Provably Tightest Linear Approximation for Robustness Verification of Sigmoid-like Neural Networks

Anonymous Author(s)

ABSTRACT

The robustness of deep neural networks is crucial to modern AI-enabled systems. Formal verification has been demonstrated effective in providing certified robustness guarantees. Sigmoid-like neural networks have been adopted in a wide range of applications. Due to their non-linearity, Sigmoid-like activation functions are usually *over-approximated* for efficient verification, which inevitably introduces imprecision. Considerable efforts have been devoted to finding the so-called *tighter* approximations to obtain more precise verification results. However, existing tightness definitions are heuristic and lack theoretical foundations.

We conduct a thorough empirical analysis of existing *neuron-wise* characterizations of tightness and reveal that they are superior only on specific neural networks. We then introduce the notion of *network-wise tightness* as a unified tightness definition and show that computing network-wise tightness is a complex non-convex optimization problem. We bypass the complexity from different perspectives via two efficient, provably tightest approximations. The results demonstrate the promising performance achievement of our approaches over state of the art: (i) achieving up to 436.36% improvement to certified lower robustness bounds; and (ii) exhibiting notably more precise verification results on convolutional networks.

1 INTRODUCTION

The reliability concerns about deep neural networks (DNNs) are increasing more drastically than ever, especially as such networks are being embedded into software systems to make them intelligent. Considerable efforts from both AI and software engineering communities have been devoted to achieving *robust* DNNs by leveraging testing and verification techniques [4, 12, 32, 40, 42, 47, 48, 54]. Among these attempts, formal methods have been demonstrated effective in offering certified robustness guarantees, giving birth to an emerging research field called *Trustworthy AI* [50]. One distinguishing feature of formal methods is that they could provide rigorous proofs of correctness automatically when the properties are satisfied or disprove them by counterexamples (i.e., witnesses to the violations) [3, 9]. Robustness is the most noteworthy, well-defined correctness property in DNN verification: Minor modifications to the neural network's inputs must *not* alter its outputs [7]. Guaranteeing robustness is indispensable to prevent AI-enabled systems from environmental perturbations and adversarial attacks.

Formal robustness verification of DNNs has been well studied in recent years [14, 16, 20, 32, 33, 42, 45, 47–49]. Most efforts are focused on the *ReLU networks* that only use the simple piece-wise ReLU activation function. Despite their wide adoptions in modern AI-enabled systems, another notable class of S-shaped (or Sigmoid-like) activation functions have not attracted much attention yet. Due to their non-linearity, Sigmoid-like activation functions are far more complex to be verified. A *de facto* solution is to over-approximate such functions by linear bounds and to transform the verification problem into efficiently solvable linear programming. Many state-of-the-art DNN verification techniques, e.g., abstract interpretation [16, 40], symbolic interval propagation [45], model checking [33], differential verification [32], reachability and output range analysis [13, 44], are based on linear approximation.

Over-approximation inevitably introduces imprecision, rendering approximation-based verification incomplete: *Unknown* results may be returned when the neural network's robustness cannot be verified. Considerable efforts have been devoted to finding the so-called *tighter* approximations to achieve more precise verification results. For example, a larger certified lower robust bound [5, 28] (the perturbation distance under which a neural network is proved robust against any allowable perturbation) is preferable in approximation. Several characterizations of tightness and approximation approaches have been proposed for Sigmoid-like activation functions [5, 18, 26, 28, 51, 55]. However, they are all heuristic and lack theoretical foundations for the individual outperformance.

We conduct a thorough empirical analysis of existing approaches and reveal that they are superior only on specific neural networks. In particular, we have found that the claimed tighter approximation actually produces smaller certified lower bounds according to the tightness defined and observed frequent occurrences of such cases.

Motivated by these observations, we introduce the notion of *network-wise tightness* as a *unified* tightness definition to characterize linear approximations of Sigmoid-like activation functions. This new definition ensures that a tighter approximation can constantly compute larger certified lower bounds. However, we show that it unfortunately implies that computing the tightest approximation is essentially a network-wise non-convex optimization problem [28], which is hard to solve in practice [31].

