A Tale of Two Approximations: Tightening Over-Approximation for DNN Robustness Verification via Under-Approximation

Anonymous Author(s)

ABSTRACT

The robustness of deep neural networks (DNNs) is crucial to the hosting system's reliability and security. Formal verification has been proven to be effective in providing provable robustness guarantees. To improve its scalability, over-approximating the non-linear activation functions in DNNs by linear constraints has been widely adopted, which transforms the verification problem into an efficiently solvable linear programming problem. Many efforts have been dedicated to defining the so-called tightest approximations to reduce overestimation imposed by over-approximation.

In this paper, we study existing approaches and identify a dominant factor in defining tight approximation, namely the *approximation domain* of the activation function. We find out that tight approximations defined on approximation domains may not be as tight as the ones on their actual domains, yet existing approaches all rely on only approximation domains. Based on this observation, we propose a novel dual-approximation approach to tighten over-approximations, leveraging an activation function's underestimated domain to define tight approximation bounds. We implement our approach with two complementary algorithms based respectively on Monte Carlo simulation and gradient descent into a tool called DualApp. We assess it on a comprehensive benchmark of DNNs with different architectures. Our experimental results show that DualApp significantly outperforms the state-of-the-art approaches with 100% – 1000% improvement on the verified robustness ratio and 10.64% on average (up to 66.53%) on the certified lower bound.

Data Availability. Our tool, proofs, and complete experimental data are included in the supplementary material.

1 INTRODUCTION

Deep Neural networks (DNNs) are the most crucial components in AI-empowered software systems. They must be guaranteed reliable and secure when the hosting system is safety-critical. Robustness is central to their safety and reliability, ensuring that neural networks can function correctly even under environmental perturbations and adversarial attacks [9, 40, 48]. Studying the robustness of DNNs from both training and engineering perspectives attracts researchers from both AI and SE communities [9, 16, 22, 25, 30, 40]. More recently, the emerging formal verification efforts on the robustness of neural networks aim at providing certifiable robustness guarantees for the neural networks [15, 24, 46]. Certified robustness of neural networks is necessary for guaranteeing that the hosting software system is both safe and secure. It is particularly crucial to those safety-critical applications such as autonomous drivings [1, 3], medical diagnoses [41], and face recognition [39].

Formally verifying the robustness of neural networks is computationally complex and expensive due to the high non-linearity and non-convexity of neural networks. The problem has been proved NP-complete even for the simplest fully-connected networks with the piece-wise linear activation function ReLU [17]. It is significantly more difficult for those networks that contain differentiable S-curve activation functions such as Sigmoid, Tanh, and Arctan [53]. To improve scalability, a practical solution is to over-approximate the nonlinear activation functions using linear upper and lower bounds. The verification problem is then transformed into an efficiently solvable linear programming problem. The linear over-approximation is a prerequisite for other advanced verification approaches based on abstraction [6, 34, 37], interval bound propagation (IBP) [13], and convex optimization [36, 47].

As over-approximations inevitably introduce overestimation, the corresponding verification approaches sacrifice completeness and may fail to prove or disprove the robustness of a neural network [24]. Consequently, we cannot conclude that a neural network is not robust when we fail to prove it is robust by over-approximation. An ideal approximation must be as tight as possible to resolve such uncertainties. Intuitively, an approximation is tighter if it introduces less overestimation to the robustness verification result.

Considerable efforts have been devoted to finding tight over-approximations for precise verification results [21, 23, 42, 49, 53]. The definition of tightness can be classified into two categories: neuron-wise and network-wise. An approximation method based on network-wise tightness is dedicated to defining a linear approximation so that the output for each neuron in the neural network is tight. An approximation method based on neuron-wise tightness only guarantees that the approximation is tight on the current neuron, while it does not consider the tightness of networks widely. Lyu *et al.* [26] and Zhang *et al.* [55] claim that computing the tightest approximation is essentially a network-wise non-convex optimization problem, and therefore almost impractical to solve directly due to high computational complexity. Hence, approximating each individual activation function separately is still an effective and practical solution. Experimental results have shown that existing tightness characterizations of neuron-wise over-approximations do not always imply precise verification results [36, 55]. It is highly desirable to explore missing factors in defining tighter neuron-wise approximations.

In this paper, we report a new, crucial factor for defining tight over-approximation, namely *approximation domain* of an activation function, which is missing by all the existing approximation approaches. Through both theoretical and experimental analyses, we identify that existing approaches all rely on only the approximation domain of an activation function to define linear lower and upper bounds, yet the bound that is tight on the approximation domain may not be tight on the activation function's *actual domain*. Unfortunately, computing the actual domain of an activation function on each neuron of a DNN is as difficult as the verification problem and thus impractical.

Towards estimating the actual domain, we propose a novel dual-approximation approach which, unlike existing approaches, leverages the underestimated domain of an activation function to define a

tight linear approximation. We first devise two under-approximation algorithms to compute the underestimated domain based on Monte Carlo simulation and gradient descent, respectively. In the Monte Carlo algorithm, we select a number of samples from the perturbed input region and feed them into a DNN, recording the maximum and minimum of each neuron as the underestimated domain. For the gradient-based algorithm, we feed the image into a DNN to obtain the gradient of each neuron relative to the input. Based on this, we fine-tune the input value and feed them into the DNN again to get the domain. We then use both underestimated and approximation domains to define tight linear bounds for the activation function. Specifically, we define a linear over-approximation bound on the underestimated domain and check if it is valid on the approximation domain. In a valid case, we approximate the activation function using the bound; otherwise, we define a bound on the original approximation domain. The underestimated domain is an inner approximation of the actual domain, which guarantees tightness, whereas the approximation domain guarantees soundness. Through an extensive analysis on a wide range of benchmarks and datasets, we demonstrate that our dual-approximation approach can produce tighter linear approximation than the state-of-the-art approaches that claim to provide the tightest approximation. In particular, our approach achieves 100% − 1000% improvement on the verified robustness ratio and 10.64% on average (up to 66.53%) on the certified lower bound.

In summary, we make three main contributions:

(1) We identify a crucial factor, called *approximation domain*, in defining tight over-approximations for the DNN robustness verification by a thorough study of the state-of-the-art over-approximation methods.
(2) We propose two under-approximation algorithms for computing underestimated domains, together with a dual-approximation approach to defining tight over-approximation for the DNN robustness verification.
(3) We implement our approach into a tool called DualApp and demonstrate its outperformance over six state-of-the-art tools on a wide range of benchmarks. We also experimentally explore the optimal parameter settings for computing more precise underestimated approximation domains.

2 PRELIMINARIES

2.1 Deep Neural Networks

A deep neural network(DNN) is a network of nodes called neurons connected end to end as shown in Figure 1, which implements a mathematical function $F : \mathbb{R}^n \to \mathbb{R}^m$, e.g., $n = 3$ and $m = 2$ for the 2-hidden-layer DNN in Figure 1. Neurons except input ones are also functions $f : \mathbb{R}^k \to \mathbb{R}$ in the form of $f(x) = \sigma(Wx + b)$, where k is the dimension of input vector x, $\sigma(\cdot)$ is called an *activation function*, W a matrix of weights and b a bias. During calculation, a vector of n numbers is fed into the neural network from the *input layer* and propagated layer by layer through the internal *hidden layers* after being multiplied by the weights on the edges, summed at the successor neurons with the bias and then computed by the neurons using the activation function. The neurons on the *output layer* compute the output values, which are regarded as probabilities of classifying an input vector to every label. The input vector can be

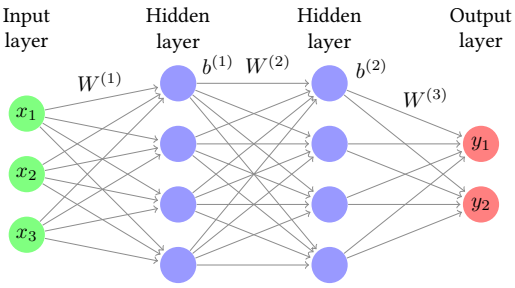


Figure 1: A 4-layer feedforward DNN with two hidden layers.

an image, a sentence, a voice, or a system state, depending on the application domains of the deep neural network.

Given an l-layer neural network, let $W^{(i)}$ be the matrix of weights between the i-th and $(i+1)$-th layers, and $b^{(i)}$ the biases on the corresponding neurons, where $i = 1, \ldots, l-1$. The function $F : \mathbb{R}^n \to \mathbb{R}^m$ implemented by the neural network can be defined by:

$$F(x) = W^{(l-1)}\sigma(z^{(l-1)}(x)), \qquad \text{(Network Function)}$$

$$\text{where } z^{(i)}(x) = W^{(i)}\sigma(z^{(i-1)}(x)) + b^{(i)} \qquad \text{(Layer Function)}$$

$$\text{and } z^{(0)}(x) = x \qquad \text{(Initialization)}$$

for $i = 1, \ldots, l-1$. For the sake of simplicity, we use $\hat{z}^{(i)}(x)$ to denote $\sigma(z^{(i)}(x))$ and $\Phi(x) = \arg\max_{\ell \in L} F(x)$ to denote the label ℓ such that the probability $F_\ell(x)$ of classifying x to ℓ is larger than those to other labels, where L represents the set of all labels. The activation function σ usually can be a Rectified Linear Unit (ReLU), $\sigma(x) = max(x, 0)$), a Sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$, a Tanh function $\sigma(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$, or an Arctan function $\sigma(x) = tan^{-1}(x)$. As ReLU neural networks have been comprehensively studied [24], we focus on the networks with only *S-curved* activation functions, i.e., Sigmoid, Tanh, and Arctan.

