can set $\alpha_{U,r}^{(k)} = \frac{\sigma(\mathbf{u}_r^{(k)}) - \sigma(\mathbf{l}_r^{(k)})}{\mathbf{u}_r^{(k)} - \mathbf{l}_r^{(k)}}$.

Table 4.3: Linear lower bound parameters of various activation functions: $h_{L,r}^{(k)}(y) = \alpha_{L,r}^{(k)}(y + \beta_{L,r}^{(k)})$

Lower bound $h_{L,r}^{(k)}$ for activation function	$r \in \mathcal{S}_k^+$		$r \in \mathcal{S}_k^-$		$r \in \mathcal{S}_k^\pm$	
	$\alpha_{L,r}^{(k)}$	$\beta_{L,r}^{(k)}$	$\alpha_{L,r}^{(k)}$	$\beta_{L,r}^{(k)}$	$\alpha_{L,r}^{(k)}$	$\beta_{L,r}^{(k)}$
ReLU	1	0	0	0	a ($0 \leq a \leq 1$, e.g. $a = \frac{\mathbf{u}_r^{(k)}}{\mathbf{u}_r^{(k)} - \mathbf{l}_r^{(k)}}, 0, 1$)	0
Sigmoid, tanh (denoted as $\sigma(y)$)	$\frac{\sigma(\mathbf{u}_r^{(k)}) - \sigma(\mathbf{l}_r^{(k)})}{\mathbf{u}_r^{(k)} - \mathbf{l}_r^{(k)}}$	$\frac{\sigma(\mathbf{l}_r^{(k)})}{\alpha_{L,r}^{(k)}} - \mathbf{l}_r^{(k)}$	$\sigma'(d)$ ($\mathbf{l}_r^{(k)} \leq d \leq \mathbf{u}_r^{(k)}$)	$\frac{\sigma(d)}{\alpha_{L,r}^{(k)}} - d$	$\sigma'(d)$ ($\frac{\sigma(d) - \sigma(\mathbf{u}_r^{(k)})}{d - \mathbf{u}_r^{(k)}} - \sigma'(d) = 0,\ d \leq 0$) †	$\frac{\sigma(\mathbf{u}_r^{(k)})}{\alpha_{L,r}^{(k)}} - \mathbf{u}_r^{(k)}$

† Alternatively, if the solution $d \leq \mathbf{l}_r^{(k)}$, then we can set $\alpha_{L,r}^{(k)} = \frac{\sigma(\mathbf{u}_r^{(k)}) - \sigma(\mathbf{l}_r^{(k)})}{\mathbf{u}_r^{(k)} - \mathbf{l}_r^{(k)}}$.

4.2.5 Extend CROWN for Quadratic Bounds

In the earlier sections, we show that adaptive linear bounds can be applied on the activation functions and hence the output of a NN function can be bounded by linear functions. In this section, we further show that, in addition to the linear bounds on activation functions, the proposed CROWN framework can also incorporate quadratic bounds by adding a quadratic term to $h_{U,r}^{(k)}$ and $h_{L,r}^{(k)}$: $h_{U,r}^{(k)}(y) = \eta_{U,r}^{(k)} y^2 + \alpha_{U,r}^{(k)}(y + \beta_{U,r}^{(k)})$, $h_{L,r}^{(k)}(y) = \eta_{L,r}^{(k)} y^2 + \alpha_{L,r}^{(k)}(y + \beta_{L,r}^{(k)})$, where $\eta_{U,r}^{(k)}, \eta_{L,r}^{(k)} \in \mathbb{R}$. Following the procedure of unwrapping the activation functions at the layer $m-1$, we show in section 4.2.6 that the output upper bound and lower bound with quadratic approximations are:

$$f_j^U(\mathbf{x}) = \Phi_{m-2}(\mathbf{x})^\top \mathbf{Q}_U^{(m-1)} \Phi_{m-2}(\mathbf{x}) + 2\mathbf{p}_U^{(m-1)} \Phi_{m-2}(\mathbf{x}) + s_U^{(m-1)}, \tag{4.19}$$

$$f_j^L(\mathbf{x}) = \Phi_{m-2}(\mathbf{x})^\top \mathbf{Q}_L^{(m-1)} \Phi_{m-2}(\mathbf{x}) + 2\mathbf{p}_L^{(m-1)} \Phi_{m-2}(\mathbf{x}) + s_L^{(m-1)}, \tag{4.20}$$

where $\mathbf{Q}_U^{(m-1)} = \mathbf{W}^{(m-1)\top} \mathbf{D}_U^{(m-1)} \mathbf{W}^{(m-1)}$, $\mathbf{Q}_L^{(m-1)} = \mathbf{W}^{(m-1)\top} \mathbf{D}_L^{(m-1)} \mathbf{W}^{(m-1)}$, $\mathbf{p}_U^{(m-1)}$, $\mathbf{p}_L^{(m-1)}$, $s_U^{(m-1)}$, and $s_L^{(m-1)}$ are defined in Section 4.2.6.

When $m = 2$, $\Phi_{m-2}(\mathbf{x}) = \mathbf{x}$ and we can directly optimize over $\mathbf{x} \in B_p(\mathbf{x_0}, \epsilon)$; otherwise, we can use the post activation bounds of layer $m-2$ as the constraints. $\mathbf{D}_U^{(m-1)}$ in (4.19) is a diagonal matrix with i-th entry being $\mathbf{W}_{j,i}^{(m)} \eta_{U,i}^{(m-1)}$, if $\mathbf{W}_{j,i}^{(m)} \geq 0$ or $\mathbf{W}_{j,i}^{(m)} \eta_{L,i}^{(m-1)}$, if $\mathbf{W}_{j,i}^{(m)} < 0$. Thus, in general $\mathbf{Q}_U^{(m-1)}$ is indefinite, resulting in a non-convex optimization when finding the global bounds as in Corollary 4.2.3. Fortunately, by properly choosing the quadratic bounds, we can make the problem $\max_{\mathbf{x} \in B_p(\mathbf{x_0}, \epsilon)} f_j^U(\mathbf{x})$ into a convex Quadratic Programming problem; for example, we can let $\eta_{U,i}^{(m-1)} = 0$ for all $\mathbf{W}_{j,i}^{(m)} > 0$ and let $\eta_{L,i}^{(m-1)} > 0$ to make $\mathbf{D}_U^{(m-1)}$ have only negative and zero diagonals for the maximization problem – this is equivalent to applying a linear upper bound and a quadratic lower bound to bound the activation function. Similarly, for $\mathbf{D}_L^{(m-1)}$, we let $\eta_{U,i}^{(m-1)} = 0$ for all $\mathbf{W}_{j,i}^{(m)} < 0$ and let $\eta_{L,i}^{(m-1)} > 0$ to make $\mathbf{D}_L^{(m-1)}$ have non-negative diagonals and hence the problem $\min_{\mathbf{x} \in B_p(\mathbf{x_0}, \epsilon)} f_j^L(\mathbf{x})$ is convex. We can solve this convex program with projected gradient descent (PGD) for $\mathbf{x} \in B_p(\mathbf{x_0}, \epsilon)$ and Armijo line search. Empirically, we find that PGD usually converges within a few iterations.

4.2.6 Constructing $f_j^U(\mathbf{x})$ and $f_j^L(\mathbf{x})$ by Quadratic Approximation

We show that if we apply quadratic bounds on the activation functions in the first hidden layer, the output of neural network function can be bounded by quadratic functions $f_j^U(\mathbf{x})$ and $f_j^L(\mathbf{x})$. Notably, we want to select the quadratic bounds on the activation functions such that $f_j^U(\mathbf{x})$ will be a concave quadratic function, while $f_j^L(\mathbf{x})$ will be a convex quadratic function. The reason is because our layer-wise bounds require solving

$$\max_{x\in B_p(\mathbf{x_0},\epsilon)} f_j^U(\mathbf{x}) \text{ and } \min_{x\in B_p(\mathbf{x_0},\epsilon)} f_j^L(\mathbf{x}).$$

Hence, if $f_j^U(\mathbf{x})$ is concave and $f_j^L(\mathbf{x})$ is convex, then the optimization problems are convex, and we would be able to solve this efficiently to global optimum. Below we first illustrate the construction on the upper bound $f_j^U(\mathbf{x})$ and then the lower bound $f_j^L(\mathbf{x})$.

Upper bound. Let $f_j^U(\mathbf{x})$ be an upper bound of $f_j(\mathbf{x})$. To compute $f_j^U(\mathbf{x})$ with quadratic approximations, we can still apply (4.5) and (4.6) except that $h_{U,r}^{(k)}(y)$ and $h_{L,r}^{(k)}(y)$ are replaced by the following quadratic functions:

$$h_{U,r}^{(k)}(y) = \eta_{U,r}^{(k)} y^2 + \alpha_{U,r}^{(k)}(y + \beta_{U,r}^{(k)}),\ h_{L,r}^{(k)}(y) = \eta_{L,r}^{(k)} y^2 + \alpha_{L,r}^{(k)}(y + \beta_{L,r}^{(k)}).$$

Therefore,

$$f_j^U(\mathbf{x}) = \sum_{\mathbf{w}_{j,i}^{(m)}\geq 0} \mathbf{W}_{j,i}^{(m)} \cdot h_{U,i}^{(m-1)}(\mathbf{y}_i^{(m-1)}) + \sum_{\mathbf{w}_{j,i}^{(m)}<0} \mathbf{W}_{j,i}^{(m)} \cdot h_{L,i}^{(m-1)}(\mathbf{y}_i^{(m-1)}) + \mathbf{b}_j^{(m)}, \tag{4.21}$$

$$= \sum_{i=1}^{n_{m-1}} \mathbf{W}_{j,i}^{(m)} \left(\tau_{j,i}^{(m-1)} \mathbf{y}_i^{(m-1)2} + \lambda_{j,i}^{(m-1)}(\mathbf{y}_i^{(m-1)} + \mathbf{\Delta}_{i,j}^{(m-1)})\right) + \mathbf{b}_j^{(m)}, \tag{4.22}$$

$$= \mathbf{y}^{(m-1)\top} \mathrm{diag}(\mathbf{q}_{U,j}^{(m-1)}) \mathbf{y}^{(m-1)} + \mathbf{\Lambda}_{j,:}^{(m-1)} \mathbf{y}^{(m-1)} + \mathbf{W}_{j,:}^{(m)} \mathbf{\Delta}_{:,j}^{(m-1)}, \tag{4.23}$$

$$= \Phi_{m-2}(\mathbf{x})^\top \mathbf{Q}_U^{(m-1)} \Phi_{m-2}(\mathbf{x}) + 2\mathbf{p}_U^{(m-1)} \Phi_{m-2}(\mathbf{x}) + s_U^{(m-1)}. \tag{4.24}$$

From (4.21) to (4.22), we replace $h_{U,i}^{(m-1)}(\mathbf{y}_i^{(m-1)})$ and $h_{L,i}^{(m-1)}(\mathbf{y}_i^{(m-1)})$ by their definitions and let

$$(\tau_{j,i}^{(m-1)}, \lambda_{j,i}^{(m-1)}, \mathbf{\Delta}_{i,j}^{(m-1)}) = \begin{cases} (\eta_{U,i}^{(m-1)}, \alpha_{U,i}^{(m-1)}, \beta_{U,i}^{(m-1)}) & \text{if } \mathbf{W}_{j,i}^{(m)} \geq 0; \\ (\eta_{L,i}^{(m-1)}, \alpha_{L,i}^{(m-1)}, \beta_{L,i}^{(m-1)}) & \text{if } \mathbf{W}_{j,i}^{(m)} < 0. \end{cases}$$

From (4.22) to (4.23), we let $\mathbf{q}_{U,j}^{(m-1)} = \mathbf{W}_{j,:}^{(m)} \odot \tau_{j,i}^{(m-1)}$, and write in the matrix form. From (4.23) to (4.24), we substitute $\mathbf{y}^{(m-1)}$ by its definition: $\mathbf{y}^{(m-1)} = \mathbf{W}^{(m-1)}\Phi_{(m-2)}(\mathbf{x}) + \mathbf{b}^{(m-1)}$ and then collect the quadratic terms, linear terms and constant terms of $\Phi_{(m-2)}(\mathbf{x})$, where

$$\begin{aligned} \mathbf{Q}_U^{(m-1)} &= \mathbf{W}^{(m-1)\top}\text{diag}(\mathbf{q}_{U,j}^{(m-1)})\mathbf{W}^{(m-1)}, \\ \mathbf{p}_U^{(m-1)} &= \mathbf{b}^{(m-1)\top} \odot \mathbf{q}_{U,j}^{(m-1)} + \mathbf{\Lambda}_{j,:}^{(m-1)}, \\ s_U^{(m-1)} &= \mathbf{p}_U^{(m-1)}\mathbf{b}^{(m-1)} + \mathbf{W}_{j,:}^{(m)}\mathbf{\Delta}_{:,j}^{(m-1)}. \end{aligned}$$

Lower bound. Similar to the above derivation, we can simply swap $h_{U,r}^{(k)}$ and $h_{L,r}^{(k)}$ and obtain lower bound $f_j^L(\mathbf{x})$:

$$\begin{aligned} f_j^L(\mathbf{x}) &= \sum_{\mathbf{w}_{j,i}^{(m)}<0} \mathbf{W}_{j,i}^{(m)} \cdot h_{U,i}^{(m-1)}(\mathbf{y}_i^{(m-1)}) + \sum_{\mathbf{w}_{j,i}^{(m)}\geq 0} \mathbf{W}_{j,i}^{(m)} \cdot h_{L,i}^{(m-1)}(\mathbf{y}_i^{(m-1)}) + \mathbf{b}_j^{(m)}, \\ &= \Phi_{m-2}(\mathbf{x})^\top \mathbf{Q}_L^{(m-1)} \Phi_{m-2}(\mathbf{x}) + 2\mathbf{p}_L^{(m-1)}\Phi_{m-2}(\mathbf{x}) + s_L^{(m-1)}, \end{aligned}$$

where

$$\mathbf{Q}_L^{(m-1)} = \mathbf{W}^{(m-1)\top}\text{diag}(\mathbf{q}_{L,j}^{(m-1)})\mathbf{W}^{(m-1)}, \ \mathbf{q}_{L,j}^{(m-1)} = \mathbf{W}_{j,:}^{(m)} \odot \nu_{j,i}^{(m-1)}; \tag{4.25}$$

$$\mathbf{p}_U^{(m-1)} = \mathbf{b}^{(m-1)\top} \odot \mathbf{q}_{U,j}^{(m-1)} + \mathbf{\Lambda}_{j,:}^{(m-1)}, \quad \mathbf{p}_L^{(m-1)} = \mathbf{b}^{(m-1)\top} \odot \mathbf{q}_{L,j}^{(m-1)} + \mathbf{\Omega}_{j,:}^{(m-1)}; \tag{4.26}$$

$$s_U^{(m-1)} = \mathbf{p}_U^{(m-1)}\mathbf{b}^{(m-1)} + \mathbf{W}_{j,:}^{(m)}\mathbf{\Delta}_{:,j}^{(m-1)}, \quad s_L^{(m-1)} = \mathbf{p}_L^{(m-1)}\mathbf{b}^{(m-1)} + \mathbf{W}_{j,:}^{(m)}\mathbf{\Theta}_{:,j}^{(m-1)}, \tag{4.27}$$

and

$$(\nu_{j,i}^{(m-1)}, \omega_{j,i}^{(m-1)}, \mathbf{\Theta}_{i,j}^{(m-1)}) = \begin{cases} (\eta_{L,i}^{(m-1)}, \alpha_{L,i}^{(m-1)}, \beta_{L,i}^{(m-1)}) & \text{if } \mathbf{W}_{j,i}^{(m)} \geq 0; \\ (\eta_{U,i}^{(m-1)}, \alpha_{U,i}^{(m-1)}, \beta_{U,i}^{(m-1)}) & \text{if } \mathbf{W}_{j,i}^{(m)} < 0. \end{cases} \tag{4.28}$$

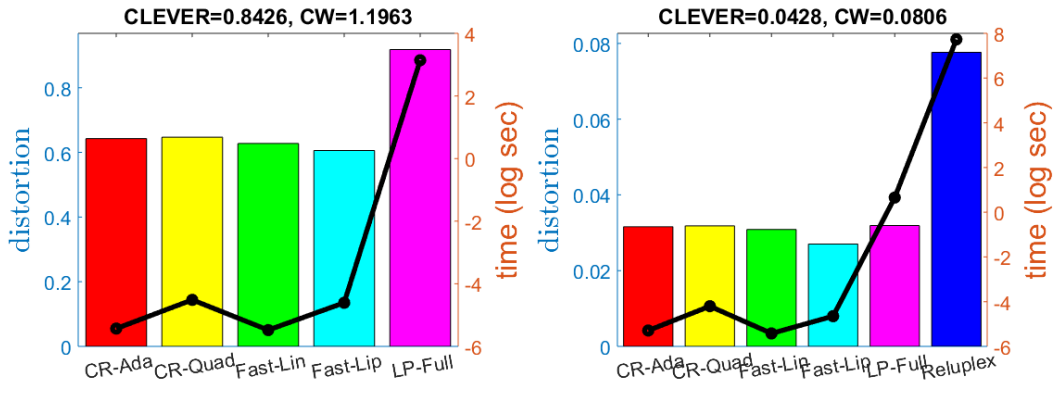
4.3 Experiments

Methods. For ReLU networks, CROWN-Ada is CROWN with adaptive linear bounds (Sec. 4.2.4), CROWN-Quad is CROWN with quadratic bounds (Sec. 4.2.5). Fast-Lin and Fast-Lip are state-of-the-art fast certified lower bound proposed in [101]. Reluplex can solve the exact minimum adversarial distortion but is only computationally feasible for very small networks. LP-Full is based on the LP formulation in [52] and we solve LPs for each neuron exactly to achieve the best possible bound. For networks with other activation functions, CROWN-general is our proposed method.