We bypass the complex optimization problem from two different perspectives, depending on the neural network architecture. For the networks *with only one hidden layer*, we leverage a gradient-based searching algorithm for computing the tightest approximations. Regarding the networks *with multiple hidden layers*, based on our empirically study of the state-of-the-art tools, we have gained an insight that *a larger robust bound can be computed when the intervals keep to be tighter during the layer-by-layer propagation*. Based on this insight, we propose a *neuron-wise* tightest approximation and prove that it guarantees the *network-wise tightest approximation* when the networks are of non-negative weights. Such networks have been demonstrated suitable in a wide range of applications such as effective defense for adversarial attacks in malware and spam detection [8, 15, 19] and balancing accuracy and robustness in autoencoding [1, 29].

We have implemented a prototype of our approach called NeWise and extensively compared it to three state-of-the-art tools, namely DeepCert [51], VeriNet [18], and RobustVerifier [26]. Our experimental results show that NeWise (i) achieves up to 436.36% improvement to certified lower robustness bounds in the provably

tightest cases and (ii) exhibits 122.22% improvement to the precision of verification results on convolutional networks.

To summarize, this paper makes three major contributions:

(1) We have introduced a novel unified definition of *network-wise tightness* to characterize the tightness of linear approximations for neural network robustness verification.
(2) We have identified two cases where we can efficiently achieve provably tightest approximations; the corresponding approaches have been proposed.
(3) We have implemented a verification tool and conducted comprehensive evaluation on its effectiveness and efficiency over three state-of-the-art verifiers.

The remainder of this paper proceeds as follows: Section 2 gives preliminaries on robustness verification of neural networks. Section 3 shows the tightness measurements of linear approximations and introduce our notion of network-wise tightness. Sections 4 and 5 present our provably tightest approximations from two different perspectives, respectively. Section 6 describes our evaluation results. We discuss related work in Section 7 and conclude in Section 8.

2 PRELIMINARIES

2.1 Robustness Verification of Neural Networks

2.1.1 Deep Neural Network. A deep neural network is a directed network, where the nodes are called neurons and arranged layer by layer. Each neuron is associated with an activation function $\sigma(x)$ and a bias b. Except for the first layer, the neurons on a layer are connected to those on the preceding layer, as shown in Figure 1. Every edge is associated with a weight, which is computed by training. The first and last layers are called input and output layers, respectively. The others between them are called hidden layers.

The *execution* of a neural network follows the style of layer-by-layer propagation. Each neuron on the input layer admits a number. The number is multiplied by the weights on the edges and then passed to the successor neurons on the next layer. All the incoming numbers are summed. The summation is fed to the activation function σ and the output of σ is added with the bias b. The result is then propagated to the next layer until reaching the output layer.

Formally, a k-layer neural network is a function $f : \mathbb{R}^n \to \mathbb{R}^m$ of the form $f^k \circ \sigma^{k-1} \circ \ldots \circ \sigma^1 \circ f^1$, with σ^t a non-linear and differentiable activation function for t-th layer. The function f^t is either an affine transformation *or* a convolutional operation:

$$f(x) = Wx + b, \qquad \text{(Affine Transformation)}$$
$$f(x) = W * x + b, \qquad \text{(Convolutional Operation)}$$

where W, b, and $*$ refer to the weight matrix, the bias vector, and the convolutional product, respectively. In this work, we focus on the networks with the Sigmoid-like activation functions i.e., Sigmoid, Tanh, and Arctan, which are defined as follows, respectively.

$$\sigma(x) = \frac{1}{1+e^{-x}}, \qquad \sigma(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \qquad \sigma(x) = tan^{-1}(x)$$

The output of a neural network f is a vector of m floating numbers between 0 and 1, denoting the probabilities of classifying an input to the m labels. Let S be the set of m classification labels for the network f. We use $\mathcal{L}(f(x))$ to represent the output label for the input x with

$$\mathcal{L}(f(x)) = \arg\max_{s \in S} f(x)[s].$$

Intuitively, $\mathcal{L}(f(x))$ returns a label s in S such that $f(x)[s]$ is maximal among the numbers in the output vector.

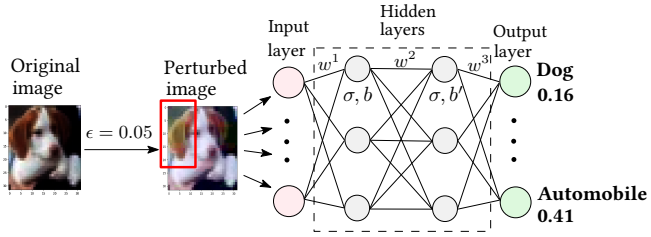


Figure 1: A perturbed image of a dog is misclassified to an automobile with 41% probability in 0.05 perturbation radius.