Given a training dataset, the task of training a DNN is to fine-tune the weights and biases so that the trained DNN achieves desired precision on test sets. Although a DNN is a precise mathematical function, its correctness is very challenging to guarantee due to the lack of formal specifications and the inexplicability of itself. Unlike programmer-composed programs, the machine-trained models are almost impossible to assign semantics to the internal computations.

2.2 Neural Network Robustness Verification

Despite the challenge in verifying the correctness of DNNs, formal verification is still useful to verify their safety-critical properties. One of the most important properties is *robustness*, stating that the prediction of a neural network is still unchanged even if the input is manipulated under a reasonable range:

Definition 1 (Neural Network Robustness). *A neural network $F : \mathbb{R}^n \to \mathbb{R}^m$ is called robust with respect to an input x_0 and an input region Ω around x_0 if $\forall x \in \Omega, \Phi(x) = \Phi(x_0)$ holds.*

Usually, input region Ω around input x_0 is defined by a ℓ_p-norm ball around x_0 with a radius of ϵ, i.e. $\mathbb{B}_p(x_0, \epsilon) = \{x | \|x - x_0\|_p \leq \epsilon\}$. In this paper, we focus on the *infinity norm* and verify the robustness of the neural network in $\mathbb{B}_\infty(x_0, \epsilon)$ on image classification tasks. A

corresponding robust verification problem is to compute the largest ϵ_0 s.t. neural network F is robust in $\mathbb{B}_\infty(x_0, \epsilon_0)$. The largest ϵ is called a *certified lower bound*, which is a metric for measuring both the robustness of neural networks and the precision of robustness verification approaches. Another problem is to compute the ratio of pictures that can be classified correctly when given a fixed ϵ, and that is called a verified robustness ratio.

Assuming that the output label of x_0 is c, i.e. $\Phi(x_0) = c$, proving F's robustness in Definition 1 is equivalent to showing $\forall x \in \Omega, \forall \ell \in L/\{c\}, F_c(x) - F_\ell(x) > 0$ holds. Thus, the verification problem is equivalent to solving the following optimization problem:

$$\min_{x \in \Omega}(F_c(x) - \max_{\ell \in L/\{c\}}(F_\ell(x))) \tag{1}$$

We can conclude that F is robust in Ω if the result is positive. Otherwise, there exists some input x' in Ω and ℓ' in $L/\{c\}$ such that $F_{\ell'}(x') \geq F_c(x')$. Namely, the probability of classifying x' by F to ℓ' is greater than or equal to the one to c, and consequently, x' may be classified as ℓ', meaning that F is not robust in Ω.

2.3 Verification via Linear Over-Approximation

The optimization problem in Equation 1 is computationally expensive, and it is almost impractical to compute the precise solution. The root reason for the high computational complexity of the problem is the non-linearity of the activation function σ. Even when σ is piece-wise linear, e.g., the commonly used ReLU ($\sigma(x) = \max(x, 0)$), the problem is NP-complete [17]. A pragmatic solution to simplify the verification problem is to over-approximate σ by linear constraints and transform it into an efficiently-solvable linear programming problem via interval propagation and approximation [43, 45].

Definition 2 (Linear Over-Approximation). *Let $\sigma(x)$ be a non-linear function on $[l, u]$ and $h_L(x) = \alpha_L x + \beta_L, h_U(x) = \alpha_U x + \beta_U$ be two linear functions for some $\alpha_L, \alpha_U, \beta_L, \beta_U \in \mathbb{R}$. $h_L(x)$ and $h_U(x)$ are called the lower and upper linear bounds of $\sigma(x)$ on $[l, u]$ respectively if $h_L(x) \leq \sigma(x) \leq h_U(x)$ holds for all x in $[l, u]$.*

By Definition 2, we can simplify Equation 1 to be the following efficiently solvable linear optimization problem. Note that z is no longer a number, but an interval here:

$$\begin{aligned} & \min(\min(z_c^{(m)}(x)) - \max(z_\ell^{(m)}(x))) \\ s.t. \quad & z^{(i)}(x) = W^{(i)}\hat{z}^{(i-1)}(x) + b^{(i)}, i \in 1, ..., m \\ & h_L^{(i)}(x) \leq \hat{z}^{(i)}(x) \leq h_U^{(i)}(x), i \in 1, ..., m-1 \\ & x \in \Omega, \ell \in L/c, \hat{z}^{(0)}(x) = [x, x] \end{aligned} \tag{2}$$

Example 1. *Let's consider an example in Figure 2, which shows the verification process of a simple neural network based on linear approximation. It is a fully-connected neural network with two hidden layers, x_1, x_2 are input neurons, and y_1, y_2 are output neurons. The intervals represent the range of neurons before the application of the activation function. We conduct linear bounds for each neuron with an activation function using the information of intervals. $h_{U,i}$ and $h_{L,i}$ are the upper and lower linear bounds of $\sigma(x_i)$ respectively. From the computed intervals of output neuron, we have $\min(y_1) - \max(y_2) > 0$ for all the possible (x_1, x_2) in the input domain $[-1, 1] \times [-1, 1]$. As a result, we can conclude that the network is robust in the input domain with respect to the class corresponding to y_1.*

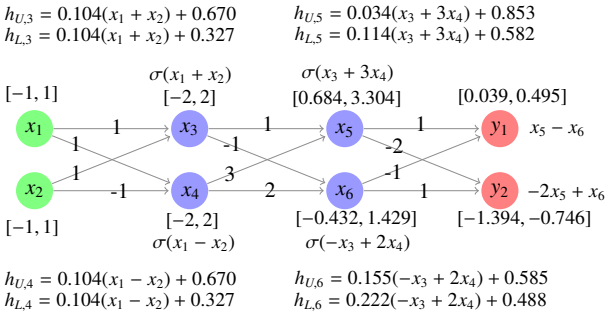


Figure 2: Verifying the robustness of a 4-layer Sigmoid network by the linear over-approximation.

The approximation inevitably introduces the overestimation of output ranges. In Example 1, the real output range of y_1 is $[0.127, 0.392]$, which is computed by solving the optimization problems of minimizing and maximizing y_1, respectively. The one computed by over-approximation is $[0.039, 0.495]$. We use the length of different intervals to measure the overestimation. The overestimation can be as large as 72.08%, even for a simple network.

The overestimation introduced by over-approximation may cause an actually robust case that cannot be verified, which is known as *incomplete*. For instance, when we have $\min(y_1) - \max(y_2) < 0$ by solving Problem 2, there are two possible reasons. One is that there exists some input such that the output on y_1 is indeed less than the one on y_2. The other is that the overestimation of y_1 and y_2 causes inequality. The network is robust in the latter case and not in the former. In this case, the algorithms just simply report *unknown* because they cannot determine which reason causes it.

Note that an activation function can be approximated more tightly using two more pieces of linear bounds. However, the one-piece linear approximation in Definition 2 is the most efficient because the reduced problem is a linear programming problem that can be efficiently solved in polynomial time. For piece-wise approximations, the number of linear bounds drastically blows up, and the corresponding reduced problem is proved NP-complete [37].

2.4 Variant Approximation Tightness Definitions

Reducing the overestimation of approximation is the key to reducing failure cases. The precision of approximation is characterized by the notion of *tightness* [55]. Many efforts have been made to define the tightest possible approximations. The tightness definitions can be classified into *neuron-wise* and *network-wise* categories.

1) Neuron-wise Tightness. The tightness of activation functions' approximations can be measured independently. Given two upper bounds $h_U(x)$ and $h'_U(x)$ of activation function $\sigma(x)$ on the interval $[l, u]$, $h_U(x)$ is apparently tighter than $h'_U(x)$ if $h_U(x) < h'_U(x)$ for any x in $[l, u]$ [26]. However, when $h_U(x)$ and $h'_U(x)$ intersect between $[l, u]$, their tightness becomes non-comparable. Another neuron-wise tightness metric is the area size of the gap between the bound and the activation function, i.e., $\int_l^u (h_U(x) - \sigma(x))dx$. A smaller area implies a tighter approximation [12, 28]. Apparently, an over-approximation that is tighter than another by the definition of [26] is also tighter by the definition of [12], but not vice versa. What's more, another metric is the output range of the

linear bounds. An approximation is considered to be *the tightest* if it preserves the same output range as the activation function [55].

2) Network-wise Tightness. Recent studies have shown that neuron-wise tightness does not always guarantee that the compound of all the approximations of the activation functions in a network is tight too [55]. This finding explains why the so-called tightest approaches based on their neuron-wise tightness metrics achieve the best verification results only for certain networks. It inspires new approximation approaches that consider multiple and even all the activation functions in a network to approximate simultaneously. The compound of all the activation functions' approximations is called the network-wise tightest with respect to an output neuron if the neuron's output range is the precisest.

Unfortunately, finding the network-wise tightest approximation has been proved a non-convex optimization problem, and thus computationally expensive [26, 55]. From a pragmatic viewpoint, a neuron-wise tight approximation is useful if all the neurons' composition is also network-wise tight. The work [55] shows that there exists such a neuron-wise tight approach under certain constraints when the networks are monotonic. However, their approach does not guarantee to be optimal when the neural networks contain both positive and negative weights.