Model and Dataset. We evaluate CROWN and other baselines on multi-layer perceptron (MLP) models trained on MNIST and CIFAR-10 datasets. We denote a feed-forward network with m layers and n neurons per layer as $m \times [n]$. For models with ReLU activation, we use pretrained models provided by [101] and also evaluate the same set of 100 random test images and random attack targets as in [101] (according to their released code) to make our results comparable. For training NN models with other activation functions, we search for best learning rate and weight decay parameters to achieve a similar level of accuracy as ReLU models.

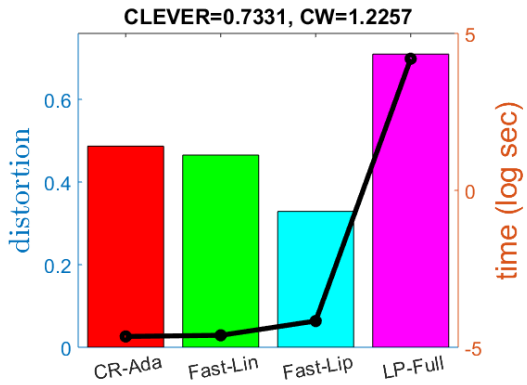
Implementation and Setup. We implement our algorithm using Python (numpy with numba). Most computations in our method are matrix operations that can be automatically parallelized by the BLAS library; however, we set the number of BLAS threads to 1 for a fair comparison to other methods. Experiments were conducted on an Intel Skylake server CPU running at 2.0 GHz on Google Cloud. Our code is available at https://github.com/CROWN-Robustness/Crown

Results on Small Networks. Figure 4-4 shows the certified lower bound for ℓ_2 and ℓ_∞ distortions found by different algorithms on small networks, where Reluplex is feasible and we can observe the gap between different certified lower bounds and the true minimum adversarial distortion. Reluplex and LP-Full are orders of magnitudes slower than other methods (note the logarithmic scale on right y-axis), and CROWN-

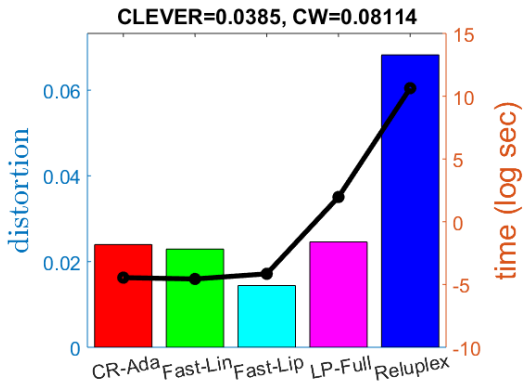


(a) MNIST $2 \times [20]$, ℓ_2

(b) MNIST $2 \times [20]$, ℓ_∞



(c) MNIST $3 \times [20]$, ℓ_2



(d) MNIST $3 \times [20]$, ℓ_∞

Figure 4-4: Certified lower bounds and minimum distortion comparisons for ℓ_2 and ℓ_∞ distortions. Left y-axis is distortion and right y-axis (black line) is computation time (seconds, logarithmic scale). On the top of figures are the avg. CLEVER score and the upper bound found by C&W attack [13]. From left to right in (a)-(d): CROWN-Ada, (CROWN-Quad), Fast-Lin, Fast-Lip, LP-Full and (Reluplex).

Quad (for 2-layer) and CROWN-Ada achieve the largest lower bounds. Improvements of CROWN-Ada over Fast-Lin are more significant in larger NNs, as we show below.

Results on Large ReLU Networks. Table 4.4 demonstrates the lower bounds found by different algorithms for all common ℓ_p norms. CROWN-Ada significantly outperforms Fast-Lin and Fast-Lip, while the computation time increased by less than 2X over Fast-Lin, and is comparable with Fast-Lip.

Results on Different Activations. Table 4.5 compares the certified lower bound computed by CROWN-general for four activation functions and different ℓ_p norm on

Table 4.4: Comparison of certified lower bounds on large ReLU networks. Bounds are the average over 100 images (skipped misclassified images) with random attack targets. Percentage improvements are calculated against Fast-Lin as Fast-Lip is worse than Fast-Lin [101].

Network	Certified Bounds				Improvement (%)	Average Computation Time (sec)		
	ℓ_p norm	Fast-Lin	Fast-Lip	CROWN-Ada	CROWN-Ada vs Fast-Lin	Fast-Lin	Fast-Lip	CROWN-Ada
MNIST $4 \times [1024]$	ℓ_1	1.57649	0.72800	**1.88217**	+19%	1.80	2.04	3.54
	ℓ_2	0.18891	0.06487	**0.22811**	+21%	1.78	1.96	3.79
	ℓ_∞	0.00823	0.00264	**0.00997**	+21%	1.53	2.17	3.57
CIFAR-10 $7 \times [1024]$	ℓ_1	0.86468	0.09239	**1.09067**	+26%	13.21	19.76	22.43
	ℓ_2	0.05937	0.00407	**0.07496**	+26%	12.57	18.71	21.82
	ℓ_∞	0.00134	0.00008	**0.00169**	+26%	8.98	20.34	16.66

Table 4.5: Comparison of certified lower bounds by CROWN-Ada on ReLU networks and CROWN-general on networks with tanh, sigmoid and arctan activations. CIFAR models with sigmoid activations achieve much worse accuracy than other networks and are thus excluded.

Network	Certified Bounds by CROWN-Ada and CROWN-general					Average Computation Time (sec)			
	ℓ_p norm	ReLU	tanh	sigmoid	arctan	ReLU	tanh	sigmoid	arctan
MNIST $3 \times [1024]$	ℓ_1	3.00231	2.48407	2.94239	2.33246	1.25	1.61	1.68	1.70
	ℓ_2	0.50841	0.27287	0.44471	0.30345	1.26	1.76	1.61	1.75
	ℓ_∞	0.02576	0.01182	0.02122	0.01363	1.37	1.78	1.76	1.77
CIFAR-10 $6 \times [2048]$	ℓ_1	0.91201	0.44059	-	0.46198	71.62	89.77	-	83.80
	ℓ_2	0.05245	0.02538	-	0.02515	71.51	84.22	-	83.12
	ℓ_∞	0.00114	0.00055	-	0.00055	49.28	59.72	-	58.04

large networks. CROWN-general is able to certify non-trivial lower bounds for all four activation functions efficiently. Comparing to CROWN-Ada on ReLU networks, certifying general activations that are not piece-wise linear only incurs about 20% additional computational overhead.

4.3.1 Results on CROWN-Ada

We compare CROWN-Ada, Fast-Lin, Fast-Lip and Op-nrom on ReLU networks in Table 4.6. LP-full and Reluplex cannot finish within a reasonable amount of time for all the networks reported here. We also include Op-norm, where we directly compute the operator norm (for example, for $p = 2$ it is the spectral norm) for each layer and use their products as a global Lipschitz constant and then compute the robustness lower bound. CLEVER is an estimated local robustness radius lower bound, and attacking algorithms (including CW [13] and EAD [15]) provide upper bounds of the minimum adversarial distortion. For each norm, we consider the robustness against three targeted attack classes: the runner-up class (with the second largest probability), a random class and the least likely class. It is clear that CROWN-Ada notably improves the lower bound comparing to Fast-Lin, especially for larger and deeper networks, the improvements can be up to 28%.

Table 4.6: Results of ReLU networks in 4.3.1

Networks			Lower bounds and upper bounds (Avg.)							Time per Image (Avg.)		
			Lower Bounds (certified)				improvements	Uncertified		Lower Bounds		
Config	p	Target	[101]		[94]	Our algorithm	over	[102]	Attacks	[101]		Our bound
			Fast-Lin	Fast-Lip	Op norm	CROWN-Ada	Fast-Lin	CLEVER	CW/EAD	Fast-Lin	Fast-Lip	CROWN-Ada
MNIST $2 \times [1024]$	∞	runner-up	0.02256	0.01802	0.00159	0.02467	+9.4%	0.0447	0.0856	163 ms	179 ms	128 ms
		rand	0.03083	0.02512	0.00263	0.03353	+8.8%	0.0708	0.1291	176 ms	213 ms	166 ms
		least	0.03854	0.03128	0.00369	0.04221	+9.5%	0.0925	0.1731	176 ms	251 ms	143 ms
	2	runner-up	0.46034	0.42027	0.24327	0.50110	+8.9%	0.8104	1.1874	154 ms	184 ms	110 ms
		rand	0.63299	0.59033	0.40201	0.68506	+8.2%	1.2841	1.8779	141 ms	212 ms	133 ms
		least	0.79263	0.73133	0.56509	0.86377	+9.0%	1.6716	2.4556	152 ms	291 ms	116 ms
	1	runner-up	2.78786	3.46500	0.20601	3.01633	+8.2%	4.5970	9.5295	159 ms	989 ms	136 ms
		rand	3.88241	5.10000	0.35957	4.17760	+7.6%	7.4186	17.259	168 ms	1.15 s	157 ms
		least	4.90809	6.36600	0.48774	5.33261	+8.6%	9.9847	23.933	179 ms	1.37 s	144 ms
MNIST $3 \times [1024]$	∞	runner-up	0.01830	0.01021	0.00004	0.02114	+15.5%	0.0509	0.1037	805 ms	1.28 s	1.33 s
		rand	0.02216	0.01236	0.00007	0.02576	+16.2%	0.0717	0.1484	782 ms	859 ms	1.37 s
		least	0.02432	0.01384	0.00009	0.02835	+16.6%	0.0825	0.1777	792 ms	684 ms	1.37 s
	2	runner-up	0.35867	0.22120	0.06626	0.41295	+15.1%	0.8402	1.3513	732 ms	1.06 s	1.26 s
		rand	0.43892	0.26980	0.10233	0.50841	+15.8%	1.2441	2.0387	711 ms	696 ms	1.26 s
		least	0.48361	0.30147	0.13256	0.56167	+16.1%	1.4401	2.4916	723 ms	655 ms	1.25 s
	1	runner-up	2.08887	1.80150	0.00734	2.39443	+14.6%	4.8370	10.159	685 ms	2.36 s	1.15 s
		rand	2.59898	2.25950	0.01133	3.00231	+15.5%	7.2177	17.796	743 ms	2.69 s	1.25 s
		least	2.87560	2.50000	0.01499	3.33231	+15.9%	8.3523	22.395	729 ms	3.08 s	1.31 s
MNIST $4 \times [1024]$	∞	runner-up	0.00715	0.00219	0.00001	0.00861	+20.4%	0.0485	0.08635	1.54 s	3.42 s	3.23 s
		rand	0.00823	0.00264	0.00001	0.00997	+21.1%	0.0793	0.1303	1.53 s	2.17 s	3.57 s
		least	0.00899	0.00304	0.00001	0.01096	+21.9%	0.1028	0.1680	1.74 s	2.00 s	3.87 s
	2	runner-up	0.16338	0.05244	0.11015	0.19594	+19.9%	0.8689	1.2422	1.79 s	2.58 s	3.52 s
		rand	0.18891	0.06487	0.17734	0.22811	+20.8%	1.4231	1.8921	1.78 s	1.96 s	3.79 s
		least	0.20671	0.07440	0.23710	0.25119	+21.5%	1.8864	2.4451	1.98 s	2.01 s	4.01 s
	1	runner-up	1.33794	0.58480	0.00114	1.58151	+18.2%	5.2685	10.079	1.87 s	1.93 s	3.34 s
		rand	1.57649	0.72800	0.00183	1.88217	+19.4%	8.9764	17.200	1.80 s	2.04 s	3.54 s
		least	1.73874	0.82800	0.00244	2.09157	+20.3%	11.867	23.910	1.94 s	2.40 s	3.72 s
CIFAR $5 \times [2048]$	∞	runner-up	0.00137	0.00020	0.00000	0.00167	+21.9%	0.0062	0.00950	18.2 s	38.2 s	33.1 s
		rand	0.00170	0.00030	0.00000	0.00212	+24.7%	0.0147	0.02351	19.6 s	48.2 s	36.7 s
		least	0.00188	0.00036	0.00000	0.00236	+25.5%	0.0208	0.03416	20.4 s	50.5 s	38.6 s
	2	runner-up	0.06122	0.00948	0.00156	0.07466	+22.0%	0.2712	0.3778	24.2 s	39.4 s	41.0 s
		rand	0.07654	0.01417	0.00333	0.09527	+24.5%	0.6399	0.9497	26.0 s	31.2 s	42.5 s
		least	0.08456	0.01778	0.00489	0.10588	+25.2%	0.9169	1.4379	25.0 s	33.2 s	44.4 s
	1	runner-up	0.93836	0.22632	0.00000	1.13799	+21.3%	4.0755	7.6529	24.7 s	45.1 s	40.5 s
		rand	1.18928	0.31984	0.00000	1.47393	+23.9%	9.7145	21.643	25.7 s	36.2 s	44.0 s
		least	1.31904	0.38887	0.00001	1.64452	+24.7%	12.793	34.497	26.0 s	31.7 s	44.9 s
CIFAR $6 \times [2048]$	∞	runner-up	0.00075	0.00005	0.00000	0.00094	+25.3%	0.0054	0.00770	27.6 s	64.7 s	47.3 s
		rand	0.00090	0.00007	0.00000	0.00114	+26.7%	0.0131	0.01866	28.1 s	72.3 s	49.3 s
		least	0.00095	0.00008	0.00000	0.00122	+28.4%	0.0199	0.02868	28.1 s	76.3 s	49.4 s
	2	runner-up	0.03462	0.00228	0.00476	0.04314	+24.6%	0.2394	0.2979	37.0 s	60.7 s	65.8 s
		rand	0.04129	0.00331	0.01079	0.05245	+27.0%	0.5860	0.7635	40.0 s	56.8 s	71.5 s
		least	0.04387	0.00385	0.01574	0.05615	+28.0%	0.8756	1.2111	40.0 s	56.3 s	72.5 s
	1	runner-up	0.59636	0.05647	0.00000	0.73727	+23.6%	3.3569	6.0112	37.2 s	65.6 s	66.8 s
		rand	0.72178	0.08212	0.00000	0.91201	+26.4%	8.2507	17.160	39.5 s	53.5 s	71.6 s
		least	0.77179	0.09397	0.00000	0.98331	+27.4%	12.603	28.958	40.7 s	42.1 s	72.5 s
CIFAR $7 \times [1024]$	∞	runner-up	0.00119	0.00006	0.00000	0.00148	+24.4%	0.0062	0.0102	8.98 s	20.1 s	16.2 s
		rand	0.00134	0.00008	0.00000	0.00169	+26.1%	0.0112	0.0218	8.98 s	20.3 s	16.7 s
		least	0.00141	0.00010	0.00000	0.00179	+27.0%	0.0148	0.0333	8.81 s	22.1 s	17.4 s
	2	runner-up	0.05279	0.00308	0.00020	0.06569	+24.4%	0.2661	0.3943	12.7 s	20.9 s	20.7 s
		rand	0.05937	0.00407	0.00029	0.07496	+26.3%	0.5145	0.9730	12.6 s	18.7 s	21.8 s
		least	0.06249	0.00474	0.00038	0.07943	+27.1%	0.6253	1.3709	12.9 s	20.7 s	22.2 s
	1	runner-up	0.76648	0.07028	0.00000	0.95204	+24.2%	4.815	7.9987	12.8 s	21.0 s	21.9 s
		rand	0.86468	0.09239	0.00000	1.09067	+26.1%	8.630	22.180	13.2 s	19.8 s	22.4 s
		least	0.91127	0.10639	0.00000	1.15687	+27.0%	11.44	31.529	13.3 s	17.6 s	22.9 s

4.3.2 Results on CROWN-general

We compare certified robustness radius lower bounds computed by CROWN-general on networks with tanh, sigmoid and arctan activations in Table 4.7. CIFAR models with sigmoid activations achieve much worse accuracy than other networks and are thus excluded. For each norm, we consider the robustness against three targeted attack classes: the runner-up class (with the second largest probability), a random class and the least likely class.