2.1.2 Robustness and Robustness Verification. Neural networks are essentially "programs" composed by computers by fine-tuning the weights in the networks from training data. Unlike the handcrafted programs developed by programmers, neural networks lack formal requirements and are almost inexplicable, making it very challenging to formalize and verify their properties.

A neural network is called *robust* if reasonable perturbations to its inputs do not alter the classification result. A perturbation is typically measured by the distance between the perturbed input x' and the original one x by using ℓ_p-norm, denoted by $||x - x'||_p \triangleq \sqrt[q]{|x_1 - x'_1|^p + \ldots + |x_n - x'_n|^p}$, where p can be 1, 2 or $+\infty$, and n is the length of the vectors x, x'. In this work, we consider the most general case when $p = \infty$.

Example 1. We consider an example to explain how a perturbed image is misclassified. As shown in Figure 1, a normal image of a dog can be correctly classified by a neural network. We assume that the image can be perturbed within a 0.05 distance under ℓ_∞-norm. There exists a perturbed image such that when it is fed into the network, the outputs of the two neurons labeled by *dog* and *automobile* are 0.16 and 0.41, respectively. It indicates that the image is classified to a dog (resp. automobile) with the probability of 16% (reps. 41%). Therefore, it is classified to be an automobile, although it still represents a dog to human eyes, apparently. The robustness of a neural network can be quantitively measured by a lower bound ϵ, which refers to a safe perturbation distance such that any perturbations below ϵ have the same classification result as the original input to the neural network.

Definition 1 (Robustness). *Given a neural network f, an input x_0, and a bound ϵ under ℓ_p-norm, f is called robust w.r.t. x_0 iff $\mathcal{L}(f(x)) = \mathcal{L}(f(x_0))$ holds for each x such that $||x - x_0||_p \leq \epsilon$. Such ϵ is called a certified lower bound.*

The twin problems of verifying f's robustness are: (i) to prove that, for each x satisfying $||x - x_0||_p \leq \epsilon$,

$$f_{s_0}(x) - f_s(x) > 0 \qquad (1)$$

holds for each $s \in S - \{s_0\}$, where $s_0 = \mathcal{L}(f(x_0))$ and $f_s(x)$ returns the probability of classifying x to the label s by f; and (ii) to compute a certified lower bound — a larger certified lower bound implies a

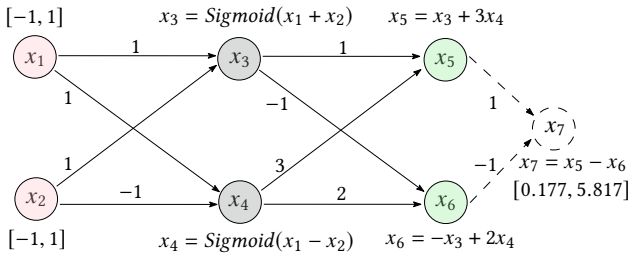


Figure 2: An example of approximation-based verification.

more precise robustness verification result. As directly computing ϵ is difficult due to the non-linearity of the constraint (1), most of the state-of-the-art approaches [5, 51, 55] adopt the efficient binary search algorithm to first determine a candidate ϵ and then check whether (1) is true or false on ϵ.

2.2 Approximation-based Robustness Verification

A neural network f is highly non-linear due to the inclusion of activation functions. Proving Formula (1) is computationally expensive, e.g., NP-complete even for the simplest fully connected ReLU networks [20, 37]. Many approaches have been investigated to improve the verification efficiency while sacrificing completeness. Representative methods include interval analysis [46], abstract interpretation [16, 40], and output range estimation [13, 52], etc. The technique underlying these approaches is to over-approximate the non-linear activation functions using linear constraints, which can be more efficiently solved than the original ones.