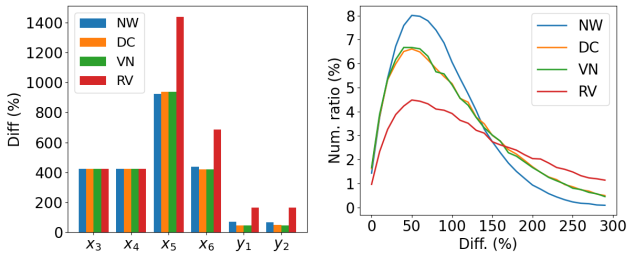
3 MOTIVATION

In this section, we show that the over-approximation inevitably introduces overestimation. We observe that the overestimation of intervals is accumulatively propagated as the intervals are propagated layer by layer. Based on this insight, we propose an important factor called *approximation domain* for explaining why existing so-called tightest over-approximation approaches still introduce large overestimation. In the end, we illustrate that the precise approximation domain and the tight over-approximation are interdependent.

3.1 Interval Overestimation Propagation

Neural network verification methods based on over-approximation will inevitably introduce overestimation more or less, as shown in Section 2.3. For the approximation of an activation function, if the maximum value of its upper bound is larger than the maximum value of the actual value, or the minimum value of its lower bound is smaller than the minimum value of the actual value, the approximation is imprecise and introduces too much overestimation.

In our experiment, we found that the overestimation can be accumulated and propagated layer by layer. We evaluate four state-of-the-art linear over-approximation approaches, including NeWise (NW) [55], DeepCert (DC) [49], VeriNet (VN) [12], and RobustVerifier (RV) [23], to verify the neural network model defined in Figure 2 with 50,000 different weights and input intervals, and record the size of real intervals and overestimated intervals during the process. Figure 3 shows the overestimation of each neuron with the network in Figure 2, together with the overestimation distribution of the 50,000 cases for y_1 and y_2. In Figure 3a, the overestimation is over 400% for x_3, x_4, x_5, x_6, and around 50% for y_1, y_2. Note that the overestimation of y_1 and y_2 is smaller than the ones in the hidden layers. This is due to the non-monotonicity of neural networks and the normalization of activation functions. We can find in Figure 3b that most of the overestimations are distributed between 50% and 100%, while it is over 300% in more than 10% cases.



(a) Overestimation on a network. (b) Overestimation distribution.

Figure 3: The overestimation of each neuron in the network in Figure 2 (a) and the overestimation distribution on 50,000 variant networks with the same network architecture (b).

There are two main reasons for the accumulative propagation. One apparent reason is the over-approximations of activation functions, which is inevitable but can be reduced by defining tight ones. The other reason is that over-approximations must be defined on overestimated domains of the activation function to guarantee the soundness of it. This further introduces overestimation to approximations as the domains' overestimation increases. Due to the layer dependency in neural networks, such dual overestimation is accumulated and propagated to the output layer.

3.2 Approximation Domain

To justify the second reason for the accumulative propagation in Section 3.1, we introduce the notion of *approximation domain*, to represent the domain of activation functions, on which over-approximations are defined.

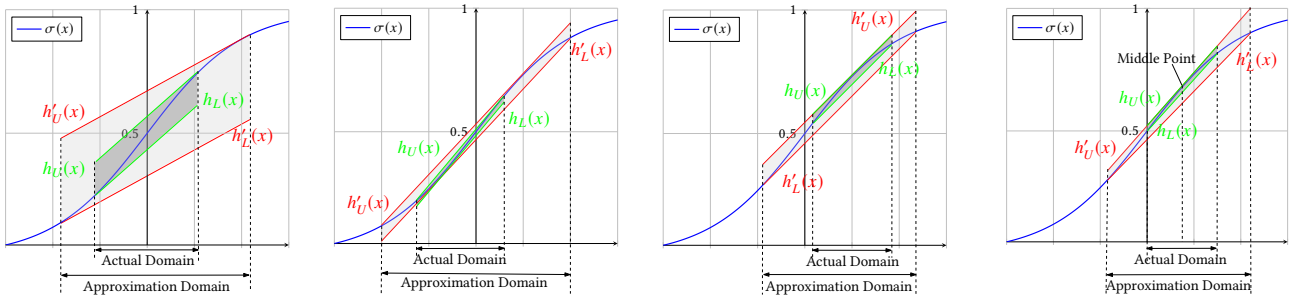
Definition 3 (Approximation Domain). *Given a neural network F and an input region $\mathbb{B}_\infty(x_0, \epsilon)$, the approximation domain of the r-th hidden neuron in the i-th layer is $[l_r^{(i)}, u_r^{(i)}]$, where,*

$$
\begin{aligned}
& l_r^{(i)} = \min z_r^{(i)}(x), u_r^{(i)} = \max z_r^{(i)}(x) \\
& s.t. \quad z^{(j)}(x) = W^{(j)} \hat{z}^{j-1}(x) + b^{(j)}, j \in 1, ..., i \\
& \qquad h_L(z^{(j)}(x)) \leq \hat{z}^{(j)}(x) \leq h_U(z^{(j)}(x)), j \in 1, ..., i-1 \\
& \qquad x \in \mathbb{B}_\infty(x_0, \epsilon), \hat{z}^{(0)}(x) = x
\end{aligned}
$$

Definition 3 formulates the way of the existing over-approximation approaches [12, 49, 53, 55] to compute overestimated domains of activation functions for defining their over-approximations. Given two different approximation domains $[l_r, u_r]$ and $[l'_r, u'_r]$, we say $[l_r, u_r]$ is more precise than $[l'_r, u'_r]$ if $l_r \geq l'_r$ and $u_r \leq u'_r$. Let us consider the activation functions on neurons x_5 and x_6 in Figure 2 as an example. Their domains are estimated based on the approximations of x_3 and x_4 by solving the corresponding linear programming problems in Definition 3. The approximation domains are $[0.684, 3.304]$ and $[-0.432, 1.429]$, respectively. As shown in Figure 3a, they are much overestimated compared with the actual ones.

3.3 The Overestimation Interdependency

We show the interdependency between the two problems of defining tight over-approximations for activation functions and computing the precise approximation domains. The interdependency



(a) Tangent line at end points [55]. (b) Minimal area [12]. (c) Parallel line [49, 53]. (d) Tangent line at middle point [12].

Figure 4: The differences between the linear over-approximations that are defined on the estimated approximation and the actual one respectively according to the four state-of-the-art approaches. The red lines refer to the upper and lower bounds defined on approximation domains, and the green lines refer to those defined on actual domains. The light and dark gray areas represent the corresponding overestimation introduced by the over-approximations, respectively.

means that tighter over-approximations of activation functions result in more precise approximation domains and vice versa. Here we follow the approximation tightness definition in [26], by which a lower bound $h_L(x)$ is called tighter than another $h'_L(x)$ if $h_L(x) \geq h'_L(x)$ holds for all x in the approximation domain of σ. Likewise, an upper bound $h_U(x)$ is tighter than another $h'_U(x)$ if $h_U(x) \leq h'_U(x)$. Apparently, a tighter approximation by definition [26] is still tighter by the minimal-area-based definition [12].

Theorem 3.1. *Suppose that there are two over-approximations* $h_L(z^{(j)}(x)), h_U(z^{(j)}(x))$ *and* $h'_L(z^{(j)}(x)), h'_U(z^{(j)}(x))$ *for each* $z^{(j)}(x)$ *in Definition 3 and* $h_L(z^{(j)}(x)), h_U(z^{(j)}(x))$ *are tighter than* $h'_L(z^{(j)}(x)), h'_U(z^{(j)}(x))$, *respectively. The approximation domain* $[l_r^{(i)}, u_r^{(i)}]$ *computed by* $h_L(z^{(j)}(x)), h_U(z^{(j)}(x))$ *must be more precise than the one* $[l'^{(i)}_r, u'^{(i)}_r]$ *by* $h'_L(z^{(j)}(x)), h'_U(z^{(j)}(x))$.

Intuitively, Theorem 3.1 claims that tighter approximations lead to more precise approximation domains for the activation functions of the neurons in subsequent layers of a DNN.

Theorem 3.2. *Given two approximation domains* $[l_r^{(i)}, u_r^{(i)}]$ *and* $[l'^{(i)}_r, u'^{(i)}_r]$ *such that* $l'^{(i)}_r < l_r^{(i)}$ *and* $u_r^{(i)} < u'^{(i)}_r$, *for any over-approximation* $(h'_L(z^{(j)}(x)), h'_U(z^{(j)}(x)))$ *of continuous function* $\sigma(x)$ *on* $[l'^{(i)}_r, u'^{(i)}_r]$, *there exists an over-approximation* $(h_L(z^{(j)}(x)), h_U(z^{(j)}(x)))$ *on* $[l_r^{(i)}, u_r^{(i)}]$ *such that* $\forall z^{(j)}(x) \in [l_r^{(i)}, u_r^{(i)}], h'_L(x) \leq h_L(z^{(j)}(x)), h'_U(x) \geq h_U(z^{(j)}(x))$.

Theorem 3.2 claims that more precise approximation domains lead to tighter over-approximations of the activation function. Altogether, the two theorems preserve the tightness of an approximation through propagation in a neural network. The proofs are given in Appendix A of the submitted technical report.