Table 4.7: Results of tanh/sigmoid/arctan networks in Section 4.3.2

Network			Certified Bounds by CROWN-general			Average Computation Time (sec)		
	ℓ_p norm	target	tanh	sigmoid	arctan	tanh	sigmoid	arctan
MNIST $3 \times [1024]$	ℓ_∞	runner-up	0.0164	0.0225	0.0169	0.3374	0.3213	0.3148
		random	0.0230	0.0325	0.0240	0.3185	0.3388	0.3128
		least	0.0306	0.0424	0.0314	0.3129	0.3586	0.3156
	ℓ_2	runner-up	0.3546	0.4515	0.3616	0.3139	0.3110	0.3005
		random	0.5023	0.6552	0.5178	0.3044	0.3183	0.2931
		least	0.6696	0.8576	0.6769	0.3869	0.3495	0.2676
	ℓ_1	runner-up	2.4600	2.7953	2.4299	0.2940	0.3339	0.3053
		random	3.5550	4.0854	3.5995	0.3277	0.3306	0.3109
		least	4.8215	5.4528	4.7548	0.3201	0.3915	0.3254
MNIST $4 \times [1024]$	ℓ_∞	runner-up	0.0091	0.0162	0.0107	1.6794	1.7902	1.7099
		random	0.0118	0.0212	0.0136	1.7783	1.7597	1.7667
		least	0.0147	0.0243	0.0165	1.8908	1.8483	1.7930
	ℓ_2	runner-up	0.2086	0.3389	0.2348	1.6416	1.7606	1.8267
		random	0.2729	0.4447	0.3034	1.7589	1.7518	1.6945
		least	0.3399	0.5064	0.3690	1.8206	1.7929	1.8264
	ℓ_1	runner-up	1.8296	2.2397	1.7481	1.5506	1.6052	1.6704
		random	2.4841	2.9424	2.3325	1.6149	1.7015	1.6847
		least	3.1261	3.3486	2.8881	1.7762	1.7902	1.8345
MNIST $5 \times [1024]$	ℓ_∞	runner-up	0.0060	0.0150	0.0062	3.9916	4.4614	3.7635
		random	0.0073	0.0202	0.0077	3.5068	4.4069	3.7387
		least	0.0084	0.0230	0.0091	3.9076	4.6283	3.9730
	ℓ_2	runner-up	0.1369	0.3153	0.1426	4.1634	4.3311	4.1039
		random	0.1660	0.4254	0.1774	4.1468	4.1797	4.0898
		least	0.1909	0.4849	0.2096	4.5045	4.4773	4.5497
	ℓ_1	runner-up	1.1242	2.0616	1.2388	4.4911	3.9944	4.4436
		random	1.3952	2.8082	1.5842	4.4543	4.0839	4.2609
		least	1.6231	3.2201	1.9026	4.4674	4.5508	4.5154
CIFAR-10 $5 \times [2048]$	ℓ_∞	runner-up	0.0005	-	0.0006	37.3918	-	37.1383
		random	0.0008	-	0.0009	38.0841	-	37.9199
		least	0.0010	-	0.0011	39.1638	-	39.4041
	ℓ_2	runner-up	0.0219	-	0.0256	47.4896	-	48.3390
		random	0.0368	-	0.0406	54.0104	-	52.7471
		least	0.0460	-	0.0497	55.8924	-	56.3877
	ℓ_1	runner-up	0.3744	-	0.4491	46.4041	-	47.1640
		random	0.6384	-	0.7264	54.2138	-	51.6295
		least	0.8051	-	0.8955	56.2512	-	55.6069
CIFAR-10 $6 \times [2048]$	ℓ_∞	runner-up	0.0004	-	0.0003	59.5020	-	58.2473
		random	0.0006	-	0.0006	59.7220	-	58.0388
		least	0.0006	-	0.0007	60.8031	-	60.9790
	ℓ_2	runner-up	0.0177	-	0.0163	78.8801	-	72.1884
		random	0.0254	-	0.0251	84.2228	-	83.1202
		least	0.0294	-	0.0306	86.2997	-	86.9320
	ℓ_1	runner-up	0.3043	-	0.2925	78.7486	-	70.2496
		random	0.4406	-	0.4620	89.7717	-	83.7972
		least	0.5129	-	0.5665	87.2094	-	86.6502
CIFAR-10 $7 \times [1024]$	ℓ_∞	runner-up	0.0006	-	0.0005	20.8612	-	20.5169
		random	0.0008	-	0.0007	21.4550	-	21.2134
		least	0.0008	-	0.0008	21.3406	-	21.1804
	ℓ_2	runner-up	0.0260	-	0.0225	27.9442	-	27.0240
		random	0.0344	-	0.0317	30.3782	-	29.8086
		least	0.0376	-	0.0371	30.7492	-	30.7321
	ℓ_1	runner-up	0.3826	-	0.3648	28.1898	-	27.1238
		random	0.5087	-	0.5244	29.6373	-	30.5106
		least	0.5595	-	0.6171	31.3457	-	30.6481

4.4 Conclusion

In this Chapter, we have presented a general framework CROWN to efficiently compute a certified lower bound of minimum distortion in neural networks for any given data point $\mathbf{x_0}$. CROWN features adaptive bounds for improved local robustness radius certification and applies to general activation functions. Moreover, experiments show that (i) CROWN outperforms state-of-the-art baselines on ReLU networks and (ii) CROWN can efficiently certify non-trivial lower bounds for large networks with over 10K neurons and with different activation functions spanning from tanh, sigmoid to ReLU and arctan.

Chapter 5

Robustness Quantification with Probabilistic Guarantees

5.1 Introduction

In the previous Chapters, we have developed two robustness quantification techniques, CLEVER and CROWN to compute the theoretical lower bound $r_{p,L}$ of minimum adversarial perturbation $r_{p,\text{min}}$. Indeed, current robustness verification literature mainly focuses on efficient methods for finding a tight *lower bound* $r_{p,L}$ on minimum distortion $r_{p,\text{min}}$ as a certified local robustness metric. Again, the term *certified* means that numerical values generated by these approaches are indeed deterministic lower bounds of minimum adversarial distortion, such as the robustness bounds delivered by CROWN algorithm.

However, the scope of major robustness verification literature is still limited to the *worst-case* setting (i.e., no exceptions are allowed), and hence there lacks proper robustness indication once the perturbation exceeds the certified range. In particular, in cases when successful adversarial perturbations are rare events (but not totally unlikely) or when some distributional characterization of input perturbations are known a priori, a *probabilistic* certificate on the confidence of local certified robustness level is more viable and comprehensive. In this chapter, we are interested in answering the following questions:

(a) How to provide a confidence level on the robustness certificate when the input data point is perturbed by random noises?

(b) How does such *probabilistic* certificate compare with the certificate from worst-case analysis?

Perhaps surprisingly, in our experiments we find that simple random perturbations can already achieve up to 100% misclassification rates on MNIST and CIFAR networks under reasonable perturbation ranges (see Table 5.1), suggesting that their threats to deep neural networks could be overlooked. With prior knowledge on the input noise distributions, our probabilistic approach is able to provide a more comprehensive robustness quantification in comparison to the prevailing worst-case analysis. More importantly, the probabilistic robustness quantification is readily applicable to understanding the sensitivity and reliability of a target model in many real-world scenarios. For example, such random noises can be caused by data quantization, input preprocessing, or environmental background noises.

Thus, we propose a novel probabilistic framework PROVEN to **PRO**babilistically **VE**rify **N**eural network robustness. We show that it is possible to extend the conventional worst-case setting to a probabilistic setting based on existing worst-case certification frameworks with very little computational overhead, meaning that the probabilistic certificate can be easily derived and incorporated into worst-case robustness verification pipelines by methods such as Fast-Lin [101], CROWN [106], and CNN-Cert [8].

5.1.1 Our Contributions

1. A probabilistic framework PROVEN is proposed for verifying the robustness of neural networks with certificates under ℓ_p norm-ball bounded threat models, when the input noise follows a given distributional characterization (Gaussian and Sub-Gaussian distributions with bounded supports). The established theoretical results are based on an ℓ_∞ constraint on the perturbation but can be easily extended to other norms such as ℓ_1 and ℓ_2.

2. Experimental results on large neural networks trained on MNIST and CIFAR datasets show that PROVEN greatly improved the robustness certificate (i.e., the certified robustness lower bound of minimum adversarial distortion) compared with the corresponding worst-case analysis results, even when the statistical risk is small. For example, with a confidence level of 99.99%, which means the robustness metric is almost 100% guaranteed to be certified, the robustness certificate improvement provided by PROVEN over worst-case analysis can be as high as 250%.

3. In addition to the noticeable improvement in the verified robustness, PROVEN is a general tool that can be readily applied to neural networks with general activation functions, including but not limited to tanh, sigmoid and arctan, and general convolutional neural networks with various building blocks, as demonstrated in our experiments. Moreover, PROVEN is as computationally efficient as the worst-case analysis because its closed-form probabilistic certificate are by-products of worst-case certification frameworks such as Fast-Lin [101], CROWN [106], and CNN-Cert [8].

5.1.2 Related Works

Recent studies have shown that solving for a non-trivial lower bound $r_{p,L}$ of the minimum adversarial perturbation $r_{p,\text{min}}$ can be made more scalable and computationally efficient [101, 36, 26]. Some analytical lower bounds, based solely on model weights, have been derived [94, 80, 41, 82] but they are either very close to trivial lower bounds (zero) or only applied to 1 or 2 hidden layers. It is worth noting that current robustness verification approaches mainly focus on "worst-case" analysis, whereas our approach takes a "probabilistic viewpoint". As demonstrated in extensive experiments, our probabilistic framework PROVEN is able to certify much larger $r_{p,L}$ value than the corresponding worst-case analysis with 99.99% certification guarantees. This indicates that, while the conventional worst-case robustness certificate may be too conservative when there is some prior knowledge about the input perturbations distribution,

PROVEN is more applicable in this situation.

In fact, deep neural networks are not only vulnerable to crafted adversarial noises but also to random noises: [7] show that they can fool LeNet and AlexNet with additive Gaussian noises and [32] show random perturbations can indeed fool VGG networks; [43] show they fool the Google Cloud Vision API by random Gaussian noises, suggesting random perturbation can result in successful attacks. Meanwhile, the robustness of classifiers to various kinds of random noises, such as uniform noise in the ℓ_p unit ball and Gaussian noise with an arbitrary covariance matrix, has been studied in [34]. Their analysis, however, requires the assumption of locally approximately flat decision boundaries on the neural networks, which is difficult to verify in reality. Recently, the robustness of classifiers to perturbations under the assumption of Gaussian distributed latent input vectors has been studied in [30]; however, their results depend on the modulus of continuity constant, which could be arbitrarily large. Due to these limitations, the bounds in these papers cannot be directly used to deliver certified local robustness metrics.

We note that only a few works have considered verifying neural network properties in the probabilistic setting. [25] verifies some properties (e.g., monotonicity and convexity) of deep generative models. The key differences are that in their setting the uncertainty source is the latent variable sampled from a distribution generated by the encoder (they consider only Gaussian distribution), whereas our uncertainty comes from input perturbations (we consider both Gaussian and general Subgaussian distributions with bounded support) and our focus is to verify neural network classifiers. A very recent work [100] uses a Monte Carlo approach to *estimate* the probability of rare events, whereas the violation probability from our framework is certifiable. We also highlight that the problem setting of this work is different from [59, 63] as our goal is to study the effect of random noises on *standard* neural networks input and derive a closed-form probabilistic certificate; on the other hand, they study *smoothed* (or stochastic) classifiers by adding a noise layer into the network and thus modifying the prediction rule. The predicted class in their setting becomes the class that has the largest expected score [59] or the largest probability [63] over input with noises,

which is different from the rule of standard networks.

5.2 PROVEN: A Probabilistic Framework for Verifying Neural Network Robustness

In this Chapter, we present a general probabilistic framework PROVEN together with related theoretical results to compute the certified local robustness bounds in probability that a classifier can never be fooled when the inputs of the classifier are perturbed with some given distributions. We first introduce the worst-case setting, where an input example can be perturbed by any perturbation bounded within an ℓ_p ball, and present corresponding worst-case analysis results in Sec. 5.2.1. We then show that it is possible to extend these worst-case analysis results to a probabilistic setting where the input perturbations follow some given distributions, and present our probabilistic framework and main theorem in Sec. 5.2.2. Lastly, we provide *closed-form* probabilistic bounds for various probabilistic distributions that the input perturbations can follow in Section 5.2.3.

5.2.1 Worst-case setting

Let $f(\mathbf{x}) : \mathbb{R}^{n_0} \to \mathbb{R}^K$ denote a K-class neural network classifier of interest, which takes an input $\mathbf{x}$ (e.g., an image) and outputs the corresponding logit scores over all classes. The ultimate goal is to efficiently find the largest $r_{p,L}$ such that the original predicted class c always has a larger score $f_c(\mathbf{x})$ than the score $f_t(\mathbf{x})$ for a targeted attack class t when the input is perturbed within the ℓ_p ball having radius $r_{p,L}$. Let $g_t(\mathbf{x}) = f_c(\mathbf{x}) - f_t(\mathbf{x})$, we want to find the largest $r_{p,L}$ such that $g_t(\mathbf{x}) > 0$ for all $\mathbf{x}$ satisfying $\|\mathbf{x} - \mathbf{x_0}\|_p \leq r_{p,L}$ and $c = \text{argmax}_i f_i(\mathbf{x_0}), t \neq c$. This $r_{p,L}$ is a *certified local robustness lower bound* of the minimum adversarial distortion introduced in Sec. 5.1.

Although a neural network classifier f is a *nonlinear*, *non-convex* and complicated function, it has been first shown in Fast-Lin [101] and later in CROWN [106] and CNN-Cert [8] that the output $f_i(\mathbf{x})$ and the margin function $g_t(\mathbf{x})$ of a general (convolutional) neural network classifier with general activation functions[1] can be bounded by two

[1] including but not limited to ReLU, tanh, arctan, sigmoid

linear functions $g_t^L(\mathbf{x})$ and $g_t^U(\mathbf{x})$:

$$g_t^L(\mathbf{x}) \leq g_t(\mathbf{x}) \leq g_t^U(\mathbf{x}), \tag{5.1}$$

where $g_t^L(\mathbf{x}) : \mathbb{R}^{n_0} \to \mathbb{R}$ and $g_t^U(\mathbf{x}) : \mathbb{R}^{n_0} \to \mathbb{R}$ are two linear functions in terms of the input $\mathbf{x}$:

$$g_t^L(\mathbf{x}) = \mathbf{A}_{t,:}^L \mathbf{x} + d^L \quad \text{and} \quad g_t^U(\mathbf{x}) = \mathbf{A}_{t,:}^U \mathbf{x} + d^U \tag{5.2}$$

with $\mathbf{A}_{t,:}^L, \mathbf{A}_{t,:}^U \in \mathbb{R}^{1 \times n_0}$ being two constant row vectors and $d^L, d^U \in \mathbb{R}$ being two constants related to the network weights $\mathbf{W}^{(k)}$ and biases $\mathbf{b}^{(k)}$ as well as the parameters bounding the activation functions in each neuron. The superscripts L and U denote the parameters corresponding to the lower bound and the upper bound of $g_t(\mathbf{x})$.

Remark. We would like to highlight again that, same as Fast-Lin, CROWN and CNN-Cert, we *do not* assume the neural network $f(\mathbf{x})$ and its variant $g_t(\mathbf{x})$ to be linear. Through out this chapter, $f(\mathbf{x})$ and $g_t(\mathbf{x})$ are a general (convolutional) neural network with general non-linear activation function and thus are always **non-linear** and **non-convex** functions. The core idea of their proposed worst-case frameworks is to use the **linear upper and lower bound** $g_t^U(\mathbf{x}), g_t^L(\mathbf{x})$ to **bound the non-linear neural network** $g_t(\mathbf{x})$, as in Equation (5.1). Again, the *linear* bounds have superscripts L and U while the non-linear neural network functions does not. Importantly, the result in Equation (5.1) is the key to develop our probabilistic framework PROVEN. We also highlight that $\mathbf{A}^L = \mathbf{A}^U$ in Fast-Lin due to the use of parallel linear bounds on non-linear activation functions, whereas $\mathbf{A}^L$ and $\mathbf{A}^U$ can be different in CROWN and CNN-Cert due to non-parallel linear bounds.