Instead of directly proving Formula (1), the approximation-based approaches over-approximate both $f_{s_0}(x)$ and $f_s(x)$ by two linear constraints and prove that the lower linear bound $f_{L,s_0}(x)$ of $f_{s_0}(x)$ is greater than the upper linear bound $f_{U,s}(x)$ of $f_s(x)$. Apparently, $f_{L,s_0}(x) - f_{U,s}(x) > 0$ is a sufficient condition of Formula (1), and it is significantly more efficient to prove or disprove than Formula (1). Therefore, it is widely adopted in neural network verification [5, 18, 26, 51], although it may produce false positives when it is disproved.

Definition 2 (Upper/Lower linear bounds). *Let $\sigma(x)$ be a non-linear function with $x \in [l, u]$, $\alpha_L, \alpha_U, \beta_L, \beta_U \in \mathbb{R}$, and*

$$h_U(x) = \alpha_U x + \beta_U, \qquad h_L(x) = \alpha_L x + \beta_L. \tag{2}$$

$h_U(x)$ and $h_L(x)$ are called upper and lower linear bounds of $\sigma(x)$ if the following condition holds:

$$\forall x \in [l, u], \quad h_L(x) \le \sigma(x) \le h_U(x). \tag{3}$$

Over-approximating the non-linear activation functions using linear lower and upper bounds is the key to the approximation of a neural network. For each activation function σ on a domain $[l, u]$, we define an upper linear bound h_U and a lower one h_L to ensure that for all x in $[l, u]$, $\sigma(x)$ is enclosed in $[h_L(x), h_U(x)]$.

Given an input range as in Definition 1, the output ranges of a network are computed by propagating the output interval of each neuron as in Definition 2 to the output layer.

Example 2. We consider an example of verifying a simple neural network based on approximation, as shown in Figure 2. The original verification problem is to prove that for any input (x_1, x_2) with $x_1 \in [-1, 1]$ and $x_2 \in [-1, 1]$, it is always classified to the label of neuron x_5. That is equivalent to prove the output of the auxiliary neuron $x_7 = (x_5 - x_6)$ is always greater than 0. We define the linear upper/lower bounds $x_{3,U}, x_{3,L}$ and $x_{4,U}, x_{4,L}$ to over-approximate x_3 and x_4, respectively. It suffices to prove that $x_{5,L} - x_{6,U} > 0$ is always true. We can over-estimate the output interval of $x_{5,L} - x_{6,U}$ is $[0.177, 5.817]$ using $x_{3,U}, x_{3,L}$ and $x_{4,U}, x_{4,L}$ and consequently prove the robustness of the network for all the inputs in $[-1, 1]^2$.

Note that it is not necessary to approximate an activation function using only one linear upper or lower bound. One may consider piece-wise linear bounds to approximate the function more tightly by being closer to it. However, the piece-wise way causes the number of constraints to blow up exponentially when propagated layer by layer [40]. It would drastically reduce the verification scalability. Thus, over-approximating an activation function using one upper linear bound and one lower linear bound is the most efficient and widely-adopted choice for the approximation-based robustness verification approaches.

3 LINEAR APPROXIMATION APPROACHES

In this section, we analyze the tightness issue of existing approximation approaches and formally define a unified network-wise tightness to characterize approximations.

3.1 The Tightness Issue of Approximations

As approximation inevitably introduces overestimation, defining the *tightest* possible approximation is crucial to obtaining precise verification results. Several approximation approaches have been proposed under different strategies.

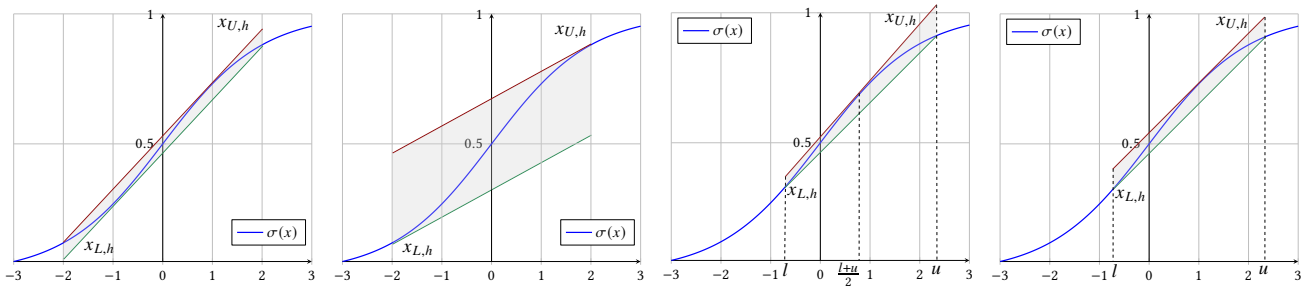
Henriksen et al. [18] proposed to measure the tightness of approximations using the enclosed area between the bound and the approximated function. An approximation is tighter if the corresponding area is smaller than another. By this definition, the approximations to x_3, x_4 should be the following linear bounds:

$$x'_{3,U} = x'_{4,U} = 0.204(x_1 + x_2) + 0.527, \tag{4}$$

$$x'_{3,L} = x'_{4,L} = 0.204(x_1 + x_2) + 0.472. \tag{5}$$

Figure 3a shows the bounds graphically. Apparently, they are closer to the activation function on the interval $[-2, 2]$. Surprisingly, using the *tighter* linear bounds the output range of x_7 is $[-0.079, 6.073]$, by which we cannot prove and disprove the robustness. Wu and Zhang adopted the same strategy for this case in their recent work [51]. In Example 2, we adopt a new strategy by taking the tangent lines at the two endpoints as its upper and lower bounds, as shown in Figure 3b. We obtain a preciser output range by these bounds, although they are far less tight than the one in Figure 3a.

In another case shown in Figure 3c, Henriksen et al. proved that the tangent line at the middle point when $x = \frac{l+u}{2}$ is the tight upper bound because the enclosed area between it and the activation function is minimal. In Wu and Zhang's approach, they adopted the tangent line that is parallel to the lower bound as its upper bound, as shown in Figure 3d. Some other approaches such



(a) Minimal area and parallel lines. **(b) Endpoints (our work).** **(c) Minimal area [18].** **(d) Parallel lines [51].**

Figure 3: Different approximations according to the domain of $\sigma(x)$ and their tightness definitions.

as [5, 26, 55] adopt similar approximation strategies, but they have been experimentally proved not as tight as the ones in [18, 51].

Lyu et al. [28] proposed a gradient-based searching approach for computing a tighter approximation if the approximation can produce tighter input intervals for the following neurons. However, the experimental results in the work [51] show that this approach neither guarantees it always produces larger certified lower robust bound than other approaches and its scalability is rather limited due to the complexity of the searching algorithm for each neuron.

3.2 Empirical Analysis

To validate our observation on the tightness issue and investigate its generality, we have performed empirical analysis of three state-of-the-art approximation approaches, i.e., DeepCert [51], VeriNet [18], and RobustVerifier [26], on 20 sigmoid neural networks collected from the public benchmarks. We evaluate the tightness of these approaches by computing the lower robust bound for each network using the tools, respectively. We randomly selected 100 images for each network, computed their lower robust bounds, and took the average value. Computing averaged lower bounds is a widely-adopted approach to reduce the affect of floating-point errors [28, 47, 51]. Thus, even a small difference in the averaged value reflects a big difference in individual input. In general, the larger bound indicates the corresponding tool has a better performance.

Table 1 shows the comparison results, where, surprisingly, none of these approaches surpass the others for all the networks. We also observe that VeriNet won the competition on 13 out of 20 networks, while DeepCert on the remaining ones. This indicates that the performances of these so-called tight approaches vary case by case. In this paper we do not indent to judge which approach is better experimentally but focus on seeking theoretical foundations for the tightness of approximations. The comparison result showed that existing tightness definitions do not rigorously guarantee that a tighter approximation can always produce a larger robust bound. This motivates us to seek a unified definition of *tightness* of approximations for robustness verification of neural networks.