The above two theorems show the overestimation interdependency of the two problems. As examples, Figure 4 depicts the differences between the over-approximations that are defined on overestimated approximation domains and the actual domains based on the corresponding approximation approaches [12, 49, 53, 55]. Apparently, there exist much tighter over-approximations if we can reduce the overestimation of approximation domains. These examples show the importance of more precise approximation domains for activation functions to define tighter over-approximations.

It is worth mentioning that it is almost impractical to define over-approximations directly on the actual domain of activation functions for non-trivial neural networks, e.g., those which have two or more hidden layers. That is because computing the actual domains is at least as computationally expensive as the neural network verification problem itself. If we could compute the domains for all the activation functions on hidden neurons, the robustness verification problem would then be efficiently achieved by solving the linear constraints between the last hidden layer and the output layer using linear programming.

4 THE DUAL-APPROXIMATION APPROACH

In this section, we present our dual-approximation approach for defining tight over-approximation for activation functions guided by under-approximations. Specifically, we propose two algorithms to compute underestimated approximation domains for each activation function and define different over-approximation strategies according to both overestimated and underestimated domains.

4.1 Approach Overview

Figure 5 depicts an illustration of our dual-approximation approach and the comparisons with those approaches defined only based on approximation domains. For each activation function, we compute an overestimated and an underestimated approximation domain, denoted by $[l_{over}, u_{over}]$ and $[l_{under}, u_{under}]$, respectively. Underestimated domains provide useful information for defining over-approximations. Let $h(x)$ be a linear lower or upper bound of σ on the interval $[l_{under}, u_{under}]$. We take it as a linear over-approximation lower or upper bound for σ on the approximation domain $[l_{over}, u_{over}]$ if it satisfies the condition in Definition 2. According to Theorem 3.2, we can define a tighter $h(x)$ on $[l_{under}, u_{under}]$ than those defined on $[l_{over}, u_{over}]$ and make sure $h(x)$ is a valid over-approximation bound on $[l_{over}, u_{over}]$. Thus, the underestimated domain is used to guarantee the over-approximation's tightness, while the approximation domain to guarantee its soundness. We therefore call it a *dual-approximation* approach.

As shown in Figure 5, we take the tangent line at $(l_{under}, \sigma(l_{under}))$ as the lower bound of σ. Apparently, this lower bound is much tighter than the one defined by the tangent line at $(l_{over}, \sigma(l_{over}))$ (the dark green line according to the approach by NeWise [55]) on

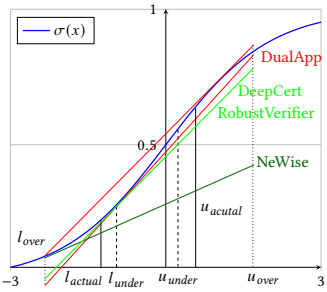


Figure 5: An illustration of our dual-approximation approach and comparison with other over-approximation approaches.

the actual domain $[l_{actual}, u_{actual}]$ of σ. It is also tighter than the green tangent line parallel to the upper bound, according to the approaches in DeepCert [49] and RobustVerifier [23].

4.2 Under-Approximation Algorithms

We introduce two approaches, i.e., *Monte Carlo* and *gradient-based*, for underestimating the actual domain of the activation functions. In other word, we propose two strategies to compute $[l_{under}, u_{under}]$. The two strategies are complementary in that the former is more efficient but computes less precise underestimated input domain, while the latter performs in the opposite way.

The Monte Carlo Approach. A simple yet efficient approach for under-approximation is to randomly generate a number of valid samples and feed them into the network to track the reachable bounds of each hidden neuron's input. A sample is valid if the distance between it and the original input is less than a preset perturbation distance ϵ.

Algorithm 1 shows the pseudo-code of the Monte Carlo approach. First, we randomly generate n valid samples from $\mathbb{B}_\infty(x_0, \epsilon)$ (Line 1) and initialize the lower and upper bounds $l_{L,r}^{(i)}$ and $u_{L,r}^{(i)}$ of each hidden neuron (Line 2). Then we feed each sample into the network (Line 3), record the input value $v_{p,r}^{(i)}$ of each activation function (Line 6), and update the corresponding lower or upper bound by $v_{L,r}^{(i)}$ (Lines 7-8). The time complexity of Algorithm 1 is $O(n\sum_{i=1}^{m} k_i k_{i-1})$, where m refers to the layer of the neural network, and k_i refers to the number of neurons of layer i.

The Gradient-Based Approach. The conductivity of neural networks allows us to approximate the actual domain of each hidden neuron by gradient descent [35]. The basic idea of gradient descent is to compute two valid samples according to the gradient of an objective function to minimize and maximize the output value of the function, respectively. Using gradient descent, we can compute locally optimal lower and upper bounds as the underestimated input domains of activation functions.

Algorithm 2 shows the pseudo-code of the gradient-based approach. Its inputs include a neural network F, an input x_0 of F, an ℓ_∞-norm radius ϵ, and a step length a of gradient descent. It returns an underestimated input domain for each neuron on the hidden layers. It gets function $F_r^{(i)}$ of the neural network on neuron r (Line 3), computes its gradient, and records its sign η_r^i as the direction to update x_0 (Line 4). Then, the gradient descent is conducted one step forward to generate a new input x_{lower} (Line 5). x_{lower} is then

Algorithm 1: The Monte Carlo Approach.

Input : F: a network; x_0: an input to F; ϵ: a ℓ_∞-norm radius; n: number of samples
Output: $l_{L,r}^{(i)}, u_{L,r}^{(i)}$ for each hidden neuron r on layer i

1 Randomly generate n samples S_n from $\mathbb{B}_\infty(x_0, \epsilon)$;
2 $l_L \leftarrow \infty, u_L \leftarrow -\infty$; *// Initialize all upper and lower bounds*
3 **for** *each sample* x_p *in* S_n **do**
4 **for** *each hidden layer* i **do**
5 **for** *each neuron* r *on layer* i **do**
6 $v_{p,r}^{(i)} := F_r^{(i)}(x_p)$; *// Compute the output of neuron r*
7 $l_{L,r}^{(i)} \leftarrow \min(l_{L,r}^{(i)}, v_{p,r}^{(i)})$; *// Update r's lower bound*
8 $u_{L,r}^{(i)} \leftarrow \max(u_{L,r}^{(i)}, v_{p,r}^{(i)})$; *// Update r's upper bound*

Algorithm 2: The Gradient-Based Approach.

Input : F: a network; x_0: an input to F; ϵ: a ℓ_∞-norm radius; a: the step length of gradient descent
Output: $l_{L,r}^{(i)}, u_{L,r}^{(i)}$ for each neuron r in each hidden layer i

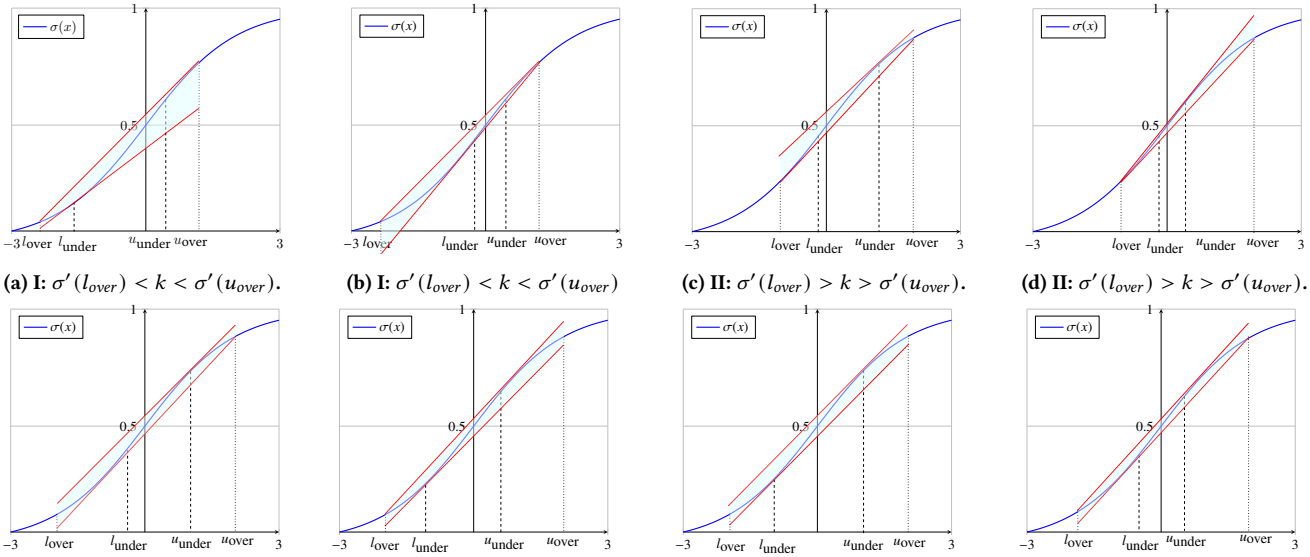
1 **for** *each hidden layer* $i = 1, \ldots, m$ **do**
2 **for** *each neuron* r *on layer* i **do**
3 Get the function $F_r^{(i)}$ of neuron r;
4 $\eta_r^{(i)} \leftarrow sign(F_r^{(i)'}(x_0))$; *// Get the sign of gradient of r*
5 $x_{lower} \leftarrow x_0 - a\eta_r^{(i)}$; *// One-step forward*
6 Cut x_{lower} s.t. $x_{lower} \in \mathbb{B}_\infty(x_0, \epsilon)$; *// Make x_{lower} valid*
7 $l_{L,r}^{(i)} \leftarrow F_r^{(i)}(x_{lower})$; *// Compute and store the lower bound*
8 $x_{upper} \leftarrow x_0 + a\eta_r^{(i)}$; *// Compute the upper case*
9 Cut x_{upper} s.t. $x_{upper} \in \mathbb{B}_\infty(x_0, \epsilon)$
10 $u_{L,r}^{(i)} \leftarrow F_r^{(i)}(x_{upper})$; *// Compute and store the upper bound*

modified to make sure it is in the normal ball. By feeding x_{lower} to $F_r^{(i)}$, we obtain an under-approximated lower bound $l_{L,r}^{(i)}$ (Line 7). The upper bound can be computed likewise (Lines 8-10).