As the network output is bounded, the positiveness of the lower bound of $g_t(\mathbf{x})$ implies that $g_t(\mathbf{x})$ is positive, i.e.,

$$g_t^L(\mathbf{x}) > 0 \implies g_t(\mathbf{x}) > 0. \tag{5.3}$$

Here, a *worst-case* analysis can be performed by minimizing the linear function $g_t^L(\mathbf{x})$

over all possible inputs in the set $\{\mathbf{x} : \|\mathbf{x} - \mathbf{x_0}\|_p \leq \epsilon\}$, which yields a closed-form solution as presented in Fast-Lin, CROWN and CNN-Cert. Therefore, the condition of whether $g_t^L(\mathbf{x}) > 0$ can be conveniently checked given some ϵ using the closed-form solutions; the largest ϵ such that $g_t^L(\mathbf{x}) > 0$ is called the *certified local robustness radius*, which can be computed by bisection with respect to ϵ.

5.2.2 Our proposed probabilistic framework: PROVEN

In addition to considering the worst-case condition for $g_t(\mathbf{x}) > 0$ over the norm ball constrained on the input $\{\mathbf{x} : \|\mathbf{x} - \mathbf{x_0}\|_p \leq \epsilon\}$, we now show that it is possible to formulate a probabilistic setting and derive bounds with guarantees by building upon the above results that the neural network output can be bounded by two linear functions. We start by presenting the problem formulation of the probabilistic setting and then present our main theoretical results in Theorem 5.2.1.

Problem formulation. Consider a general **non-linear** neural network classifier $f(\mathbf{x})$ and an input example $\mathbf{x_0}$. Let the predicted class of $\mathbf{x_0}$ be c, the targeted attack class t, and the **non-linear** margin function $g_t(\mathbf{x}) := f_c(\mathbf{x}) - f_t(\mathbf{x})$. Suppose the perturbed input random vector X follows some given distribution $\mathcal{D}$, i.e., $X \sim \mathcal{D}$. We are interested in the probability of the margin function $g_t(\mathbf{x})$ being greater than some value $a \in \mathbb{R}$, i.e., $\mathbb{P}[g_t(X) > a]$.

Given that the general neural networks are highly **non-linear** and **non-convex** in $\mathbf{x}$, it is hard to directly compute the distribution of the **non-linear** function $g_t(X)$ given the input $X \sim \mathcal{D}$. Fortunately, we can still derive *analytic lower and upper bounds* for $\mathbb{P}[g_t(X) > 0]$ based on the **linear** function $g_t^L(X)$ thanks to the relations of **linear** $g_t^L(\mathbf{x})$ and **non-linear** $g_t(\mathbf{x})$ in Eq. (5.1). In other words, we can obtain probabilistic guarantees based on the worst-case analysis result where the **non-linear** margin function $g_t(x)$ can be bounded by two **linear** functions. The following theorem provides such theoretical guarantees on $\mathbb{P}[g_t(X) > 0]$.

Theorem 5.2.1 (Probabilistic bounds of network output)**.** *Let $f(\mathbf{x})$ be a K-class neural network classifier function, $\mathbf{x_0}$ an input example, and ϵ such that $\|\mathbf{x} - \mathbf{x_0}\|_p \leq \epsilon$,*

$p \geq 1$. *Let* $c = argmax_i f_i(\mathbf{x_0})$, *let* $t(\neq c)$ *be some targeted class, and define the margin function* $g_t(\mathbf{x}) = f_c(\mathbf{x}) - f_t(\mathbf{x})$. *Suppose the input random vector* $X \in \mathbb{R}^{n_0}$ *follows some given distribution* $\mathcal{D}$ *with mean* $\mathbf{x_0}$ *and let* $a \in \mathbb{R}$ *be some real number. There exists an explicit lower bound* γ_L *and an explicit upper bound* γ_U *on the probability* $\mathbb{P}[g_t(X) > a]$ *such that*

$$\gamma_L \leq \mathbb{P}[g_t(X) > a] \leq \gamma_U, \tag{5.4}$$

where

$$\gamma_L = 1 - F_{g_t^L(X)}(a), \quad \gamma_U = 1 - F_{g_t^U(X)}(a), \tag{5.5}$$

$F_Z(z)$ *is the cumulative distribution function (CDF) of the random variable* Z, *and* $g_t^L(\mathbf{x})$, $g_t^U(\mathbf{x})$ *satisfy Equation* (5.1).

Proof. Let $h_1 : \mathbb{R}^d \to \mathbb{R}$, $h_2 : \mathbb{R}^d \to \mathbb{R}$, and $h_1(\mathbf{x}) \geq h_2(\mathbf{x}), \forall \mathbf{x} \in \mathbb{R}^d$. Let $X \in \mathbb{R}^d$ be a random vector. Since the cumulative distribution function (CDF) is nondecreasing [88], we have

$$\mathbb{P}[h_1(X) > a] \geq \mathbb{P}[h_2(X) > a]. \tag{5.6}$$

From the results in [101], we know that the relationships in Eq. (5.1), i.e.,

$$g_t^L(\mathbf{x}) \leq g_t(\mathbf{x}) \leq g_t^U(\mathbf{x}),$$

satisfy $\|\mathbf{x} - \mathbf{x_0}\|_p \leq \epsilon$ for all $\mathbf{x}$. Hence, upon applying Eq. (5.6) to Eq. (5.1) and using the fact that $\mathbb{P}[Z > a] = 1 - F_Z(a)$, we obtain

$$\gamma_L \leq \mathbb{P}[g_t(X) > a] \leq \gamma_U,$$

with $\gamma_L = 1 - F_{g_t^L(X)}(a)$, $\gamma_U = 1 - F_{g_t^U(X)}(a)$. □

As discussed in Section 5.2.1, the neural network output and the margin function can be bounded by two linear functions as derived in Fast-Lin and its extensions CROWN and CNN-Cert. Here, we take an additional step to investigate the relationship between the margin function and its linear bounds in the probabilistic setting. Specifically,

Theorem 5.2.1 shows that the probability of the neural network margin function being greater than some value a can also be bounded by the CDFs of its linear bounds. Note that in the worst-case analysis of Section 5.2.1, we usually concern ourselves with the margin function $g_t(x) > 0$, i.e., $a = 0$. Analogously, in the probabilistic setting, we concern ourselves with the probability of the margin function $g_t(x) > 0$. This is indeed the guarantee provided by Theorem 5.2.1: when the input $X \sim \mathcal{D}$, the result guarantees that the probability of $g_t(X) > a$ is at least γ_L and at most γ_U.

5.2.3 Evaluating the probabilistic bounds γ_L and γ_U

Theorem 5.2.1 provides a theoretical lower bound γ_L and upper bound γ_U for $\mathbb{P}[g_t(X) > a]$. In practice, we would like to numerically compute such bounds. Below we show it is possible to obtain explicit forms for γ_L and γ_U in terms of $\mathbf{A}^L_{t,;}, \mathbf{A}^U_{t,;}, d^L, d^U$, as well as the parameters of the probability distributions of input perturbations. By Theorem 5.2.1, γ_L and γ_U only depend on the CDFs $F_{g_t^L(X)}$ and $F_{g_t^U(X)}$, and we observe that $g_t^L(X)$ and $g_t^U(X)$ are both linear functions of random vector X as follows:

$$g_t^L(X) = \sum_{i=1}^{n_0} \mathbf{A}^L_{t,i} X_i + d^L,$$
$$g_t^U(X) = \sum_{i=1}^{n_0} \mathbf{A}^U_{t,i} X_i + d^U.$$

Hence, the problem of computing the CDFs $F_{g_t^L(X)}$ and $F_{g_t^U(X)}$ becomes a problem of computing the CDFs of a weighted sum of X_i given $X \sim \mathcal{D}$. We primarily consider the following two cases:

(i) When X_i are independent random variables with probability density function (PDF) f_{X_i};

(ii) When X follows a multivariate normal distribution with mean $\mathbf{x_0}$ and covariance Σ.

It also appears that these results may be extended to address some forms of negative correlation [74, 24].

Case (i)

When X_i are independent random variables with probability density function f_{X_i}, there are two approaches for computing the CDFs of the weighted sum.

Approach 1: Direct convolutions. The PDF of the weighted sum is simply the convolution of the PDFs for each of the weighted random variables $\mathbf{A}^L_{t,i}X_i$. Specifically, we have

$$f_{\mathbf{A}^L_{t,:}X} = \circledast_{i=1}^{n_0} f_{\mathbf{A}^L_{t,i}X_i},$$

where $\circledast_{i=1}^{N} h_i$ denotes convolution over the N functions h_1 to h_N. The CDF of $\mathbf{A}^L_{t,:}X$ can therefore be obtained from the PDF $f_{\mathbf{A}^L_{t,:}X}$ and we obtain $F_{g^L_t(X)}(z) = F_{\mathbf{A}^L_{t,:}X}(z - d^L)$; similarly, $F_{g^U_t(X)}(z) = F_{\mathbf{A}^U_{t,:}X}(z - d^U)$. Hence, we have

$$\gamma_L = 1 - F_{\mathbf{A}^L_{t,:}X}(a - d^L), \quad \gamma_U = 1 - F_{\mathbf{A}^U_{t,:}X}(a - d^U).$$

Approach 2: Probabilistic inequalities. Approach 1 is useful in cases where n_0 is not large. However, for large n_0, it might not be easy to directly compute the CDF through convolutions. For such cases, an alternative approach can be based on applying the probabilistic inequalities on the CDFs. Since we want to provide *guarantees* on the desired probability in (5.4), we need to find a lower bound on γ_L and an upper bound on γ_U via the probabilistic inequalities. These results are given in the following corollary.

Corollary 5.2.2. *Let X_i be independent random variables following Sub-Gaussian distributions with bounded support $X_i \in [\mathbf{x_0}_i - \epsilon, \mathbf{x_0}_i + \epsilon], \forall i \in [n_0]$, and symmetric around the mean $\mathbf{x_0}_i$. Define*

$$\mu_L = \mathbf{A}^L_{t,:}\mathbf{x_0} + d^L, \ \mu_U = \mathbf{A}^U_{t,:}\mathbf{x_0} + d^U.$$

Then, we have

$$\gamma_L \geq \begin{cases} 1 - \exp\left(-\frac{(\mu_L - a)^2}{2\epsilon^2 \|\mathbf{A}^L_{t,:}\|_2^2}\right), & \text{if } \mu_L - a \geq 0 \\ 0, & \text{otherwise;} \end{cases}$$

$$\gamma_U \leq \begin{cases} \exp\left(-\frac{(\mu_U - a)^2}{2\epsilon^2 \|\mathbf{A}^U_{t,:}\|_2^2}\right), & \text{if } -\mu_U + a \geq 0 \\ 1, & \text{otherwise.} \end{cases}$$

Proof. Let $W_i = \mathbf{A}^L_{t,i}(X_i - \mathbf{x_0}_i)$ and $\mu_L = \mathbf{A}^L_{t,:}\mathbf{x_0} + d^L$. We then have $-|\mathbf{A}^L_{t,i}|\epsilon \leq W_i \leq |\mathbf{A}^L_{t,i}|\epsilon$ where W_i is symmetric with respect to zero since X_i is symmetric. By using the fact that the sum of independent symmetric random variables is still a symmetric random variable [19], we derive

$$\begin{aligned} \gamma_L &= \mathbb{P}\left[g^L_t(X) > a\right] \\ &= \mathbb{P}\left[\sum_{i=1}^{n_0} W_i > a - \mu_L\right] \\ &= \mathbb{P}\left[\sum_{i=1}^{n_0} W_i < -a + \mu_L\right] \\ &= 1 - \mathbb{P}\left[\sum_{i=1}^{n_0} W_i \geq -a + \mu_L\right]. \end{aligned}$$

From the Hoeffding inequality [85], we obtain the following upper bound on the term $\mathbb{P}\left[\sum_{i=1}^{n_0} W_i \geq -a + \mu_L\right]$ when $-a + \mu_L > 0$:

$$\mathbb{P}\left[\sum_{i=1}^{n_0} W_i \geq -a + \mu_L\right] \leq \exp\left(-\frac{(\mu_L - a)^2}{2\epsilon^2 \|\mathbf{A}^L_{t,:}\|_2^2}\right),$$

and thus $\gamma_L \geq 1 - \exp\left(-\frac{(\mu_L - a)^2}{2\epsilon^2 \|\mathbf{A}^L_{t,:}\|_2^2}\right)$. When $-a + \mu_L \leq 0$, we use the trivial bound of $\gamma_L = 0$. Similarly, for γ_U, we can define μ_U correspondingly and directly apply the Hoeffding inequality to obtain $\gamma_U \leq \exp\left(-\frac{(\mu_U - a)^2}{2\epsilon^2 \|\mathbf{A}^U_{t,:}\|_2^2}\right)$, or use the trivial bound of $\gamma_U = 1$. □

Remark. The above result can be further extended to correlated random variables with Hoeffding inequality similarly with the assumption that the sum of W_i is symmetric.

Case (ii)

When X follows a multivariate normal distribution with mean $\mathbf{x_0}$ and covariance Σ, we are able to obtain an explicit form for the CDFs $F_{g_t^L(X)}$ and $F_{g_t^U(X)}$ based on the fact that the sum of normally distributed random variables still follows the normal distribution [19]. Note that we include here both cases where (a) X_i are independent Gaussian random variables (Σ is a diagonal matrix) and (b) X_i are correlated random variables (Σ is a general covariance matrix and positive semidefinite). Our result is stated in the following corollary.

Corollary 5.2.3. *Let X follow a multivariate normal distribution with mean $\mathbf{x_0}$ and covariance Σ. Define*

$$\mu_L = \mathbf{A}_{t,:}^L \mathbf{x_0} + d^L,\ \sigma_L^2 = \mathbf{A}_{t,:}^L \Sigma (\mathbf{A}_{t,:}^L)^\top,$$
$$\mu_U = \mathbf{A}_{t,:}^U \mathbf{x_0} + d^U,\ \sigma_U^2 = \mathbf{A}_{t,:}^U \Sigma (\mathbf{A}_{t,:}^U)^\top,$$

where $\top$ denotes the transpose operator. We then have

$$\gamma_L \approx \frac{1}{2} - \frac{1}{2} erf(\frac{a-\mu_L}{\sigma_L \sqrt{2}}), \quad \gamma_U \approx \frac{1}{2} - \frac{1}{2} erf(\frac{a-\mu_U}{\sigma_U \sqrt{2}})$$

with $erf(\cdot)$ as the error function.

Proof. The result is obtained in a straightforward manner from the fact [19] that if $X \sim \mathcal{N}(\mu, \Sigma)$, then its linear combination $Z = wX + v$ also follows the normal distribution: $Z \sim \mathcal{N}(w\mu + v, w\Sigma w^\top)$. The CDF of Z is then given by $\frac{1}{2}(1 + \text{erf}(\frac{z-\mu_Z}{\sigma_Z \sqrt{2}}))$, leading to the stated approximations. □

Remark. Note that in our framework, all possible inputs have to lie in the ℓ_p ball with given radius ϵ. Thus, in order to apply the Gaussian perturbation in our setting,

we need to set an upper limit on the variance of the input such that 99.7% of the density is within the ℓ_p ball, i.e., the 3-σ rule. See Section 5.3 **Experiment (b)** for more details.

Connection to ℓ_1 and ℓ_2 norms. Our foregoing probabilistic analysis is established under the ℓ_∞ norm constraint. We note that this presented analysis can be easily extended to ℓ_1 and ℓ_2 norms by using the norm inequalities: $\|\mathbf{x}\|_1 \leq \sqrt{n_0}\|\mathbf{x}\|_2 \leq n_0\|\mathbf{x}\|_\infty$.

5.3 Experimental Results

In this section, we conduct two major experiments:

(a) attack neural network models with random noises

(b) calculate the local robustness lower bounds certified by PROVEN with various confidence levels

The goal of the Experiment (a) is to validate the issues that neural networks are also vulnerable to random noises in addition to the specifically crafted adversarial noises, and Experiment (b) is to demonstrate the effectiveness and capability of our proposed probabilistic framework PROVEN and compare with the conservative worst-case certifications.

Model, Dataset, and Setup. We use the pre-trained MNIST and CIFAR networks provided in [101] and [106]. The MLP networks for MNIST 2-4 layers have 1024 hidden nodes per layer unless otherwise noted. We also have MNIST 2 and 3 layers with 20 nodes, denoted as '(20)' in the Tables. For CIFAR MLP networks, 5 and 6 layers models have 2048 hidden nodes per layer, and 7 layers models have 1024 hidden nodes per layer. In addition to the pre-trained fully-connected models, we also train (1) 2 layer and 3 layer MNIST CNNs with 5 filters and ReLU, tanh, sigmoid

Table 5.1: **Attacks with Uniform & Bernoulli noises**: success rate over 100 randomly selected images.