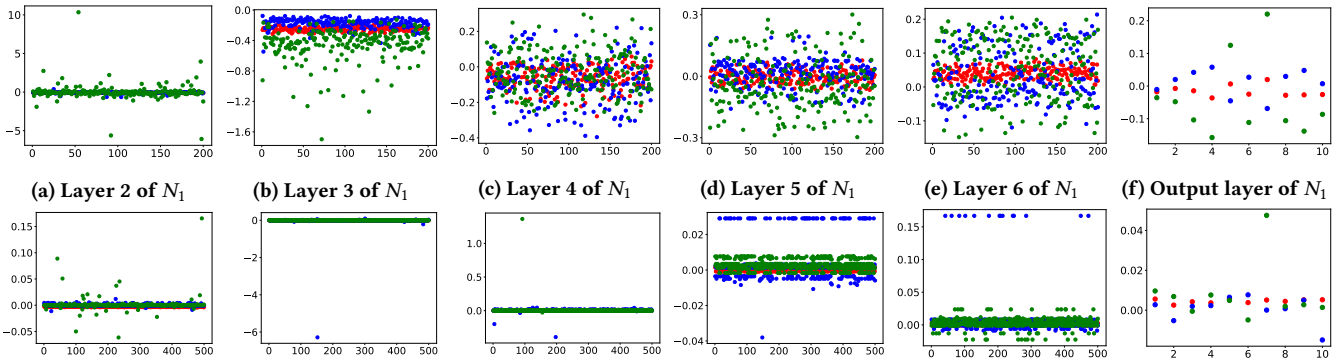
We empirically analyzed the computation process of the three tools and identified the missing factor that influences verification results. We tracked the computation of the intermediate intervals of the neurons on hidden layers during their layer-by-layer propagation and compared their tightness under different approximations. Figure 4 shows the layer-by-layer comparison of the intermediate intervals on the hidden neurons and output neurons. These intervals are computed during verification using the approximation

Table 1: Tightness evaluation of state of the art: DeepCert[51], VeriNet[28], and RobustVerifier[26]. CNN_{t-c} denotes a CNN with t layers and c filters of size 3×3. The models are pre-trained [2, 40, 51] by, e.g., DeepPoly [40].

Dataset	Model	#Neurons	Certified Lower Bound (Average)		
			DeepCert	VeriNet	Rob.Ver.
MNIST	3x50	160	0.0076	0.0077	0.0065
	3x100	310	0.0086	0.0087	0.0074
	3x200	610	0.0091	0.0092	0.0079
	5x100	510	0.0061	0.0062	0.0052
	6x500	3,010	0.0778	0.0776	0.0665
	CNN_{3-2}	2,514	0.0579	0.0580	0.0569
	CNN_{3-4}	5,018	0.0473	0.0472	0.0464
	CNN_{4-5}	8,680	0.0539	0.0543	0.0522
	CNN_{5-5}	10,680	0.0548	0.0550	0.0513
	CNN_{6-5}	12,300	0.0590	0.0588	0.0541
	CNN_{8-5}	14,570	0.0878	0.0882	0.0685
Fashion MNIST	3x50	160	0.0101	0.0102	0.0086
	5x100	510	0.0078	0.0079	0.0066
	CNN_{4-5}	8,680	0.0721	0.0720	0.0666
	CNN_{5-5}	10,680	0.0676	0.0677	0.0605
	CNN_{6-5}	12,300	0.0695	0.0691	0.0627
CIFAR10	3x50	160	0.0045	0.0046	0.0042
	5x100	510	0.0038	0.0037	0.0033
	CNN_{3-2}	3,378	0.0312	0.0313	0.0311
	CNN_{6-5}	17,110	0.0224	0.0223	0.0213

approaches in [18, 51], respectively. The figures from 4a to 4f show one 5-hidden-layer network named N_1 on which VeriNet computes a larger bound than DeepCert, while those from 4g to 4l show another 5-hidden-layer network named N_2 on which DeepCert computes a larger bound than VeriNet. For each neuron, we use $[l, u]$ and $[l', u']$ to represent the intervals computed by VeriNet and DeepCert, respectively. We introduce three dots in red, blue and green to represent $\frac{(u-l)-(u'-l')}{u'-l'}$, $\frac{l-l'}{l'}$ and $\frac{u-u'}{u'}$, respectively. The x-axis represents the neurons on the corresponding layer, and the y-axis represents the differences of the interval length, lower bounds, and upper bounds of the intervals, respectively.

Horizontally on N_1, most of the red dots are below 0 from layer 2 to the output layer, except layer 6. That indicates the intervals computed by VeriNet usually have smaller (tighter) sizes than those by DeepCert. The blue dots gradually move up above 0, indicating that the lower bounds of the intervals computed by VeriNet become greater than the those by DeepCert. Similarly, the green dots gradually move down below 0, indicating that the upper bounds computed by VeriNet become smaller. The trends of the three values reflect that the intermediate intervals computed by VeriNet are statistically tighter than those by DeepCert on network N_1.