Considering the time complexity of Algorithm 2, we need to compute the gradient for each neuron on the ith hidden layer, of which time complexity is $O(\sum_{j=1}^{i} k_j k_{j-1})$. Thus, given an m-hidden-layer network, the time complexity of the gradient-based algorithm is $O(\sum_{i=1}^{m} k_i(\sum_{j=1}^{i} k_j k_{j-1}))$. This is higher than the time complexity of the Monte Carlo algorithm, while it obtains a more precise underestimated domain. We compare the efficiency and effectiveness of the two algorithms in Experiment II.

4.3 Over-Approximation Strategies

We omit the superscript and subscript and consider finding the approximation method of $\sigma(x)$ with the information of upper and lower approximation domains. We assume that the lower approximation domain of input x is $[l_{under}, u_{under}]$ and the upper approximation interval is $[l_{over}, u_{over}]$. As in [49], we consider three cases according to the relation between the slopes of σ at the two endpoints of upper approximation interval $\sigma'(l_{over})$, $\sigma'(u_{over})$ and $k = \frac{\sigma(u_{over}) - \sigma(l_{over})}{u_{over} - l_{over}}$.



(a) I: $\sigma'(l_{over}) < k < \sigma'(u_{over})$. **(b) I:** $\sigma'(l_{over}) < k < \sigma'(u_{over})$ **(c) II:** $\sigma'(l_{over}) > k > \sigma'(u_{over})$. **(d) II:** $\sigma'(l_{over}) > k > \sigma'(u_{over})$.

(e) III: $\sigma'(l_{over}) < k \wedge \sigma'(u_{over}) < k$. **(f) III:** $\sigma'(l_{over}) < k \wedge \sigma'(u_{over}) < k$. **(g) III:** $\sigma'(l_{over}) < k \wedge \sigma'(u_{over}) < k$. **(h) III:** $\sigma'(l_{over}) < k \wedge \sigma'(u_{over}) < k$.

Figure 6: The linear over-approximation based on overestimated and underestimated approximation domains.

Case I. When $\sigma'(l_{over}) < k < \sigma'(u_{over})$, the line connecting the two endpoints is the upper bound. For the lower bound, the tangent line of σ at l_{under} is chosen if it is sound (Figure 6a), otherwise the tangent line of σ at d crossing $(u_{over}, \sigma(u_{over}))$ is chosen (Figure 6b). Namely, we have $h_U(x) = k(x - u_{over}) + \sigma(u_{over})$, and

$$h_L(x) = \begin{cases} \sigma'(l_{under})(x - l_{under}) + \sigma(l_{under}), & l_{under} < d \\ \sigma'(d)(x - d) + \sigma(d), & l_{under} \geq d. \end{cases} \quad (3)$$

Case II. When $\sigma'(l_{over}) > k > \sigma'(u_{over})$, it is the symmetry of Case 1. the line connecting the two endpoints can be the lower bound. For upper bound, the tangent line of σ at u_{under} is chosen if it is sound (Figure 6c), otherwise the tangent line of σ at d crossing $(l_{under}, \sigma(l_{under}))$ is chosen (Figure 6d). That is, $h_L(x) = k(x - l_{over}) + \sigma(l_{over})$, and

$$h_U(x) = \begin{cases} \sigma'(u_{under})(x - u_{under}) + \sigma(u_{under}), & u_{under} > d \\ \sigma'(d)(x - d) + \sigma(d), & u_{under} \leq d. \end{cases} \quad (4)$$

Case III. When $\sigma'(l_{over}) < k$ and $\sigma'(u_{over}) < k$, we first consider the upper bound. If the tangent line of σ at u_{under} is sound, we choose it to be the upper bound (Figure 6e and Fig 6g); otherwise we choose the tangent line of σ at d_1 crossing $(l_{under}, \sigma(l_{under}))$ (Figure 6f and Figure 6h). Then we consider the lower bound. The tangent line of σ at l_{under} is chosen if it is sound (Figure 6f and Figure 6g), otherwise we choose the tangent line of σ at d_2 crossing $(u_{over}, \sigma(u_{over}))$ (Figure 6e and Figure 6h). Namely, we have:

$$h_U(x) = \begin{cases} \sigma'(u_{under})(x - u_{under}) + \sigma(u_{under}), & u_{under} > d_1 \\ \sigma'(d_1)(x - d_1) + \sigma(d_1), & u_{under} \leq d_1, \end{cases} \quad (5)$$

$$h_L(x) = \begin{cases} \sigma'(l_{under})(x - l_{under}) + \sigma(l_{under}), & l_{under} < d_2 \\ \sigma'(d_2)(x - d_2) + \sigma(d_2), & l_{under} \geq d_2. \end{cases} \quad (6)$$

The goal of our approximation strategy is to make the over-estimated output interval as close as possible to the actual one

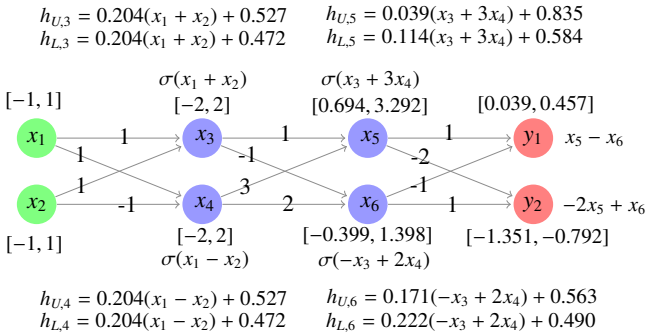
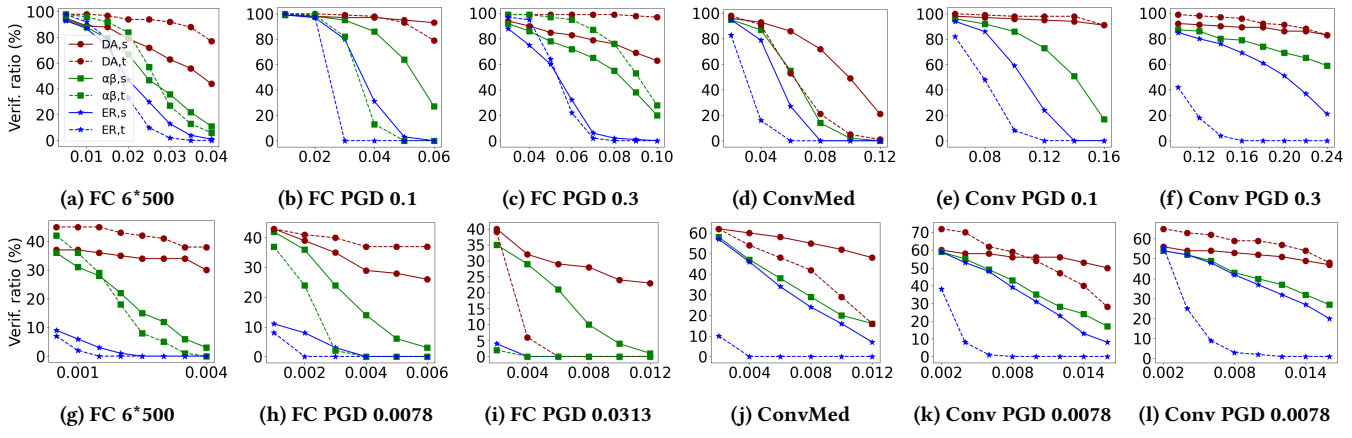


Figure 7: The approximations defined in our approach and the propagated intervals for the network in Figure 2.

of each hidden neuron. An over-approximation is the provably tightest neuron-wise if it preserves the same output interval as the activation function on the actual domain [55]. As described in Theorem 3.2, a more precise range allows us to define a tighter over-approximation. Under the premise of guaranteeing soundness, we use the guiding significance of the underestimated domain to make the over-approximation closer to the actual domain so as to obtain more precise approximation bounds. Through layer-by-layer transmission, we obtain more accurate intervals for the deeper hidden neurons (by Theorem 3.1), on which tighter over-approximations can be defined (by Theorem 3.2). In this way, the overestimation interdependency during defining over-approximations is alleviated by computing the underestimated domains.