Perturbed ℓ_∞ magnitude	$\epsilon = 0.25$		$\epsilon = 0.20$	
MNIST model	Uniform	Bernoulli	Uniform	Bernoulli
2-layer CNN, ReLU	19%	72%	9%	35%
2-layer CNN, tanh	50%	94%	28%	81%
2-layer CNN, sigmoid	17%	68%	7%	47%
2-layer CNN, arctan	61%	90%	48%	84%
3-layer CNN, ReLU	9%	54%	6%	25%
3-layer CNN, tanh	45%	97%	37%	83%
3-layer CNN, sigmoid	22%	82%	14%	58%
3-layer CNN, arctan	0%	91%	23%	72%
Perturbed ℓ_∞ magnitude	$\epsilon = 0.025$		$\epsilon = 0.020$	
CIFAR model	Uniform	Bernoulli	Uniform	Bernoulli
5×[2048], ReLU	7%	8%	5%	7%
6×[2048], ReLU	11%	20%	6%	17%
5-layer CNN, ReLU	21%	37%	16%	30%

and arctan activations (2) MNIST ResNet with 3 residual blocks (3) CIFAR 5-layer CNN. In addition, we also train some *robust* models by adversarial training [71] with 2 and 3 layer CNN structures. All the data are normalized to the range $[-0.5, 0.5]$. We implement PROVEN in Python and perform experiments on a laptop with 8 Intel Cores i7-4700 HQ CPU at 2.40 GHz and a AMD Zen sever CPU. Our code is available at https://github.com/lilyweng/PROVEN.

Experiment (a). We generate random noises with range $[-\epsilon, \epsilon]$ following uniform distribution and Bernoulli distribution with coin probability 0.5, and apply the random noises on each pixel. We generate at most 10^4 samples to perform random attacks on 100 randomly selected test images with correct prediction. We report the success rate and the corresponding ϵ for various MNIST and CIFAR (convolution) neural networks in Table 5.1. The results show that random noises can indeed achieve successful attack with surprisingly rates, up to 98%, thus validating the issues of random noises and the necessity to analyze its effects, as one of the main purpose of this work. We note that for $\epsilon = 0.25$, the attacks on CIFAR models almost all succeed and thus we decrease the ϵ by ten-fold.

Experiment (b). We apply Corollaries 5.2.2 and 5.2.3 (i.e. Approach 2 in Sec. 5.2.3) to compute the largest ϵ (denoted as $r_{p,L,\text{PROVEN}}$, note $p = \infty$ in all experiments) that PROVEN can certify with confidence of at least γ_L when the input follows the two cases discussed in Section 5.2.3. The certified local robustness radius computed by the worst-case analysis in Fast-Lin [101], CROWN [106] and CNN-Cert [8] is denoted as $r_{p,L,\text{worst-case}}$. Below is the setting of the input distributions in our simulations:

- **Case (i)**. X_i are independent random variables following Sub-Gaussian distribution with bounded support $[\mathbf{x_0}_i - r_{p,L,\text{worst-case}}, \mathbf{x_0}_i + r_{p,L,\text{worst-case}}]$ with mean $\mathbf{x_0}_i$. The results are presented in Table 5.2.

- **Case (ii)**. X follows a multivariate normal distribution with mean $\mathbf{x_0}$ and covariance Σ. We consider both situations where Σ is a positive diagonal matrix

or a positive semidefinite matrix with diagonals whose square roots are less than or equal to $r_{p,L,\text{worst-case}}/3$. The results are presented in Tables 5.6 and 5.5.

Note that in all the Tables, we express γ_L as a percentage. We report $r_{p,L,\mathsf{PROVEN}}$ for the following values: $\{99.99, 75, 50, 25, 5\}\%$ and calculate the improvement of $r_{p,L,\mathsf{PROVEN}}$ over $r_{p,L,\text{worst-case}}$ obtained by 99.99% in the last column in Table 5.2 for Case (i). The results in Table 5.2 are averaged over 10 randomly selected images in the test sets for MLP networks and 100 images for the CNN networks. On the other hand, we also investigate how robust it is for the results in Table 5.2 by computing the average $r_{p,L,\mathsf{PROVEN}}$ over randomly chosen $\{10, 50, 100\}$ images in 100 random trials. We report the mean and standard deviation in Table 5.3 and show that the variation of using 10 sample average in Table 5.2 is less $\sim 10\%$ and the average $r_{p,L,\mathsf{PROVEN}}$ and improvement has less deviations when we use 50 or 100 samples.

Result on small and large ReLU networks. We perform simulations on both small 2-3 layer MNIST networks with 20 neurons per layer and large 2-7 layer MNIST and CIFAR networks with 1024 or 2048 neurons per layer; the full results are summarized in Tables 5.2. These results show that on the small networks, PROVEN can certify up to 1.7× with respect to the certified local robustness radius at the expense of decreasing the confidence by only $\eta = 10^{-2}$. In other words, PROVEN guarantees that at least 99.99% of the ϵ computed (e.g., 0.02746 in MNIST 2×[20] is a certified local robustness radius as compared to 0.04620 for the $r_{p,L,\text{worst-case}}$ delivered by CROWN [106], where the improvement we obtained for this model is 1.7×. For large MLP networks, PROVEN can certify up to 1.3×, which is significant.

Results on large networks with general activations and CNNs. We also ran experiments on various MNIST and CIFAR networks with non-ReLU activations, e.g., tanh, sigmoid and arctan and CNN structures. The results are summarized in Table 5.2. In comparison to the same architecture but with ReLU activations, the improvement of these activations are better than the non-adaptive bounding technique in general, and can achieve up to 260% on CNN networks. Note that the computational

overhead of our approach compared to the worst-case certification frameworks Fast-Lin, CROWN and CNN-Cert is very little, as we only need to perform a few additional binary searches on the ϵ that will satisfy Corollary 5.2.2.

Table 5.2: The largest ϵ that PROVEN can certify with confidence of at least $\gamma_L = \{99.99, 75, 50, 25, 5\}\%$ when X_i are independent random variables in Case (i). We compare the largest ϵ that PROVEN can certify with 99.99% with the largest ϵ from state-of-the-art worst-case certification algorithms CROWN [106] (for MLPs), CNN-Cert [8] (for CNNs) and show in the last column that PROVEN can certify more than the worst-case analysis by giving up 0.01% confidence.

Certification Method	Worst-case	Our probabilistic approach: PROVEN					Certification
Guarantees γ_L	100%†	99.99%†	75%	50%	25%	5%	Bound increase†
MNIST 2-layer arctan	0.02105	0.02756	0.02869	0.02893	0.02913	0.02935	30.93%
MNIST 2-layer relu	0.03158	0.04064	0.04256	0.04293	0.04326	0.04360	28.69%
MNIST 2-layer relu (20)	0.02746	0.04620	0.05107	0.05205	0.05293	0.05387	68.24%
MNIST 2-layer sigmoid	0.02785	0.03805	0.04032	0.04083	0.04124	0.04174	36.62%
MNIST 2-layer tanh	0.02232	0.02754	0.02857	0.02876	0.02892	0.02906	23.39%
MNIST 3-layer relu	0.02397	0.02693	0.02747	0.02757	0.02766	0.02775	12.35%
MNIST 3-layer relu (20)	0.02236	0.02949	0.03092	0.03120	0.03145	0.03171	31.89%
MNIST 3-layer sigmoid	0.01856	0.02190	0.02253	0.02266	0.02278	0.02288	18.0%
MNIST 3-layer tanh	0.01121	0.01224	0.01241	0.01244	0.01247	0.01250	9.19%
MNIST 4-layer arctan	0.00726	0.00763	0.00769	0.00770	0.00771	0.00772	5.1%
MNIST 4-layer relu	0.00962	0.00993	0.00998	0.00999	0.01000	0.01001	3.22%
MNIST 4-layer tanh	0.00682	0.00708	0.00712	0.00712	0.00713	0.00714	3.81%
MNIST CNN 2-layer	0.04163	0.05648	0.06069	0.06154	0.06229	0.06310	35.67%
MNIST CNN 2-layer, atan	0.03115	0.11344	0.16911	0.18289	0.19587	0.21040	264.17%
MNIST CNN 2-layer, sigmoid	0.09606	0.18939	0.22177	0.22873	0.23521	0.24186	97.16%
MNIST CNN 2-layer, tanh	0.03050	0.09464	0.13029	0.13872	0.14635	0.15490	210.3%
MNIST CNN 3-layer	0.04259	0.05786	0.06189	0.06269	0.06340	0.06415	35.85%
MNIST CNN 3-layer, atan	0.02637	0.05218	0.05970	0.06129	0.06268	0.06396	97.88%
MNIST CNN 3-layer, sigmoid	0.07845	0.13611	0.15348	0.15671	0.15947	0.16254	73.5%
MNIST CNN 3-layer, tanh	0.02933	0.05331	0.06078	0.06235	0.06367	0.06510	81.76%
CIFAR 5-layer relu	0.00228	0.00234	0.00234	0.00234	0.00234	0.00234	2.63%
CIFAR 5-layer tanh	0.00081	0.00084	0.00084	0.00084	0.00084	0.00084	3.7%
CIFAR 7-layer relu	0.00189	0.00192	0.00192	0.00193	0.00193	0.00193	1.59%
CIFAR CNN 5-layer	0.00420	0.00490	0.00496	0.00497	0.00498	0.00499	16.67%
MNIST 2-layer relu, adv	0.02073	0.02965	0.03133	0.03164	0.03192	0.03221	43.03%
MNIST 3-layer relu, adv	0.18605	0.18793	0.20147	0.20460	0.20748	0.21081	1.01%
MNIST CNN 2-layer, adv	0.08848	0.10714	0.11506	0.11665	0.11805	0.11954	21.09%
MNIST CNN 2-layer, tanh adv	0.05329	0.09615	0.12356	0.12991	0.13580	0.14237	80.43%
MNIST CNN 3-layer, adv	0.10377	0.11856	0.12412	0.12512	0.12598	0.12689	14.25%
MNIST CNN 3-layer, tanh adv	0.04064	0.12065	0.16757	0.17794	0.18673	0.19624	196.87%

Table 5.3: With input perturbations being independent random variables in case (i), we perform $\{10, 50, 100\}$ random trials to randomly choose $\{10, 50, 100\}$ input samples (images) in each trial and then compute the average of the largest ϵ that can be certified by worst-case analysis [8] (denoted as $r_{p,L,\text{worst-case}}$) and by PROVEN with 99.99% confidence (denoted as $r_{p,L,\text{PROVEN}}$) together with the improved certification of $r_{p,L,\text{PROVEN}}$ over $r_{p,L,\text{worst-case}}$. The mean and std converge as the number of samples increases.

			$r_{p,L,\text{worst-case}}$		$r_{p,L,\text{PROVEN}}$	
			mean	std	mean	std
model	# samples	# trials				
MNIST 3layer relu	10	10	0.025954	0.002120	0.028899	0.002280
		50	0.026053	0.001786	0.028982	0.001915
		100	0.025976	0.001676	0.028913	0.001772
	50	10	0.025870	0.000630	0.028814	0.000737
		50	0.025723	0.000789	0.028647	0.000841
		100	0.025668	0.000712	0.028593	0.000756
	100	10	0.025654	0.000512	0.028577	0.000564
		50	0.025716	0.000465	0.028645	0.000486
		100	0.025668	0.000457	0.028603	0.000483
MNIST 3layer tanh	10	10	0.011712	0.000631	0.012776	0.000586
		50	0.011742	0.000691	0.012798	0.000650
		100	0.011793	0.000676	0.012840	0.000636
	50	10	0.011895	0.000284	0.012932	0.000282
		50	0.011855	0.000281	0.012893	0.000269
		100	0.011815	0.000284	0.012854	0.000271
	100	10	0.011884	0.000119	0.012920	0.000121
		50	0.011865	0.000174	0.012900	0.000166
		100	0.011831	0.000191	0.012868	0.000181
MNIST 4layer relu	10	10	0.009566	0.000575	0.009852	0.000584
		50	0.009822	0.000502	0.010114	0.000511
		100	0.009833	0.000545	0.010127	0.000552
	50	10	0.009871	0.000317	0.010166	0.000321
		50	0.009916	0.000249	0.010210	0.000255
		100	0.009891	0.000247	0.010185	0.000250
	100	10	0.009859	0.000182	0.010153	0.000184
		50	0.009903	0.000141	0.010197	0.000144
		100	0.009903	0.000156	0.010197	0.000158

Table 5.4: **(Full table)** success rates with random attacks using Uniform noises and Bernoulli noises on 100 randomly chosen test images.

model	noisetype	ϵ	0.20	0.25	0.30
MNIST 2-layer, ReLU	Bernoulli	succ. rate	0.130	0.260	0.510
	Uniform	succ. rate	0.040	0.050	0.090
MNIST 3-layer, ReLU	Bernoulli	succ. rate	0.110	0.210	0.450
	Uniform	succ. rate	0.020	0.040	0.100
MNIST 4-layer, ReLU	Bernoulli	succ. rate	0.140	0.260	0.550
	Uniform	succ. rate	0.030	0.070	0.090
MNIST CNN 2-layer, ReLU	Bernoulli	succ. rate	0.351	0.722	0.907
	Uniform	succ. rate	0.093	0.186	0.258
MNIST CNN 2-layer (robust), ReLU	Bernoulli	succ. rate	0.130	0.220	0.380
	Uniform	succ. rate	0.040	0.080	0.110
MNIST CNN 2-layer, atan	Bernoulli	succ. rate	0.842	0.895	0.979
	Uniform	succ. rate	0.484	0.611	0.726
MNIST CNN 2-layer, sigmoid	Bernoulli	succ. rate	0.469	0.677	0.833
	Uniform	succ. rate	0.073	0.167	0.281
MNIST CNN 2-layer, tanh	Bernoulli	succ. rate	0.806	0.939	0.990
	Uniform	succ. rate	0.276	0.500	0.673
MNIST CNN 2-layer (robust), tanh	Bernoulli	succ. rate	0.261	0.402	0.554
	Uniform	succ. rate	0.109	0.152	0.185
MNIST CNN 3-layer, ReLU	Bernoulli	succ. rate	0.250	0.540	0.850
	Uniform	succ. rate	0.060	0.090	0.160
MNIST CNN 3-layer (robust), ReLU	Bernoulli	succ. rate	0.091	0.172	0.495
	Uniform	succ. rate	0.040	0.061	0.081
MNIST CNN 3-layer, atan	Bernoulli	succ. rate	0.717	0.909	0.970
	Uniform	succ. rate	0.232	0.404	0.616
MNIST CNN 3-layer, sigmoid	Bernoulli	succ. rate	0.582	0.816	0.918
	Uniform	succ. rate	0.143	0.224	0.388
MNIST CNN 3-layer, tanh	Bernoulli	succ. rate	0.827	0.969	0.980
	Uniform	succ. rate	0.367	0.449	0.643
MNIST CNN 3-layer (robust), tanh	Bernoulli	succ. rate	0.149	0.223	0.298
	Uniform	succ. rate	0.085	0.117	0.128
MNIST CNN lenet, No Pool	Bernoulli	succ. rate	0.130	0.290	0.570
	Uniform	succ. rate	0.040	0.060	0.120
model		ϵ	0.02	0.025	0.03
CIFAR 5-layer, ReLU	Bernoulli	succ. rate	0.065	0.081	0.113
	Uniform	succ. rate	0.048	0.065	0.065
CIFAR 6-layer, ReLU	Bernoulli	succ. rate	0.172	0.203	0.219
	Uniform	succ. rate	0.062	0.109	0.141
CIFAR CNN 5-layer, ReLU	Bernoulli	succ. rate	0.299	0.373	0.463
	Uniform	succ. rate	0.164	0.209	0.254

Table 5.5: **Gaussian i.i.d noises**: compare PROVEN with worst-case certification CNN-Cert [8]