(a) Layer 2 of N_1 (b) Layer 3 of N_1 (c) Layer 4 of N_1 (d) Layer 5 of N_1 (e) Layer 6 of N_1 (f) Output layer of N_1

(g) Layer 2 of N_2 (h) Layer 3 of N_2 (i) Layer 4 of N_2 (j) Layer 5 of N_2 (k) Layer 6 of N_2 (l) Output layer of N_2

Figure 4: Visualization of intermediate intervals during layer-by-layer propagation under different approximations (Red dot: $\frac{(u-l)-(u'-l')}{u'-l'}$; blue dot: $\frac{l-l'}{l'}$; green dot: $\frac{u-u'}{u'}$; $[l,u]$: interval computed by VeriNet, $[l',u']$: interval computed by DeepCert).

The trend of intermediate intervals on network N_2 is an opposite of the one on N_1. The red dots move up above 0 layer by layer, indicating that the interval sizes computed by VeriNet become larger than those by DeepCert. The green dots gradually move down below 0 and the blue ones move up above 0.

The analysis result from Figure 4 reveals that the intermediate intervals are statistically tighter than other tools when a tool produces larger certified lower bounds.

3.3 The Network-Wise Tightest Approximation

Existing characterizations of tightness are individually heuristic under a presumption that a tighter bound gives rise to a more precise robustness verification result. Unfortunately, the above examples show that this presumption does not always hold. We introduce the notion of *network-wise tightness* to characterize the tightness of approximations to the activation functions in a neural network and ensure that a tighter approximation must produce a preciser verification result.

Definition 3 (Network-wise tightness). *Given a neural network $f : \mathbb{R}^n \to \mathbb{R}^m$ and $x \in \mathbb{B}_p(x_0, \epsilon)$, let (f_L, f_U) be a linear approximation of f with f_U and f_L the upper and lower bounds, respectively. (f_L, f_U) is network-wise tightest if, for any different linear approximation $(\hat{f}_L, \hat{f}_U)$,*

$$\forall s \in S, \quad \min_{x\in\mathbb{B}_p(x_0,\epsilon)} f_{L,s}(x) \geq \min_{x\in\mathbb{B}_p(x_0,\epsilon)} \hat{f}_{L,s}(x),$$
$$\max_{x\in\mathbb{B}_p(x_0,\epsilon)} f_{U,s}(x) \leq \max_{x\in\mathbb{B}_p(x_0,\epsilon)} \hat{f}_{U,s}(x),$$

where $f_{L,s}(x), f_{U,s}(x)$ denote s-th item of $f_L(x), f_U(x)$, respectively.

Intuitively, (f_L, f_U) is tighter than $(\hat{f}_L, \hat{f}_U)$ in that for all output neurons s, f_L (*resp.* f_U) always computes a lower (*resp.* an upper) bound that is greater (*resp.* less) than the one $\hat{f}_L$ (*resp.* $\hat{f}_U$) does. Note that Definition 3 is universal in that it is applicable to (i) all activation functions and (ii) all ℓ_p norms.

Example 3. By Definition 3, the approximations $x_{3,U}, x_{3,L}, x_{4,U}, x_{4,L}$ to the activation functions on x_3 and x_4 in Example 2 are tighter than $x'_{3,U}, x'_{3,L}, x'_{4,U}, x'_{4,L}$, which is consistent to the verification results. Using the former approximations, we can compute tighter output ranges for both x_5 and x_6 than those by the latter, although the former are less tight than the latter according to the tightness definition with respect to minimal area [18].

Theorem 1. *The approximation (f_L, f_U) of a neural network always produces more precise robustness verification result than $(\hat{f}_L, \hat{f}_U)$ if (f_L, f_U) is tighter than $(\hat{f}_L, \hat{f}_U)$ by Definition 3.*