Example 2. *We reconsider the network in Figure 2 and define tighter over-approximations with our approach. Figure 7 shows the approximations and the propagated intervals for neurons in hidden layers and the output layer. As x_3, x_4 have precise input intervals, only the activation functions of x_5, x_6 need to be under-approximated. Thus, we only need to redefine their approximations according to our*



(a) FC 6*500 (b) FC PGD 0.1 (c) FC PGD 0.3 (d) ConvMed (e) Conv PGD 0.1 (f) Conv PGD 0.3

(g) FC 6*500 (h) FC PGD 0.0078 (i) FC PGD 0.0313 (j) ConvMed (k) Conv PGD 0.0078 (l) Conv PGD 0.0078

Figure 8: Comparison of the verified robustness rate of DualApp, α-β-CROWN, and ERAN on 24 neural networks and two datasets. (a-f) refer to the comparison on MNIST while (g-l) to CIFAR-10. The horizontal axis represents the perturbation ϵ, and the vertical axis represents the ratio of images that are verified robustness. Legend DA, s refers to the result of DualApp on the neural network with sigmoid activation function, and ER, t refers to the result of ERAN with tanh.

approach. We underestimate the input domains of x_5, x_6 and use them to guide the over-approximations in our approach. We achieve 9.74% and 0.27% reductions for the overestimation of y_1, y_2's output ranges.

Example 2 demonstrates the effectiveness of our proposed dual-approximation approach even when it is applied to only one hidden layer. That is, the overestimation of propagated intervals can be further reduced through defining tighter over-approximations based on both underestimated and approximation domains. Tighter over-approximations can produce more precise verification results, and we will show this experimentally in the next section.

5 IMPLEMENTATION AND EVALUATION

We implement our dual-approximation approach into a tool called DualApp. We assess it, as well as six state-of-the-art approaches, with respect to the DNNs with the S-curve activation functions. Our goal is threefold:

(1) to demonstrate that, compared to the state of the art, DualApp is more effective on robustness verification results;
(2) to explore the hyper-parameter space for our two methods that leverage under-approximation; and
(3) to measure the trade-off between these two complementary methods.

5.1 Benchmark and Implementation

Competitors. We consider six state-of-the-art DNN robustness verification tools: α-β-CROWN [52], ERAN [28], NeWise [55], DeepCert [49], VeriNet [12], and RobustVerifier [23]. They rely on the over-approximation domain to define and optimize linear lower and upper bounds except that ERAN is based on the abstract domain combining floating point polyhedra with intervals [37].

Datasets and Neural Networks. In the comparison experiment with α-β-CROWN and ERAN, we collect 24 models exposed by ERAN for MNIST [19] and CIFAR-10 [18], including CNNs and FNNs trained with normal training method and adversarial training method with PGD attack [44]. These models contain Sigmoid and Tanh activation functions. For NeWise, DeepCert, VeriNet, and RobustVerifier, we collect and train totally 84 convolutional neural networks (CNNs) and fully-connected neural networks (FNNs) on image datasets MNIST, Fashion MNIST [50] and CIFAR-10. Sigmoid, Tanh, and Arctan are contained in these models, respectively. As most researches are based on the ReLU activation function, there are few public neural networks in benchmarks with S-curved activate functions, and their sizes are small.

Metrics. We use two metrics in our comparisons: (i) *verified robustness ratio*, which is the percentage of images that must be correctly classified under a fixed perturbation ϵ, and (ii) *certified lower bound*, which is the largest perturbation ϵ within which all input images must be correctly classified. We consider strong baselines in that we assess DualApp on the benchmarks and metrics for which the competitors report the optimal performance. In particular, we use (i) for the comparison with α-β-CROWN and ERAN as both report the highest verified robustness rate as in, e.g., the 2022 VNN-COMP competition [29].

Implementation. For a fair comparison, we implemented the approximation strategies of NeWise, DeepCert, VeriNet, RobustVerifier, and our dual-approximation algorithm in DualApp using python with the TensorFlow framework. We apply the method used in NeWise to train neural networks and load datasets. We implement the algorithms and strategies defined in Section 4 to compute under-approximation domains and linear upper and lower bounds, thus obtaining the final output intervals and verification results for each image. To compute the certified lower bound, we set the initial value of ϵ to 0.05 and update it 15 times using the dichotomy method based on the verification results.

Experimental Setup. We conducted all the experiments on a workstation equipped with a 32-core AMD Ryzen Threadripper PRO 5975WX CPU and 256GB RAM running Ubuntu 22.04.

5.2 Experimental Results

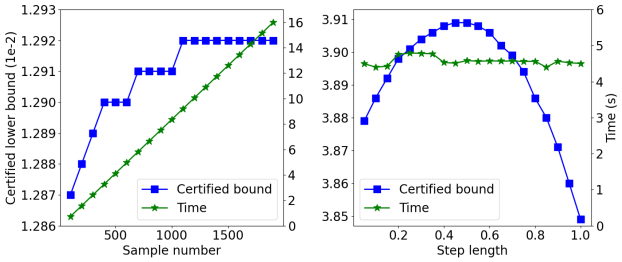
Experiment I: Comparisons with the Competitors. Figure 8 shows the comparison results among our approach with the Monte

Table 1: Comparing the DualApp (DA) and four state-of-the-art tools including NeWise (NW), DeepCert (DC), VeriNet (VN), and RobustVerifier (RV) on the CNNs and FNNs with the Sigmoid activation function. CNN_{l-k} denotes a CNN with l layers and k filters of size 3×3 on each layer. $\text{FNN}_{l \times k}$ denotes a FNN with l layers and k neurons on each layer.

Dataset	Model	Nodes	DA	NW		DC		VN		RV		DA	Others
			Bounds	Bounds	Impr. (%)	Bounds	Impr. (%)	Bounds	Impr. (%)	Bounds	Impr. (%)	Time (s)	Time(s)
Mnist	CNN_{4-5}	8,690	0.05819	0.05698	2.12	0.05394	7.88	0.05425	7.26	0.05220	11.48	14.70	0.98 ± 0.02
	CNN_{5-5}	10,690	0.05985	0.05813	2.96	0.05481	9.20	0.05503	8.76	0.05125	16.78	20.13	2.67 ± 0.29
	CNN_{6-5}	12,300	0.06450	0.06235	3.45	0.05898	9.36	0.05882	9.66	0.05409	19.25	25.09	4.86 ± 0.34
	CNN_{8-5}	14,570	0.11412	0.09559	19.38	0.08782	29.95	0.08819	29.40	0.06853	66.53	34.39	11.89 ± 0.21
	$\text{FNN}_{5\times100}$	510	0.00633	0.00575	10.09	0.00607	4.28	0.00616	2.76	0.00519	21.97	7.10	0.79 ± 0.05
	$\text{FNN}_{6\times200}$	1,210	0.02969	0.02909	2.06	0.02511	18.24	0.02829	4.95	0.01811	63.94	8.64	2.82 ± 0.34
Fashion Mnist	CNN_{4-5}	8,690	0.07703	0.07473	3.08	0.07204	6.93	0.07200	6.99	0.06663	15.61	15.26	1.06 ± 0.09
	CNN_{5-5}	10,690	0.07288	0.07044	3.46	0.06764	7.75	0.06764	7.75	0.06046	20.54	20.95	3.18 ± 0.42
	CNN_{6-5}	12,300	0.07655	0.07350	4.15	0.06949	10.16	0.06910	10.78	0.06265	22.19	25.96	5.63 ± 0.77
	$\text{FNN}_{1\times50}$	60	0.03616	0.03284	10.11	0.03511	2.99	0.03560	1.57	0.02922	23.75	0.84	0.02 ± 0.00
	$\text{FNN}_{5\times100}$	510	0.00801	0.00710	12.82	0.00776	3.22	0.00789	1.52	0.00656	22.10	2.98	0.65 ± 0.00
Cifar-10	CNN_{3-2}	2,514	0.03197	0.03138	1.88	0.03120	2.47	0.03119	2.50	0.03105	2.96	5.54	0.32 ± 0.02
	CNN_{5-5}	10,690	0.01973	0.01926	2.44	0.01921	2.71	0.01913	3.14	0.01864	5.85	31.45	4.86 ± 0.41
	CNN_{6-5}	12,300	0.02338	0.02289	2.14	0.02240	4.38	0.02234	4.66	0.02124	10.08	43.51	10.53 ± 0.67
	$\text{FNN}_{5\times100}$	510	0.00370	0.00329	12.46	0.00368	0.54	0.00368	0.54	0.00331	11.78	2.97	0.64 ± 0.01
	$\text{FNN}_{3\times700}$	2,110	0.00428	0.00348	22.99	0.00427	0.23	0.00426	0.47	0.00397	7.81	32.68	10.85 ± 0.58

Carlo algorithm, α-β-CROWN, and ERAN on 24 networks with Sigmoid and Tanh activation functions. In this experiment, the perturbation ϵ is set to increase in a fixed step for each network. Following the strategy for choosing image samples in all the competing tools, we choose the first 100 images from the corresponding test set to verify. We take 1000 samples for each image to compute the underestimated domain in our method. The experimental results strongly show our method can reduce overestimation and compute higher verified robustness ratios. In most cases, our method improves by *dozens to hundreds of percent* compared with the other two methods. In particular, the improvement becomes significantly greater as the perturbation ϵ increases. For example, in Figure 8b, the improvement of our method relative to α-β-CROWN is 12.79% when $\epsilon = 0.04$ on Sigmoid networks, and the number reaches 244.44% when ϵ enlarge to 0.06. In Figure 8l, the improvement even reaches 5600% compared with ERAN on Tanh networks when $\epsilon = 0.012$. That is because large perturbations imply large input intervals and consequently large overestimation of approximation domains. The underestimated domains become more dominant in defining tight over-approximations. These experimental results further demonstrate the importance of underestimated domains in tightening the over-approximations.