Certification Method	Worst-case	Our probabilistic approach: PROVEN					Certification
Guarantees γ_L	100%†	99.99%†	75%	50%	25%	5%	Bound increase†
MNIST 2-layer arctan	0.02105	0.02906	0.02944	0.02953	0.02962	0.02974	38.05%
MNIST 2-layer relu	0.03158	0.04314	0.04372	0.04385	0.04397	0.04416	36.61%
MNIST 2-layer relu (20)	0.02746	0.05319	0.05433	0.05458	0.05483	0.05519	93.7%
MNIST 2-layer sigmoid	0.02785	0.04118	0.04192	0.04202	0.04213	0.04233	47.86%
MNIST 2-layer tanh	0.02232	0.02884	0.02911	0.02918	0.02925	0.02936	29.21%
MNIST 3-layer relu	0.02397	0.02759	0.02777	0.02781	0.02785	0.02791	15.1%
MNIST 3-layer relu (20)	0.02236	0.03138	0.03180	0.03190	0.03199	0.03213	40.34%
MNIST 3-layer sigmoid	0.01856	0.02272	0.02292	0.02297	0.02301	0.02308	22.41%
MNIST 3-layer tanh	0.01121	0.01245	0.01251	0.01252	0.01253	0.01255	11.06%
MNIST 4-layer arctan	0.00726	0.00770	0.00772	0.00773	0.00773	0.00774	6.06%
MNIST 4-layer relu	0.00962	0.00999	0.01001	0.01001	0.01002	0.01002	3.85%
MNIST 4-layer tanh	0.00682	0.00712	0.00714	0.00714	0.00715	0.00715	4.4%
MNIST CNN 2-layer	0.04163	0.06143	0.06328	0.06371	0.06414	0.06477	47.56%
MNIST CNN 2-layer atan	0.03115	0.18108	0.21373	0.22139	0.22939	0.24141	481.32%
MNIST CNN 2-layer sigmoid	0.09606	0.22781	0.24334	0.24690	0.25012	0.25471	137.15%
MNIST CNN 2-layer tanh	0.03050	0.13764	0.15686	0.16142	0.16609	0.17282	351.28%
MNIST CNN 3-layer	0.04259	0.06259	0.06432	0.06471	0.06511	0.06569	46.96%
MNIST CNN 3-layer atan	0.02637	0.06108	0.06428	0.06504	0.06584	0.06696	131.63%
MNIST CNN 3-layer sigmoid	0.07845	0.15625	0.16324	0.16478	0.16640	0.16859	99.17%
MNIST CNN 3-layer tanh	0.02933	0.06215	0.06543	0.06614	0.06694	0.06806	111.9%
CIFAR 5-layer relu	0.00228	0.00234	0.00234	0.00234	0.00234	0.00234	2.63%
CIFAR 5-layer tanh	0.00081	0.00084	0.00084	0.00084	0.00084	0.00084	3.7%
CIFAR 7-layer relu	0.00189	0.00193	0.00193	0.00193	0.00193	0.00193	2.12%
CIFAR CNN 5-layer	0.00420	0.00497	0.00499	0.00499	0.00500	0.00501	18.33%
MNIST CNN 2-layer, adv	0.08848	0.11645	0.11987	0.12066	0.12144	0.12253	31.61%
MNIST CNN 2-layer tanh, adv	0.05329	0.12907	0.14388	0.14753	0.15133	0.15706	142.2%
MNIST CNN 3-layer adv	0.10377	0.12499	0.12709	0.12755	0.12801	0.12867	20.45%
MNIST CNN 3-layer tanh, adv	0.04064	0.17667	0.19838	0.20346	0.20856	0.21607	334.72%

Table 5.6: **Gaussian correlated noises**: compare PROVEN with worst-case certification CNN-Cert [8]

Certification Method	Worst-case	Our probabilistic approach: PROVEN					Certification
Guarantees γ_L	100%†	99.99%†	75%	50%	25%	5%	Bound increase†
MNIST 2-layer arctan	0.02105	0.02906	0.02944	0.02953	0.02962	0.02974	38.05%
MNIST 2-layer relu	0.03158	0.04315	0.04372	0.04385	0.04397	0.04416	36.64%
MNIST 2-layer relu (20)	0.02746	0.05319	0.05433	0.05458	0.05483	0.05519	93.7%
MNIST 2-layer sigmoid	0.02785	0.04118	0.04192	0.04202	0.04213	0.04233	47.86%
MNIST 2-layer tanh	0.02232	0.02885	0.02911	0.02918	0.02925	0.02936	29.26%
MNIST 3-layer relu	0.02397	0.02759	0.02777	0.02781	0.02785	0.02791	15.1%
MNIST 3-layer relu (20)	0.02236	0.03138	0.03180	0.03190	0.03199	0.03213	40.34%
MNIST 3-layer sigmoid	0.01856	0.02272	0.02292	0.02297	0.02301	0.02308	22.41%
MNIST 3-layer tanh	0.01121	0.01245	0.01251	0.01252	0.01253	0.01255	11.06%
MNIST 4-layer arctan	0.00726	0.00770	0.00772	0.00773	0.00773	0.00774	6.06%
MNIST 4-layer relu	0.00962	0.00999	0.01001	0.01001	0.01002	0.01002	3.85%
MNIST 4-layer tanh	0.00682	0.00712	0.00714	0.00714	0.00715	0.00715	4.4%
MNIST CNN 2-layer	0.04163	0.06144	0.06328	0.06371	0.06414	0.06475	47.59%
MNIST CNN 2-layer atan	0.03115	0.18122	0.21367	0.22139	0.22930	0.24158	481.77%
MNIST CNN 2-layer sigmoid	0.09606	0.22788	0.24333	0.24690	0.25016	0.25472	137.23%
MNIST CNN 2-layer tanh	0.03050	0.13742	0.15682	0.16142	0.16612	0.17281	350.56%
MNIST CNN 3-layer	0.04259	0.06258	0.06432	0.06471	0.06511	0.06570	46.94%
MNIST CNN 3-layer atan	0.02637	0.06111	0.06428	0.06504	0.06585	0.06697	131.74%
MNIST CNN 3-layer sigmoid	0.07845	0.15624	0.16325	0.16478	0.16641	0.16863	99.16%
MNIST CNN 3-layer tanh	0.02933	0.06217	0.06544	0.06614	0.06694	0.06806	111.97%
CIFAR 5-layer relu	0.00228	0.00234	0.00234	0.00234	0.00234	0.00234	2.63%
CIFAR 5-layer tanh	0.00081	0.00084	0.00084	0.00084	0.00084	0.00084	3.7%
CIFAR 7-layer relu	0.00189	0.00193	0.00193	0.00193	0.00193	0.00193	2.12%
CIFAR CNN 5-layer	0.00420	0.00497	0.00499	0.00499	0.00500	0.00501	18.33%
MNIST CNN 2-layer, adv	0.08848	0.11647	0.11988	0.12066	0.12143	0.12253	31.63%
MNIST CNN 2-layer tanh adv	0.05329	0.12907	0.14388	0.14753	0.15133	0.15711	142.2%
MNIST CNN 3-layer adv	0.10377	0.12501	0.12709	0.12755	0.12801	0.12867	20.47%
MNIST CNN 3-layer tanh adv	0.04064	0.17672	0.19833	0.20346	0.20853	0.21614	334.84%

5.4 Conclusions

In this Chapter, we proposed a novel probabilistic framework PROVEN to verify the robustness of neural networks and derived theoretical bounds on the local robustness certificate with statistical guarantees. PROVEN is a general tool that can build on top of existing state-of-the-art neural network robustness certification algorithms including Fast-Lin, CROWN and CNN-Cert and hence can be readily applied to fully-connected and convolutional neural networks with different activation functions. Experimental results on large neural networks demonstrated significant benefits of PROVEN over the standard worst-case analysis results.

Chapter 6

Extensions of Robustness Quantification Framework

In this Chapter, we summarized a few follow-up works under my supervision based on the general adaptive bounding framework we proposed in Chapter 4. Specifically, we show that the adaptive bounding framework can be further extended to general convolutional neural networks (CNNs) and recurrent neural networks (RNNs) in Section 6.1 and 6.2 respectively. In Section 6.3, we show that it is possible to build on-top of the proposed adaptive bounding framework to quantify robustness of neural networks on a family of *semantic* perturbations, including color changes, brightness, contrast, rotations, translations, etc.

6.1 Robustness Verification on General CNNs

This work is published in AAAI 2019 by Boopathy et al. [8] with the title "*Cnn-cert: An Efficient Framework for Certifying Robustness of Convolutional Neural Networks*".

6.1.1 Motivation

By 2019, existing methods lack generality in supporting different network architectures and activation functions. In addition, existing methods often deal with convolutional

layers by simply converting back to fully-connected layers, which may lose efficiency if not fully optimized with respect to the NNs, as demonstrated in our experiments. To bridge this gap, we propose CNN-Cert, a general and efficient verification framework for certifying robustness of a broad range of convolutional neural networks (CNNs). The generality of CNN-Cert enables robustness certification of various architectures, including convolutional layers, max-pooling layers batch normalization layers and residual blocks, and general activation functions. The efficiency of CNN-Cert is optimized by exploiting the convolution operation.

6.1.2 Overview

We show that the theory in Chapter 4 can be generalized to general CNNs and that the linear bounding framework can be modularized to commonly-used building blocks, including convolutional layers, pooling layers, residual blocks and batch-normalization layers, depicted in Figure 6-1. The algorithm is named accordingly as CNN-Cert. Thanks to the modularity, CNN-Cert is able verify more complicated neural networks (e.g. LeNet, ResNet) with arbitrary combinations of popular building blocks.

The key idea is to characterize the input-output relations between each module (e.g. relations between Φ^r and Φ^{r-1}, see Figure 6-1) and derive the linear upper bound and lower bound for each module by unwrapping the non-linearity. The corresponding linear coefficients $\mathbf{A}_U^r, \mathbf{A}_L^r$ and biases $\mathbf{B}_U^r, \mathbf{B}_L^r$ can be obtained such that we have

$$\Phi^r \leq \mathbf{A}_U^r * \Phi^{r-1} + \mathbf{B}_U^r,$$
$$\Phi^r \geq \mathbf{A}_L^r * \Phi^{r-1} + \mathbf{B}_L^r.$$

It can be shown that by applying the relationship between Φ^{r-1} and Φ^{r-2}, we can further get

$$\Phi^r \leq \mathbf{A}_U^{r-1} * \Phi^{r-2} + \mathbf{B}_U^{r-1},$$
$$\Phi^r \geq \mathbf{A}_L^{r-1} * \Phi^{r-2} + \mathbf{B}_L^{r-1}.$$

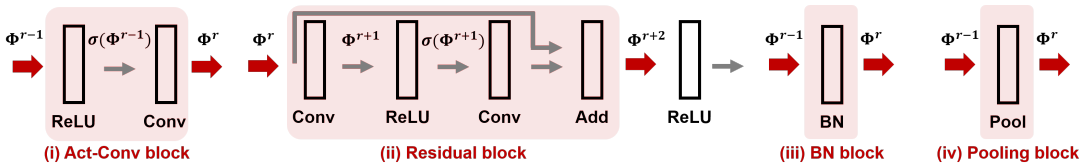


Figure 6-1: Cartoon graph of commonly-used building blocks considered in the CNN-Cert framework. The key step in deriving explicit network output bound is to consider the input/output relations of each building block, marked as red arrows. The activation layer can be general activations but here is denoted as ReLU.

Hence, with the linear coefficients $\mathbf{A}_U^k, \mathbf{A}_L^k$ and biases $\mathbf{B}_U^k, \mathbf{B}_L^k$ for each module, and viewing the neural network as cascading different modules in series, we can obtain the final $(\mathbf{A}_U^1, \mathbf{A}_L^1, \mathbf{B}_U^1, \mathbf{B}_L^1)$ that relates the output of neural networks to the input vector $\mathbf{x}$:

$$\Phi^r \leq \mathbf{A}_U^1 * \mathbf{x} + \mathbf{B}_U^1,$$
$$\Phi^r \geq \mathbf{A}_L^1 * \mathbf{x} + \mathbf{B}_L^1.$$

Our experiments showed that CNN-Cert provide more computationally efficient local robustness certificate on CNNs (up to 11× faster than CROWN).

6.1.3 Contributions

- CNN-Cert is *general* – it can certify robustness on general CNNs with various building blocks, including convolutional/pooling/batch-norm layers and residual blocks, as well as general activation functions such as ReLU, tanh, sigmoid and arctan. Other variants can easily be incorporated. Moreover, certification algorithms Fast-Lin [101] and CROWN [106] are special cases of CNN-Cert.

- CNN-Cert is *computationally efficient* – the cost is similar to forward-propagation as opposed to NP-completeness in formal verification methods, e.g. Reluplex [48]. Extensive experiments show that CNN-Cert achieves up to 17 times of speed-up compared to state-of-the-art certification algorithms Fast-Lin and up to 366 times of speed-up compared to dual-LP approaches while CNN-Cert obtains similar or even better verification bounds.

6.2 Robustness Verification on RNNs

This work is published in ICML 2019 by Ko et al. [51] with the title "*POPQORN: Quantifying robustness of recurrent neural networks*".

6.2.1 Motivations

Recurrent neural networks (RNNs) are essential in the applications such as natural language processing and speech recognition. Compared to feed-forward networks such as CNNs, RNNs have additional challenges due to sequential inputs and non-linear interaction between gates and states. In particular, there are a few challenges to be overcome:

- Some popular RNN formulations, including LSTM and GRU, involve hidden states that are indirectly determined by three to four gates. These nonlinear gates are tightly coupled together, which we refer to as *cross-nonlinearity*. This substantially complicates the verification of robustness,

- RNNs are widely adopted for applications with sequential inputs, *e.g.* sentences or time series. However, previous verification methods typically assume that the inputs were fed into the network at the bottom layer. Hence, they are not directly applicable here

- For applications with sequential inputs, imperceptible adversarial examples may correspond to texts with least number of changed words [35]. Thus it is critical to evaluate the hardness of manipulating one single word (one input frame) instead of all words.

6.2.2 Overview

We tackle the aforementioned challenges by proposing an effective robustness quantification framework called POPQORN (**P**ropagated-**o**ut**p**ut **Q**uantified R**o**bustness for **RN**Ns) for RNNs. We bound the non-linear activation function using linear functions.

Starting from the output layer, linear bounds are propagated back to the first layer recursively. Using a variant of linear bounding framework, we proposed a general algorithm to quantify robustness of general RNNs, including vanilla RNNs, LSTMs, and GRUs, and show that robustness quantification can also provide insights on the importance and sensitivity of a single frame in sequence data. In particular, we show that the output of an RNN can be bounded by two *linear* functions when the input sequence of the network is perturbed within an ℓ_p ball with a radius ϵ. By applying the *linear* bounds on the *non-linear* activation functions (e.g. sigmoid and tanh) and the *non-linear* operations (e.g. cell states in LSTM), we can *decouple* the non-linear activations and the *cross-nonlinearity* in the hidden states layer by layer and eventually bound the network output by two *linear* functions in terms of input[1].

For a vanilla RNN, the derivations are similar to the CNN-Cert and CROWN except that the recurrent neural network structure is now more complicated. However, for LSTM, there are additional challenges on cross non-linearity, i.e. there are operations that couple two non-linear functions together rather than just a single non-linear function with composition in vanilla RNNs:

$$\sigma(\mathbf{v}) \odot \mathbf{z} \quad \text{and} \quad \sigma(\mathbf{v}) \odot \tanh(\mathbf{z}),$$

where $\mathbf{v}$ and $\mathbf{z}$ are both perturbed, and hence the original linear bounding framework is not directly applicable. The key idea of this work is to use *planes* instead of *lines* to bound these cross terms:

$$\sigma(\mathbf{v}) \odot \mathbf{z} \leq \mathbf{A}^r_{U,1}\mathbf{v} + \mathbf{A}^r_{U,2}\mathbf{z} + \mathbf{B}^r_U,$$
$$\sigma(\mathbf{v}) \odot \mathbf{z} \geq \mathbf{A}^r_{L,1}\mathbf{v} + \mathbf{A}^r_{L,2}\mathbf{z} + \mathbf{B}^r_L.$$

The graphical illustration of the bounding planes is shown in Figure 6-2.

[1] Proposed bounding techniques are applicable to any non-linear activation that is bounded above and below for the given interval.

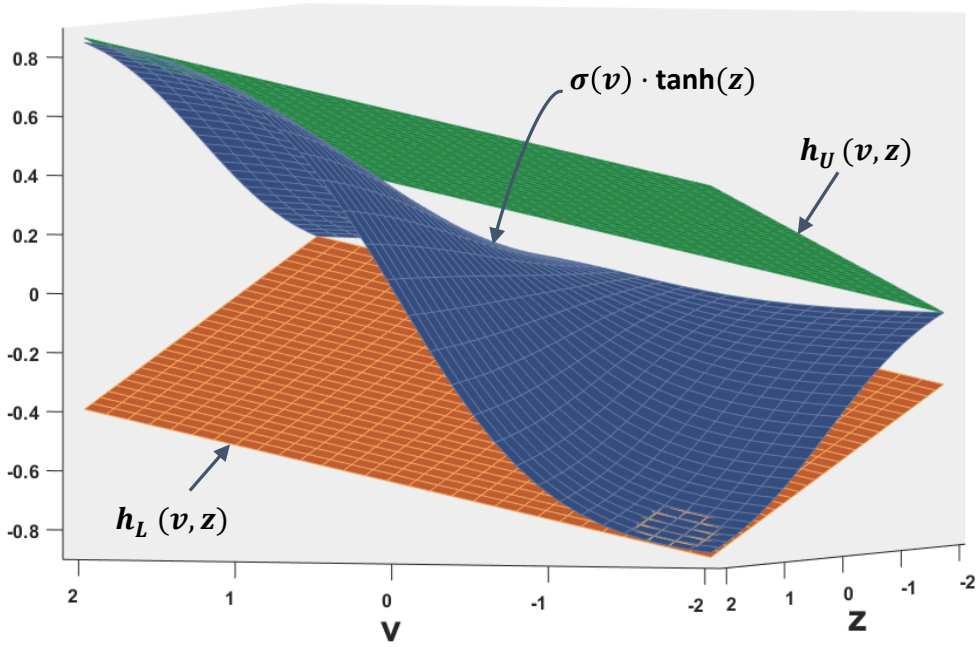


Figure 6-2: Illustration of the upper bounding plane and lower bounding plane of $\sigma(\mathbf{v}) \odot \mathbf{z}$.