Table 1 shows the comparison results between our approach with the Monte Carlo algorithm and the other four tools on 16 networks with Sigmoid activation function. We randomly choose 100 inputs from each test set and compute the average of their certified lower bounds. In our method, we take 1000 samples for each image to compute the underestimated domain. The result shows that our approach outperforms all four competitors in all cases. On average, the improvement of our approach achieves 10.64% compared to others. Regarding efficiency, our Monte Carlo approach takes a little more time than other tools because of the sampling procedure. We trade time for a more precise approximation. For Tanh and Arctan neural networks, our approach also performs best on these models among all the tools. We defer the results to Appendix B.1 of our accompanying technical report.



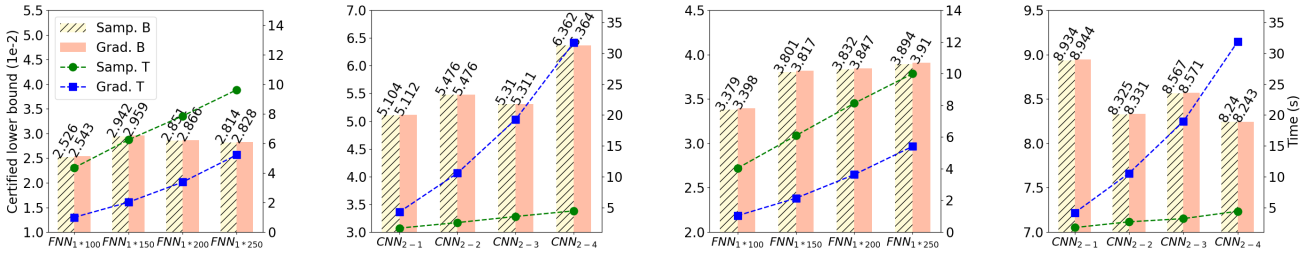
(a) Monte Carlo algorithm **(b) Gradient-based algorithm**

Figure 9: Exploring the influence of sample number and step length on certified lower bounds.

To conclude, the experimental results demonstrate that, compared with those approaches that rely only on approximation domains, our dual-approximation approach introduces less overestimation and returns more precise verification results on both robustness rates and certified bounds. The improvement is even more significant with larger perturbations.

Experiment II: Hyper-parameters. As approximation-based verification is intrinsically incomplete and the optimal values of hyper-parameters are unknowable, it is important to explore the hyper-parameter space for more effective and more efficient verification. Hence, we measure the impacts of the two hyper-parameters, i.e., the sample number and the step length of gradient descent, in our Monte Carlo and gradient-based algorithms, respectively, with respect to the verification results and time.

We conduct the experiments on eight neural networks trained on MNIST and Fashion MNIST, respectively. Figure 9a shows the relation between certified lower bounds and the number of samples, resp. the time cost, for an FNN_{1*150} trained on Fashion MNIST. The complete results are given in Appendix B.2 of our technical report. The computed bound is monotonously increasing with more samples, and stabilizes after 1000 samples, which indicates that the under-approximation domain cannot be improved by simply



(a) FNNs on MNIST. (b) CNNs on MNIST. (c) FNNs on Fashion MNIST. (d) FNNs on Fashion MNIST.

Figure 10: Comparison between the Monte Carlo and gradient-based algorithms for the robustness verification on 4 FNNs and 4 CNNs. The sample number is set to 1000 while the step length to 0.45.

running more simulations. For efficiency, we observe a linear relationship between the number of samples and the overhead.

Figure 9b shows the result of the gradient-based algorithm for the same FNN_{1*150}. We consider the step length from 0.05ϵ to ϵ by step of 0.05ϵ. It indicates that when the step length is set around 0.45ϵ, the computed bound is maximal. This conclusion is applicable to other networks and perturbation ϵ. The time cost is almost the same and independent of the step length.

Experiment III: Monte Carlo vs. Gradient. We evaluate the performance of our two under-approximation algorithms. Figure 10 shows the certified lower bounds and the time cost of two algorithms on eight FNNs and eight CNNs with the Sigmoid activation function, respectively. We set the sample number to 1000 and step length to 0.45 (the optimal hyper-parameters from Experiment II). We observe that, compared to the Monte Carlo algorithm, the certified lower bounds computed by the gradient-based algorithm are always larger. The reason is that local information of neural network functions such as monotonicity can be obtained through gradients for computing more precise underestimated domains. This further demonstrates the usefulness of underestimated domains in defining tight over-approximations. For efficiency, the gradient-based algorithm costs less time on simple FNNs, while more time on complex CNNs. With small-sized networks, the gradient-based algorithm is faster as fewer steps are required in computing the gradient.

6 RELATED WORK

This work is a sequel to existing efforts on approximation-based DNN robustness analysis. We classify them into two categories.

Over-approximation approaches. Due to the intrinsic complexity in the neural network robustness verification, approximating the non-linear activation functions is the mainstreaming approach for scalability. Zhang *et al.* defined three cases for over-approximating S-curved activation functions [53]. Wu and Zhang proposed a fine-grained approach and identified five cases for defining tighter approximations [49]. Lyu *et al.* proposed to define tight approximations by optimization at the price of sacrificing efficiency. Henriksen and Lomuscio [12] defined tight approximations by minimizing the gap area between the bound and the curve. However, all these approaches are proved superior to others only on specific networks [55]. The approximation approach in [55] is proved to be the tightest when the networks are monotonous. All these approaches only consider overestimated approximation domains. Paulsen and Wang recently proposed an interesting approach for synthesizing tight approximations guided by generated examples [32, 33]. Similar to our approach, these approaches compute sound and tight over-approximations from unsound templates. However, they require global optimization techniques to guarantee soundness, while our approach ensures the soundness of individual neurons statistically.

Under-approximation approaches. The essence of our dual approximation approach is to underestimate activation functions' domains for guiding the definition of tight over-approximations. There are several related under-approximation approaches based on either white-box attacks [5] or testings [11]. For instance, the fast gradient sign method (FGSM) [9] is a well-known approach for generating adversarial examples to intrigue corner cases for classifications. Other attack approaches include C&W [4], DeepFool [27], and JSMA [31]. The white-box testing for neural networks is to generate specific test cases to intrigue target neurons under different coverage criteria. Various test case generation and selection approaches have been proposed [7, 10, 20, 38, 51].

We believe that our approach provides a flexible hybrid mechanism to combine these attack- and testing-based under-approximation approaches into over-approximation-based verification approaches for neural network robustness verification.

7 CONCLUSION

We have proposed a dual-approximation approach to define tight over-approximations for the robustness verification of DNNs. Underlying this approach is our finding of *approximation domain* of the activation function that is crucial in defining tight over-approximations, yet overlooked by all existing approximation approaches. Accordingly, we have devised two complementary under-approximation algorithms to compute underestimated domains. Our experimental results have demonstrated the outperformance of our approach over the state of the art.

Our dual-approximation approach could be integrated into other abstraction-based neural network verification approaches [8, 37, 54] as they require non-linear activation functions that shall be over-approximated to handle abstract domains. In addition to the robustness verification, we believe that our approach is also applicable to the variants of robustness verification problems, such as fairness [2] and ϵ-weekend robustness [14]. Verifying those properties can be reduced to optimization problems that contain the nonlinear activation functions in networks.

REFERENCES

[1] Apollo. 2017. ApolloAuto. https://github.com/ApolloAuto/apollo. Accessed: 2022-05-06.

[2] Osbert Bastani, Xin Zhang, and Armando Solar-Lezama. 2019. Probabilistic verification of fairness properties via concentration. *Proc. ACM Program. Lang.* 3 (2019), 118:1–118:27.

[3] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. 2016. End to End Learning for Self-Driving Cars. *CoRR* abs/1604.07316 (2016).

[4] Nicholas Carlini and David A. Wagner. 2017. Towards Evaluating the Robustness of Neural Networks. In *IEEE Symposium on Security and Privacy (S&P)*. 39–57.

[5] Anirban Chakraborty, Manaar Alam, Vishal Dey, Anupam Chattopadhyay, and Debdeep Mukhopadhyay. 2018. Adversarial Attacks and Defences: A Survey. *CoRR* abs/1810.00069 (2018).

[6] Yizhak Yisrael Elboher, Justin Gottschlich, and Guy Katz. 2020. An abstraction-based framework for neural network verification. In *CAV'20*. Springer, 43–65.

[7] Xinyu Gao, Yang Feng, Yining Yin, Zixi Liu, Zhenyu Chen, and Baowen Xu. 2022. Adaptive Test Selection for Deep Neural Networks. In *ICSE'22*. ACM, 73–85.

[8] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. 2018. AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation. In *S&P'18*. IEEE, 3–18.

[9] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. In *ICLR'15*.

[10] Jianmin Guo, Yu Jiang, Yue Zhao, Quan Chen, and Jiaguang Sun. 2018. DLFuzz: differential fuzzing testing of deep learning systems. In *ESEC/FSE'18*. 739–743.