6.2.3 Contributions

Compared to existing methods, POPQORN has three important advantages:

1. *Novel* - it is a general framework, which is, to the best of our knowledge, the **first** work to provide a quantified robustness evaluation for RNNs with robustness guarantees.

2. *Effective* - it can handle complicated RNN structures besides vanilla RNNs, including LSTMs and GRUs that contain challenging coupled nonlinearities.

3. *Versatile* - it can be widely applied in applications including but not limited to computer vision, natural language processing, and speech recognition.

Experiment on MNIST, MNIST sequence and Question classification tasks on UIUC's CogComp QC Dataset [65] have demonstrated that POPQORN can compute non-trivial certified local robustness radius, and provide insights on the importance and sensitivity of a single frame in sequence data.

6.3 Beyond Norm-based Distance Metrics

This work is accepted by CVPR 2020 by Mohapatra et al. [72] with the title "*Towards Verifying Robustness of Neural Networks Against a Family of Semantic Perturbations*".

6.3.1 Motivations

Beyond the ℓ_p-norm bounded threat model, recent works have shown the possibility of generating *semantic* adversarial examples based on semantic perturbation techniques such as color shifting, lighting adjustment and rotation [42, 66, 6, 47, 31, 28]. We refer the readers to Figure 6-3 for the illustration of some semantic perturbations for images. Notably, although semantically similar, these semantic adversarial attacks essentially consider different threat models than ℓ_p-norm bounded attacks in the RGB space. Therefore, semantic adversarial examples usually incur large ℓ_p-norm perturbations to the original data sample and thus exceed the verification capacity of ℓ_p-norm based verification methods. To bridge this gap and with an endeavor to render robustness verification methods more inclusive, we propose *Semantify-NN*, a model-agnostic and generic robustness verification against semantic perturbations.

6.3.2 Overview

In this work, we show that the ℓ_p-norm based verification techniques [52, 101, 26, 106, 90, 98, 83, 8] can be extended to handle other types of perturbations that are not norm-bounded in the input space. By representing the semantic transformation via neural network operations, we create a semantify-NN model such that the norm based verification tools can be applied to verify robustness against semantic perturbations. *Semantify-NN* is model-agnostic because it can apply to any given trained model by simply inserting our designed *semantic perturbation layers* (SP-layers). It is also generic since after adding SP-layers, one can apply any ℓ_p-norm based verification tools for certifying semantic perturbations. In other words, our proposed SP-layers work as a carefully designed converter that transforms semantic threat models to ℓ_p-norm threat models.

The proposed semantify-NN works for a variety of semantic attacks, including hue/saturation/lightness change in color space, brightness and contrast adjustment, rotation, translation and occlusion. We show that the proposed method can achieve 2-3 orders of magnitude larger (tighter) semantic robustness certificate than the baselines that directly uses the same ℓ_p-norm verification methods to handle semantic perturbations on rich combinations of three datasets (MNIST, CIFAR-10 and GTSRB) and five different network architectures (MLPs and CNNs). In particular, our method without further refinement can already achieve around 2-3 orders of magnitude larger (tighter) semantic robustness certificate than the baselines that directly uses the same ℓ_p-norm verification methods to handle semantic perturbations. With the proposed refinement technique, the semantic robustness certificate can be further improved by 100-300%.

6.3.3 Contributions

- We propose *Semantify-NN*, a model-agnostic and generic robustness verification toolkit for semantic perturbations. *Semantify-NN* can be viewed as a powerful extension module consisting of novel *semantic perturbation layers* (SP-layers) and is compatible to existing ℓ_p-norm based verification tools. The results show that *Semantify-NN* can support robustness verification against a wide range of semantic perturbations.

- We elucidate the design principles of our proposed SP-layers for a variety of semantic attacks, including hue/saturation/lightness change in color space, brightness and contrast adjustment, rotation, translation and occlusion. We also propose to use input space refinement and splitting methods to further improve the performance of robustness verification. In addition, we illustrate the need and importance of robustness verification for continuously parameterized perturbations.

- We propose an efficient refinement technique, *input splitting*, that can further tighten the semantic certificate delivered by *Semantify-NN*. Our extensive exper-

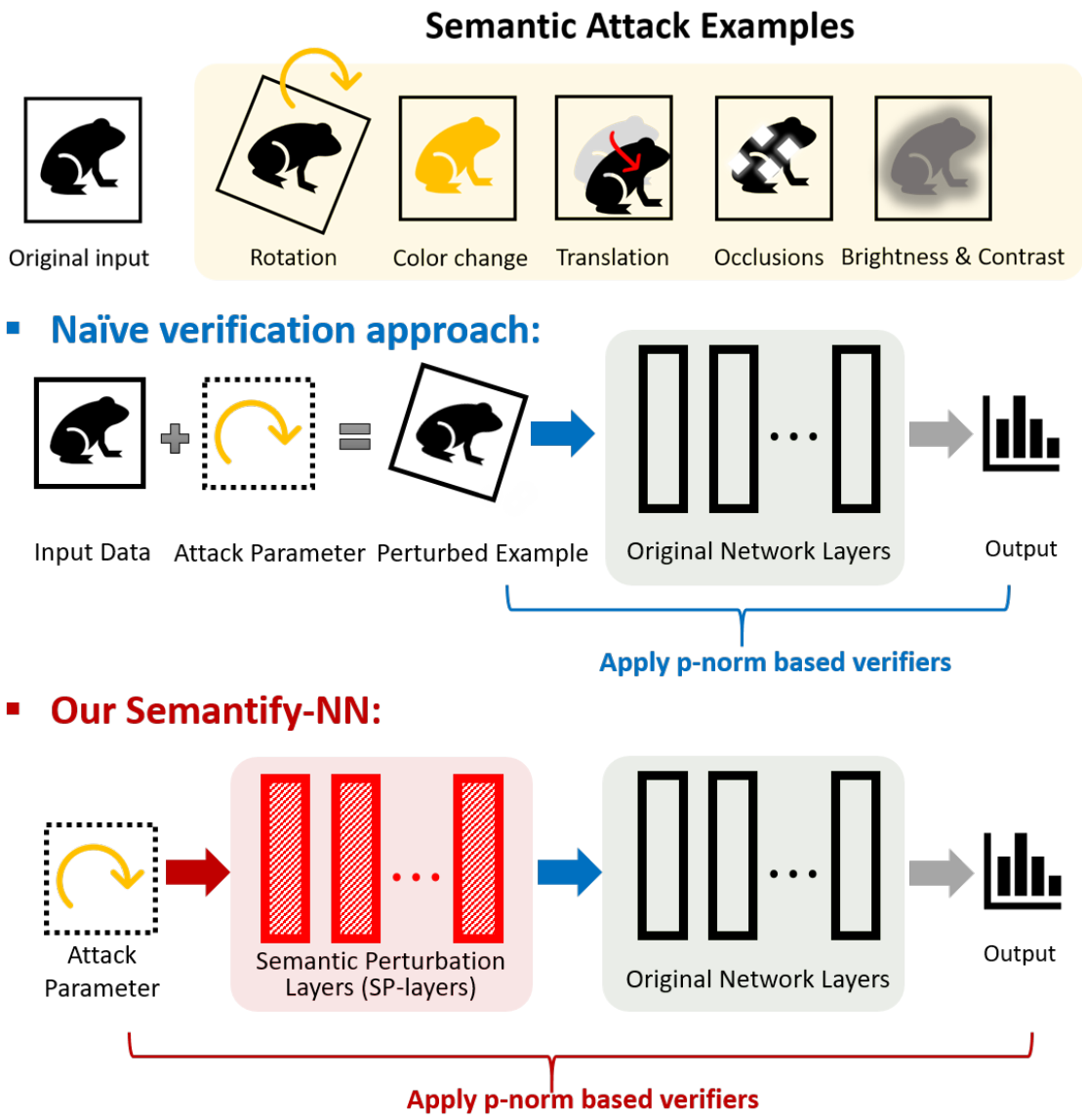


Figure 6-3: Schematic illustration of our proposed *Semantify-NN* robustness verification framework. Given a semantic attack threat model, *Semantify-NN* designs the corresponding semantic perturbation layers (SP-layers) and inserts them to the input layer of the original network for verification. With SP-layers, *Semantify-NN* can use any ℓ_p-norm based verification method for verifying semantic perturbations.

iments evaluated on the rich combinations of three datasets (MNIST, CIFAR-10 and GTSRB) and five different network architectures (MLPs and CNNs) corroborate the superior verification performance of *Semantify-NN* over naive ℓ_p-norm based verification methods. In particular, our method without further refinement can already achieve around 2-3 orders of magnitude larger (tighter) semantic robustness certificate than the baselines that directly uses the same ℓ_p-norm verification methods to handle semantic perturbations. With our *input splitting* technique, the semantic robustness certificate can be further improved by 100-300% .

Chapter 7

Conclusions and Future Work

7.1 Summary of Results

In this thesis, we have developed a set of algorithms and frameworks to efficiently quantify the robustness of neural networks against both adversarial and non-adversarial input perturbations. Our algorithms have demonstrated significant scalability on practical dataset ($10^2\times$ to $10^4\times$ larger) and state-of-the-art DNNs (up to $10^5\times$ more neurons) over formal verification based method [48]. In particular, for our proposed linear-bounding based robustness verification framework, we can achieve up to $10^6\times$ speed-up compared to formal verification methods based on SMT solvers, and up to $10^2\times$ compared to Linear Programming based solvers while having similar quality of robustness certificate.

Our main results of this thesis are summarized below.

Chapter 3 has proposed a novel robustness metric, CLEVER score, which is the first robustness metric that is attack-independent and can be applied to state-of-the-art neural network classifier and scales to large networks for ImageNet. CLEVER score is well supported by our theoretical analysis on formal robustness guarantees based on Lipschitz continuity and the use of extreme value theory, and our robustness analysis extends the results in [41] from continuously differentiable functions to a special class of non-differentiable functions – neural networks with ReLU activations. We corroborated the effectiveness of CLEVER by conducting experiments on state-of-the-art models for

ImageNet, including ResNet [39], Inception-v3 [93] and MobileNet [44]. We also use CLEVER to investigate defended networks against adversarial examples, including the use of defensive distillation [76] and bounded ReLU [105]. Experimental results show that our CLEVER score well aligns with the attack-specific robustness indicated by the ℓ_2 and ℓ_∞ distortions of adversarial examples.

Chapter 4 has developed a generic verification framework to verify neural network robustness with *provable guarantees*. Specifically, we have proposed a generic analysis framework CROWN for certifying NNs using linear or quadratic upper and lower bounds for *general* activation functions that are not necessarily piece-wise linear. CROWN is a generalized version of our previous robustness verification algorithm Fast-Lin [101] based on linear parallel bounding techniques in that CROWN allows flexible selections of upper/lower bounds for activation functions, enabling an *adaptive* scheme that helps to reduce approximation errors. Our experiments show that CROWN achieves up to 26% improvements in certified local robustness radius compared to Fast-Lin [101]. In addition, CROWN is efficient and can scale to large NNs with various activation functions. For a NN with over 10,000 neurons, we can give a certified local robustness radius in about 1 minute on 1 CPU core.

Chapter 5 has further developed a probabilistic framework PROVEN to verify neural network robustness with provable *probabilistic* guarantees. PROVEN can provide robustness certificates under ℓ_p norm-ball bounded threat models, when the input noise follows a given distributional characterization (Gaussian and Sub-Gaussian distributions with bounded supports). The established theoretical results are based on an ℓ_∞ constraint on the perturbation but can be easily extended to other norms such as ℓ_1 and ℓ_2. Experimental results on large neural networks trained on MNIST and CIFAR datasets show that PROVEN greatly improved the robustness certificate (i.e., the certified local robustness radius) compared with the corresponding worst-case analysis results, even when the statistical risk is small. For example, with a confidence level of 99.99%, which means the robustness metric is almost 100% guaranteed to be certified, the robustness certificate improvement provided by PROVEN over worst-case analysis can be as high as 250%. In addition to the noticeable improvement in the

verified robustness, PROVEN is a general tool that can be readily applied to neural networks with general activation functions, including but not limited to tanh, sigmoid and arctan, and general convolutional neural networks with various building blocks, as demonstrated in our experiments. Moreover, PROVEN is as computationally efficient as the worst-case analysis because its closed-form probabilistic certificate are by-products of worst-case certification frameworks such as Fast-Lin [101], CROWN [106], and CNN-Cert [8].

Chapter 6 summarized three follow-up works based on the general linear bounding framework developed in Chapter 4. Specifically, we show that the linear bounding framework can be further extended to general convolutional neural networks (CNNs) and recurrent neural networks (RNNs), and that it is possible to build on-top of the proposed linear bounding framework to quantify robustness of neural networks on a family of *semantic* perturbations, including color changes, brightness, contrasts, rotations, translations, etc simply with an additional transformation layer.

7.2 Future works

There exist lots of topics worth further study and investigation. We summarize a few of them in below.

Scalable Robustness Verification on ImageNet with Provable Guarantees. In this thesis, we have developed several pioneering techniques CROWN, PROVEN that can scale to standard mnist and cifar dataset with non-trivial robustness guarantees, which is much better than formal verification based methods [48]. However, it is still limited on the standard MNIST and CIFAR networks and can hardly scale to ImageNet, similarly for other methods in this field. Fortunately, there are some recent work trying to resolve this bottleneck, and the key idea is to *change the original prediction rule* by aggregating the prediction results of multiple perturbed samples (e.g. perturbed by Gaussian noises). By modifying the prediction rule, this technique is called *randomized* smoothing, and there are multiple pioneering works that are able to provide robustness guarantees with high probability on ℓ_0 norm [61] and

ℓ_2 norm [21, 60, 64]. There are some interesting questions remain to be answered: for example, is there a systematic way to design the prediction rules to boost up the robustness certificate? Also, is it possible to take advantage of the layer-wise bound information from linear bounding framework with the randomized smoothing technique to obtain a tighter certificate (i.e. the so-called 'grey-box' setting)?

Beyond input perturbation. In this thesis, we only focus on perturbing the input of neural networks. However, there exists other type of perturbations in practice, e.g. perturbation on the neural network parameters (so-called weight perturbations) are causing real threat and hence analyzing weight perturbations of DNNs are of realistic significance. A concrete example is demonstrated in [68, 108], where a new threat model of weight perturbations was proposed and the authors showed that the so-called *fault sneaking/injection attack* can enforce DNN to misclassify some natural input images into target labels by slightly modifying weights at a single layer, while maintaining the classification of unspecified input images intact. This implies that the outputs of DNNs are also sensitive to weight perturbations. Moreover, there also exist weight perturbations in the non-adversarial setting. For example, *weight quantization* [109, 62, 84], a major DNN model compression technique commonly utilized by industry for DNN acceleration/implementation, induces weight perturbations by replacing full floating-point precision weights with fixed-point lower precision weights. It is even well supported in GPUs and mobile devices, e.g., PyTorch [78] in NVIDIA GPUs and TensorFlow Lite [1] for mobile devices. However, such direct mapping from full precision weights of DNNs into quantized weights could result in significant generalization error [89].

Beyond classification tasks. The framework proposed in this paper can be applied in the supervised learning tasks, including classifications and regressions. However, there are other type of machine learning tasks, such as deep reinforcement learning (RL) that has a totally different problem setting than supervised learning tasks. In particular, over the last five years, deep RL has achieved great success in many previously difficult reinforcement learning tasks thanks to the power of deep neural networks. In addition, neural network policies have been widely deployed

in physical system to replace traditional model predictive control policy, e.g. in guided policy search, a neural network policy is used to directly control a physical system in a feedback loop. Yet, recent studies show that deep RL agents are also unavoidably susceptible to adversarial perturbations, similar to deep neural networks in classification tasks. Therefore, it is of great importance and urgent needs to develop efficient and effective robustness verification tools for the field of deep reinforcement learning. One idea is to leverage the neural network robustness certification algorithms that I developed earlier to derive robustness certificate for neural network control systems.