[11] Yingzhe He, Guozhu Meng, Kai Chen, Xingbo Hu, and Jinwen He. 2022. Towards Security Threats of Deep Learning Systems: A Survey. *IEEE Trans. Software Eng.* 48 (2022), 1743–1770.

[12] Patrick Henriksen and Alessio Lomuscio. 2020. Efficient Neural Network Verification via Adaptive Refinement and Adversarial Search. In *ECAI'20*. 2513–2520.

[13] Po-Sen Huang, Robert Stanforth, Johannes Welbl, Chris Dyer, Dani Yogatama, Sven Gowal, Krishnamurthy Dvijotham, and Pushmeet Kohli. 2019. Achieving Verified Robustness to Symbol Substitutions via Interval Bound Propagation. In *EMNLP-IJCNLP'19*. 4081–4091.

[14] Pei Huang, Yuting Yang, Minghao Liu, Fuqi Jia, Feifei Ma, and Jian Zhang. 2022. ϵ-weakened robustness of deep neural networks. In *International Symposium on Software Testing and Analysis (ISSTA)*. 126–138.

[15] Xiaowei Huang, Daniel Kroening, Wenjie Ruan, James Sharp, Youcheng Sun, Emese Thamo, Min Wu, and Xinping Yi. 2020. A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability. *Comput. Sci. Rev.* 37 (2020), 100270.

[16] Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Madry. 2019. Adversarial Examples Are Not Bugs, They Are Features. In *NeurIPS'19*. 125–136.

[17] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. 2017. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In *CAV'17*. Springer, 97–117.

[18] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning Multiple Layers of Features from Tiny Images. (2009).

[19] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86 (1998), 2278–2324.

[20] Seokhyun Lee, Sooyoung Cha, Dain Lee, and Hakjoo Oh. 2020. Effective white-box testing of deep neural networks with adaptive neuron-selection strategy. In *ISSTA'20*. 165–176.

[21] Sungyoon Lee, Jaewook Lee, and Saerom Park. 2020. Lipschitz-Certifiable Training with a Tight Outer Bound. In *NeurIPS'20*. 16891–16902.

[22] Renjue Li, Pengfei Yang, Cheng-Chao Huang, Youcheng Sun, Bai Xue, and Lijun Zhang. 2022. Towards Practical Robustness Analysis for DNNs based on PAC-Model Learning. In *ICSE'22*. 2189–2201.

[23] Wang Lin, Zhengfeng Yang, Xin Chen, Qingye Zhao, Xiangkun Li, Zhiming Liu, and Jifeng He. 2019. Robustness Verification of Classification Deep Neural Networks via Linear Programming. In *CVPR'19*. 11418–11427.

[24] Changliu Liu, Tomer Arnon, Christopher Lazarus, Christopher A. Strong, Clark W. Barrett, and Mykel J. Kochenderfer. 2021. Algorithms for Verifying Deep Neural Networks. *Found. Trends Optim.* 4 (2021), 244–404.

[25] Zixi Liu, Yang Feng, Yining Yin, and Zhenyu Chen. 2022. DeepState: Selecting Test Suites to Enhance the Robustness of Recurrent Neural Networks. In *ICSE'22*. 598–609.

[26] Zhaoyang Lyu, Ching-Yun Ko, Zhifeng Kong, Ngai Wong, Dahua Lin, and Luca Daniel. 2020. Fastened CROWN: Tightened Neural Network Robustness Certificates. In *AAAI'20*. 5037–5044.

[27] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. 2016. DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks. In *CVPR'16*. 2574–2582.

[28] Mark Müller, Gleb Makarchuk, Gagandeep Singh, Markus Püschel, and Martin T Vechev. 2022. PRIMA: general and precise neural network certification via scalable convex hull approximations. *Proc. ACM Program. Lang.* 6 (2022), 1–33.

[29] Mark Niklas Müller, Christopher Brix, Stanley Bak, Changliu Liu, and Taylor T Johnson. 2022. The Third International Verification of Neural Networks Competition (VNN-COMP 2022): Summary and Results. *arXiv preprint arXiv:2212.10376* (2022).

[30] Rangeet Pan. 2020. Does fixing bug increase robustness in deep learning?. In *ICSE'20 (Companion Volume)*. 146–148.

[31] Nicolas Papernot, Patrick D. McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. 2016. The Limitations of Deep Learning in Adversarial Settings. In *EuroS&P*. 372–387.

[32] Brandon Paulsen and Chao Wang. 2022. Example Guided Synthesis of Linear Approximations for Neural Network Verification. In *CAV'22*. Springer, 149–170.

[33] Brandon Paulsen and Chao Wang. 2022. LinSyn: Synthesizing Tight Linear Bounds for Arbitrary Neural Network Activation Functions. In *TACAS'22*. Springer, 357–376.

[34] Luca Pulina and Armando Tacchella. 2010. An Abstraction-Refinement Approach to Verification of Artificial Neural Networks. In *CAV'10*. Springer, 243–257.

[35] Sebastian Ruder. 2016. An overview of gradient descent optimization algorithms. *CoRR* abs/1609.04747 (2016).

[36] Hadi Salman, Greg Yang, Huan Zhang, Cho-Jui Hsieh, and Pengchuan Zhang. 2019. A Convex Relaxation Barrier to Tight Robustness Verification of Neural Networks. In *NeurIPS'19*. 9832–9842.

[37] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin T. Vechev. 2019. An abstract domain for certifying neural networks. *Proc. ACM Program. Lang.* 3, POPL (2019), 41:1–41:30.

[38] Youcheng Sun, Xiaowei Huang, Daniel Kroening, James Sharp, Matthew Hill, and Rob Ashmore. 2019. Structural test coverage criteria for deep neural networks. In *ICSE'19*. 320–321.

[39] Yi Sun, Ding Liang, Xiaogang Wang, and Xiaoou Tang. 2015. DeepID3: Face Recognition with Very Deep Neural Networks. *CoRR* abs/1502.00873 (2015).

[40] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. In *ICLR'14*.

[41] Joseph J Titano, Marcus Badgeley, Javin Schefflein, Margaret Pain, Andres Su, Michael Cai, Nathaniel Swinburne, John Zech, Jun Kim, Joshua Bederson, et al. 2018. Automated deep-neural-network surveillance of cranial images for acute neurologic events. *Nat. Med.* 24, 9 (2018), 1337–1341.

[42] Christian Tjandraatmadja, Ross Anderson, Joey Huchette, Will Ma, Krunal Patel, and Juan Pablo Vielma. 2020. The Convex Relaxation Barrier, Revisited: Tightened Single-Neuron Relaxations for Neural Network Verification. In *NeurIPS'20*. 21675–21686.

[43] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. 2018. Formal Security Analysis of Neural Networks using Symbolic Intervals. In *USENIX Security'18*. 1599–1614.

[44] Tai Wang, Xinge Zhu, Jiangmiao Pang, and Dahua Lin. 2021. Probabilistic and Geometric Depth: Detecting Objects in Perspective. In *ICRL'21*, Vol. 164. PMLR, 1475–1485.

[45] Zi Wang, Aws Albarghouthi, Gautam Prakriya, and Somesh Jha. 2022. Interval universal approximation for neural networks. *Proc. ACM Program. Lang.* 6, POPL (2022), 1–29.

[46] Jeannette M. Wing. 2021. Trustworthy AI. *Commun. ACM* 64 (2021), 64–71.

[47] Eric Wong and J. Zico Kolter. 2018. Provable Defenses against Adversarial Examples via the Convex Outer Adversarial Polytope. In *ICML'18*. 5283–5292.

[48] Min Wu and Marta Kwiatkowska. 2020. Robustness Guarantees for Deep Neural Networks on Videos. In *CVPR'20*. 308–317.

[49] Yiting Wu and Min Zhang. 2021. Tightening Robustness Verification of Convolutional Neural Networks with Fine-Grained Linear Approximation. In *AAAI'21*. 11674–11681.

[50] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *CoRR* abs/1708.07747 (2017).

[51] Jing Yu, Shukai Duan, and Xiaojun Ye. 2022. A White-Box Testing for Deep Neural Networks Based on Neuron Coverage. *IEEE Trans. Neural Netw. and Learn. Syst.* (2022), 1–13.

[52] Huan Zhang, Shiqi Wang, Kaidi Xu, Linyi Li, Bo Li, Suman Jana, Cho-Jui Hsieh, and J. Zico Kolter. 2022. General Cutting Planes for Bound-Propagation-Based Neural Network Verification. *CoRR* abs/2208.05740 (2022). https://doi.org/10.48550/arXiv.2208.05740 arXiv:2208.05740

[53] Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. 2018. Efficient Neural Network Robustness Certification with General Activation Functions. In *NeurIPS'18*. 4944–4953.

[54] Yuhao Zhang, Luyao Ren, Liqian Chen, Yingfei Xiong, Shing-Chi Cheung, and Tao Xie. 2020. Detecting numerical bugs in neural network architectures. In *ESEC/FSE'20*. 826–837.

[55] Zhaodi Zhang, Yiting Wu, Si Liu, Jing Liu, and Min Zhang. 2022. Provably Tightest Linear Approximation for Robustness Verification of Sigmoid-like Neural Networks. In *ASE'22*. ACM, 80:1–80:13.