Connections to Fairness. As machine learning models are deployed widely to support decision making in real life, it is essential to ensure that the machine learning models do not unethically discriminate against protected groups. The field of fair AI is closely related to robust AI in the following sense: the goal of robust AI is to ensure that the output of the neural network model is invariant with respect to (semantic) perturbation, while the goal of fair AI is to ensure that the neural network output is invariant against factors such as race, gender, or age when other factors are identical. This connection suggests that robust AI problems and fair AI problems can be treated using the same mathematical framework, and the verification and robust training technology I developed could be useful in fair AI as well. One of my future research interest is to formalize a holistic mathematical framework to unify robust AI and fair AI problems to enhance mutual trust between human and next generation AI systems.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, and et. al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2016. Software available from tensorflow.org.

[2] Giovanni Apruzzese, Michele Colajanni, Luca Ferretti, Alessandro Guido, and Mirco Marchetti. On the effectiveness of machine and deep learning for cyber security. In *2018 10th International Conference on Cyber Conflict (CyCon)*, pages 371–390. IEEE, 2018.

[3] Anish Athalye and Ilya Sutskever. Synthesizing robust adversarial examples. *arXiv preprint arXiv:1707.07397*, 2017.

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

[5] Osbert Bastani, Yani Ioannou, Leonidas Lampropoulos, Dimitrios Vytiniotis, Aditya Nori, and Antonio Criminisi. Measuring neural net robustness with constraints. In *Advances in Neural Information Processing Systems*, pages 2613–2621, 2016.

[6] Anand Bhattad, Min Jin Chong, Kaizhao Liang, Bo Li, and David A Forsyth. Big but imperceptible adversarial perturbations via semantic manipulation. *arXiv preprint arXiv:1904.06347*, 2019.

[7] Adel Bibi, Modar Alfadly, and Bernard Ghanem. Analytic expressions for probabilistic moments of pl-dnn with gaussian input. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[8] Akhilan Boopathy, Tsui-Wei Weng, Pin-Yu Chen, Sijia Liu, and Luca Daniel. Cnn-cert: An efficient framework for certifying robustness of convolutional neural networks. In *AAAI*, Jan 2019.

[9] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[10] Wieland Brendel, Jonas Rauber, and Matthias Bethge. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. *ICLR*, 2018.

[11] Xiaoyu Cao and Neil Zhenqiang Gong. Mitigating evasion attacks to deep neural networks via region-based classification. In *ACM Annual Computer Security Applications Conference*, pages 278–287, 2017.

[12] Nicholas Carlini, Guy Katz, Clark Barrett, and David L Dill. Ground-truth adversarial examples. *arXiv preprint arXiv:1709.10207*, 2017.

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

[14] Hongge Chen, Huan Zhang, Pin-Yu Chen, Jinfeng Yi, and Cho-Jui Hsieh. Show-and-fool: Crafting adversarial examples for neural image captioning. *arXiv preprint arXiv:1712.02051*, 2017.

[15] Pin-Yu Chen, Yash Sharma, Huan Zhang, Jinfeng Yi, and Cho-Jui Hsieh. Ead: Elastic-net attacks to deep neural networks via adversarial examples. *arXiv preprint arXiv:1709.04114*, 2017.

[16] Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. ZOO: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *ACM Workshop on Artificial Intelligence and Security*, pages 15–26, 2017.

[17] Chih-Hong Cheng, Georg Nührenberg, and Harald Ruess. Maximum resilience of artificial neural networks. In *International Symposium on Automated Technology for Verification and Analysis*, pages 251–268. Springer, 2017.

[18] Travers Ching, Daniel S Himmelstein, Brett K Beaulieu-Jones, Alexandr A Kalinin, Brian T Do, Gregory P Way, Enrico Ferrero, Paul-Michael Agapow, Michael Zietz, Michael M Hoffman, et al. Opportunities and obstacles for deep learning in biology and medicine. *Journal of The Royal Society Interface*, 15(141):20170387, 2018.

[19] Yuan Shih Chow and Henry Teicher. *Probability Theory: Independence, Interchangeability, Martingales*. Springer, third edition, 2003.

[20] Moustapha M Cisse, Yossi Adi, Natalia Neverova, and Joseph Keshet. Houdini: Fooling deep structured visual and speech recognition models with adversarial examples. In *NIPS*, 2017.

[21] Jeremy Cohen, Elan Rosenfeld, and Zico Kolter. Certified adversarial robustness via randomized smoothing. In *ICML*, 2019.

[22] Laurens De Haan and Ana Ferreira. *Extreme value theory: an introduction*. Springer Science & Business Media, 2007.

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

[24] Devdatt P. Dubhashi and Desh Ranjan. Balls and bins: A study in negative dependence. *Random Structures and Algorithms*, 13(2):99–124, 1998.

[25] Krishnamurthy Dvijotham, Marta Garnelo, Alhussein Fawzi, and Pushmeet Kohli. Verification of deep probabilistic models. In *arXiv preprint arXiv:1812.02795*, 2018.

[26] Krishnamurthy Dvijotham, Robert Stanforth, Sven Gowal, Timothy Mann, and Pushmeet Kohli. A dual approach to scalable verification of deep networks. *UAI*, 2018.

[27] Ruediger Ehlers. Formal verification of piece-wise linear feed-forward neural networks. *arXiv preprint arXiv:1705.01320*, 2017.

[28] Logan Engstrom, Brandon Tran, Dimitris Tsipras, Ludwig Schmidt, and Aleksander Madry. A rotation and a translation suffice: Fooling cnns with simple transformations. *arXiv preprint arXiv:1712.02779*, 2017.

[29] Ivan Evtimov, Kevin Eykholt, Earlence Fernandes, Tadayoshi Kohno, Bo Li, Atul Prakash, Amir Rahmati, and Dawn Song. Robust physical-world attacks on machine learning models. *arXiv preprint arXiv:1707.08945*, 2017.

[30] Alhussein Fawzi, Hamza Fawzi, and Omar Fawzi. Adversarial vulnerability for any classifier. *arXiv preprint arXiv:1802.08686*, 2018.

[31] Alhussein Fawzi and Pascal Frossard. Manitest: Are classifiers really invariant? *BMVC*, 2015.

[32] Alhussein Fawzi, Seyed-Mohsen Moosavi-Dezfooli, and Pascal Frossard. Robustness of classifiers: from adversarial to random noise. In *Advances in Neural Information Processing Systems*, pages 1632–1640, 2016.

[33] Matteo Fischetti and Jason Jo. Deep neural networks as 0-1 mixed integer linear programs: A feasibility study. *arXiv preprint arXiv:1712.06174*, 2017.

[34] Jean-Yves Franceschi, Alhussein Fawzi, and Omar Fawzi. Robustness of classifiers to uniform ℓ_p and gaussian noise. *arXiv preprint arXiv:1802.07971*, 2018.

[35] J. Gao, J. Lanchantin, M. L. Soffa, and Y. Qi. Black-box generation of adversarial text sequences to evade deep learning classifiers. In *2018 IEEE Security and Privacy Workshops*, pages 50–56, 2018.

[36] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *IEEE Symposium on Security and Privacy (SP)*, volume 00, pages 948–963, 2018.

[37] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *ICLR'15; arXiv preprint arXiv:1412.6572*, 2015.

[38] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, et al. Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*, 2014.

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

[40] JB Heaton, Nicholas G Polson, and Jan Hendrik Witte. Deep learning in finance. *arXiv preprint arXiv:1602.06561*, 2016.

[41] Matthias Hein and Maksym Andriushchenko. Formal guarantees on the robustness of a classifier against adversarial manipulation. *arXiv preprint arXiv:1705.08475*, 2017.

[42] Hossein Hosseini and Radha Poovendran. Semantic adversarial examples. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 1614–1619, 2018.

[43] Hossein Hosseini, Baicen Xiao, and Radha Poovendran. Google's cloud vision api is not robust to noise. In *Machine Learning and Applications (ICMLA), 2017 16th IEEE International Conference on*, pages 101–105. IEEE, 2017.

[44] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[45] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks. In *International Conference on Computer Aided Verification*, pages 3–29. Springer, 2017.

[46] Robin Jia and Percy Liang. Adversarial examples for evaluating reading comprehension systems. In *EMNLP*, 2017.

[47] Ameya Joshi, Amitangshu Mukherjee, Soumik Sarkar, and Chinmay Hegde. Semantic adversarial attacks: Parametric transformations that fool deep classifiers. *arXiv preprint arXiv:1904.08489*, 2019.

[48] Guy Katz, Clark Barrett, David Dill, Kyle Julian, and Mykel Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. *arXiv preprint arXiv:1702.01135*, 2017.

[49] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Towards proving the adversarial robustness of deep neural networks. *arXiv preprint arXiv:1709.02802*, 2017.

[50] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[51] Ching-Yun Ko, Zhaoyang Lyu, Tsui-Wei Weng, Luca Daniel, Ngai Wong, and Dahua Lin. Popqorn: Quantifying robustness of recurrent neural networks. *arXiv preprint arXiv:1905.07387*, 2019.

[52] J Zico Kolter and Eric Wong. Provable defenses against adversarial examples via the convex outer adversarial polytope. *ICML*, 2018.

[53] Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009.

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

[55] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016.

[56] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *ICLR'17; arXiv preprint arXiv:1611.01236*, 2016.

[57] Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. Recurrent convolutional neural networks for text classification. In *Twenty-ninth AAAI conference on artificial intelligence*, 2015.

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

[59] Mathias Lecuyer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, and Suman Jana. Certified robustness to adversarial examples with differential privacy. *arXiv preprint arXiv:1802.03471*, 2018.

[60] Mathias Lecuyer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, and Suman Jana. Certified robustness to adversarial examples with differential privacy. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 656–672. IEEE, 2019.

[61] Guang-He Lee, Yang Yuan, Shiyu Chang, and Tommi Jaakkola. Tight certificates of adversarial robustness for randomly smoothed classifiers. In *Advances in Neural Information Processing Systems*, pages 4911–4922, 2019.

[62] Cong Leng, Zesheng Dou, Hao Li, et al. Extremely low bit neural network: Squeeze the last bit out with admm. In *AAAI*, 2018.

[63] Bai Li, Changyou Chen, Wenlin Wang, and Lawrence Carin. Second-order adversarial attack and certifiable robustness. *arXiv preprint arXiv:1809.03113*, 2018.

[64] Bai Li, Changyou Chen, Wenlin Wang, and Lawrence Carin. Certified adversarial robustness with additive noise. In *NeurIPS*. 2019.

[65] Xin Li and Dan Roth. Learning question classifiers. In *COLING*, pages 1–7, 2002.

[66] Hsueh-Ti Derek Liu, Michael Tao, Chun-Liang Li, Derek Nowrouzezahrai, and Alec Jacobson. Beyond pixel norm-balls: Parametric adversaries using an analytically differentiable renderer. *International Conference on Learning Representations*, 2019.

[67] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8759–8768, 2018.

[68] Y. Liu, L. Wei, B. Luo, and Q. Xu. Fault injection attack on deep neural network. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 131–138, Nov 2017.

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

[70] Alessio Lomuscio and Lalit Maganti. An approach to reachability analysis for feed-forward relu neural networks. *arXiv preprint arXiv:1706.07351*, 2017.

[71] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[72] Jeet Mohapatra, Pin-Yu Chen, Sijia Liu, Luca Daniel, et al. Towards verifying robustness of neural networks against semantic perturbations. *arXiv preprint arXiv:1912.09533*, 2019.

[73] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2574–2582, 2016.

[74] Alessandro Panconesi and Aravind Srinivasan. Randomized distributed edge coloring via an extension of the chernoff-hoeffding bounds. *SIAM Journal on Computing*, 26(2):350–368, 1997.

[75] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *ACM Asia Conference on Computer and Communications Security*, pages 506–519, 2017.

[76] 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 (SP)*, pages 582–597, 2016.

[77] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics, 2002.

[78] Adam Paszke, Sam Gross, Soumith Chintala, and Gregory Chanan. Pytorch, 2017.

[79] Remigijus Paulavičius and Julius Žilinskas. Analysis of different norms and corresponding lipschitz constants for global optimization. *Technological and Economic Development of Economy*, 12(4):301–306, 2006.

[80] Jonathan Peck, Joris Roels, Bart Goossens, and Yvan Saeys. Lower bounds on the robustness to adversarial perturbations. In *NIPS*, 2017.

[81] Harry A Pierson and Michael S Gashler. Deep learning in robotics: a review of recent research. *Advanced Robotics*, 31(16):821–835, 2017.

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

[83] Aditi Raghunathan, Jacob Steinhardt, and Percy S Liang. Semidefinite relaxations for certifying robustness to adversarial examples. In *Advances in Neural Information Processing Systems*, pages 10877–10887, 2018.

[84] Ao Ren, Tianyun Zhang, Shaokai Ye, et al. Admm-nn: An algorithm-hardware co-design framework of dnns using alternating direction methods of multipliers. In *ASPLOS*, pages 925–938. ACM, 2019.

[85] Sidney I. Resnick. *A Probability Path*. Birkhäuser, 2014.

[86] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[87] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in Neural Information Processing Systems*, pages 2234–2242, 2016.

[88] Moshe Shaked and George Shanthikumar. *Stochastic Orders*. Springer, 2007.

[89] Tao Sheng, Chen Feng, Shaojie Zhuo, et al. A quantization-friendly separable convolution for mobilenets. In *2018 1st Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications*, pages 14–18. IEEE, 2018.

[90] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. Fast and effective robustness certification. In *NeurIPS*, 2018.

[91] Aman Sinha, Hongseok Namkoong, and John Duchi. Certifiable distributional robustness with principled adversarial training. *ICLR*, 2018.

[92] Christian Szegedy, Alexander Toshev, and Dumitru Erhan. Deep neural networks for object detection. In *Advances in neural information processing systems*, pages 2553–2561, 2013.

[93] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.

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

[95] Tijmen Tieleman and Geoffery Hinton. Rmsprop gradient optimization. *URL http://www. cs. toronto. edu/tijmen/csc321/slides/lecture_ slides_ lec6. pdf*, 2014.

[96] Beilun Wang, Ji Gao, and Yanjun Qi. A theoretical framework for robustness of (deep) classifiers under adversarial noise. *arXiv preprint arXiv:1612.00334*, 2016.

[97] Qinglong Wang, Wenbo Guo, Kaixuan Zhang, Alexander G Ororbia II, Xinyu Xing, Xue Liu, and C Lee Giles. Adversary resistant deep neural networks with an application to malware detection. In *SIGKDD*. ACM, 2017.

[98] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Efficient formal safety analysis of neural networks. In *NeurIPS*, 2018.

[99] Rachel Ward, Xiaoxia Wu, and Leon Bottou. Adagrad stepsizes: Sharp convergence over nonconvex landscapes, from any initialization. *arXiv preprint arXiv:1806.01811*, 2018.

[100] Stefan Webb, Tom Rainforth, Yee Whye Teh, and M Pawan Kumar. A statistical approach to assessing neural network robustness. *ICLR, arXiv preprint arXiv:1811.07209*, 2019.

[101] 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. *ICML*, 2018.

[102] Tsui-Wei Weng, Huan Zhang, Pin-Yu Chen, Jinfeng Yi, Dong Su, Yupeng Gao, Cho-Jui Hsieh, and Luca Daniel. Evaluating the robustness of neural networks: An extreme value theory approach. *ICLR*, 2018.

[103] GR Wood and BP Zhang. Estimation of the lipschitz constant of a function. *Journal of Global Optimization*, 8(1):91–103, 1996.

[104] Cihang Xie, Jianyu Wang, Zhishuai Zhang, Yuyin Zhou, Lingxi Xie, and Alan Yuille. Adversarial examples for semantic segmentation and object detection. In *ICCV*, 2017.

[105] Valentina Zantedeschi, Maria-Irina Nicolae, and Ambrish Rawat. Efficient defenses against adversarial attacks. *arXiv preprint arXiv:1707.06728*, 2017.

[106] Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. In *Advances in Neural Information Processing Systems (NIPS)*, 2018.

[107] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2881–2890, 2017.

[108] Pu Zhao, Siyue Wang, Cheng Gongye, et al. Fault sneaking attack: a stealthy framework for misleading deep neural networks. In *Proceedings of the 56th Annual Design Automation Conference*, 2019.

[109] Aojun Zhou, Anbang Yao, Yiwen Guo, et al. Incremental network quantization: Towards lossless cnns with low-precision weights. *2017 ICLR*, 2017.