Proof sketch. To verify the robustness of network at input x_0, let $s_0 = \mathcal{L}(f(x_0))$, with fixed ϵ, we check whether

$$\min_{x\in\mathbb{B}_p(x_0,\epsilon)} f_{L,s_0}(x) > \max_{x\in\mathbb{B}_p(x_0,\epsilon)} f_{U,s}(x) \tag{6}$$

for all s other than s_0. By Definition 3, if $(\hat{f}_L, \hat{f}_U)$ satisfies (6), then we have:

$$\min_{x\in\mathbb{B}_p(x_0,\epsilon)} f_{L,s_0}(x) \geq \min_{x\in\mathbb{B}_p(x_0,\epsilon)} \hat{f}_{L,s_0}(x) >$$
$$\max_{x\in\mathbb{B}_p(x_0,\epsilon)} \hat{f}_{U,s}(x) \geq \max_{x\in\mathbb{B}_p(x_0,\epsilon)} f_{U,s}(x),$$

for all s other than s_0. Thus, (f_L, f_U) certainly satisfies (6) as well. Namely, the result verified by $(\hat{f}_L, \hat{f}_U)$ can also be deduced by (f_L, f_U). On the contrary, the result verified by (f_L, f_U) may not be verified by $(\hat{f}_L, \hat{f}_U)$. Consequently, (f_L, f_U) always produce more precise verification result than $(\hat{f}_L, \hat{f}_U)$. □

Next, we show that computing the network-wise tightest linear approximation is essentially an optimization problem.

Given a k-layer neural network $f : \mathbb{R}^n \to \mathbb{R}^m$, we use ϕ^t to denote the compound function of f's layers before t-th activation function is applied, *i.e.*,

$$\phi^t = f^t \circ \sigma^{t-1} \circ f^{t-1} \circ \ldots \circ \sigma^1 \circ f^1.$$

For layer t with n^t neurons, let $\phi^t_r(x)$ indicate r-th item of its output (with $r \in \mathbb{Z}$ and $1 \leq r \leq n^t$). For each activation function $\sigma(x)$ with $x \in [l, u]$, we denote the upper (resp. lower) bound of $\sigma(x)$ by $h_U(x) = \alpha_U x + \beta_U$ (resp. $h_L(x) = \alpha_L x + \beta_L$), with variables $\alpha_L, \alpha_U, \beta_L, \beta_U \in \mathbb{R}$.

The problem of computing the network-wise tightest approximation can then be formalized as the following optimization problems:

$$\max(\min_{x\in\mathbb{B}_\infty(x_0,\epsilon)}(A^k_{L,s}x+B^k_{L,s})), \text{and} \tag{7}$$

$$\min(\max_{x\in\mathbb{B}_\infty(x_0,\epsilon)}(A^k_{U,s}x+B^k_{U,s})) \tag{8}$$

$$s.t.\quad \forall r\in\mathbb{Z}, 1\le r\le n^t, \forall t\in\mathbb{Z}, 1\le t<k,$$

$$\begin{cases} \alpha^t_{L,r}z^t_r+\beta^t_{L,r}\le\sigma(z^t_r)\le\alpha^t_{U,r}z^t_r+\beta^t_{U,r};\\ \min\limits_{x\in\mathbb{B}_\infty(x_0,\epsilon)} A^t_{L,r}x+B^t_{L,r}\le z^t_r\le \max\limits_{x\in\mathbb{B}_\infty(x_0,\epsilon)} A^t_{U,r}x+B^t_{U,r}. \end{cases}$$

Here, $A^t_{L,r}x+B^t_{L,r}$ and $A^t_{U,r}x+B^t_{U,r}$ are the lower and upper linear bounds of $\phi^t_r(x)$, respectively. $A^t_{L,r}, A^t_{U,r}, B^t_{L,r}$ and $B^t_{U,r}$ are constant tensors defined on W^t, b^t, where W^t, b^t are the weights and biases of the t-th layer. ϕ^t_r is the compound function of f's first t layers. $\phi^t_r(x)$ can be approximated by a lower linear bound $A^t_{L,r}x+B^t_{L,r}$ and an upper linear bound $A^t_{U,r}x+B^t_{U,r}$ with:

$$A^t_{L,r}=\begin{cases} W^t_r, & t=1\\ W^t_{\ge0,r}\alpha^{t-1}_L\odot A^{t-1}_L+W^t_{<0,r}\alpha^{t-1}_U\odot A^{t-1}_L, & t\ge2\end{cases}$$

$$B^t_{L,r}=\begin{cases} b^t_r, & t=1\\ W^t_{\ge0,r}(\alpha^{t-1}_L\odot B^{t-1}_L+\beta^{t-1}_L)+\\ W^t_{<0,r}(\alpha^{t-1}_U\odot B^{t-1}_L+\beta^{t-1}_L)+b^t_r, & t\ge2\end{cases}$$

$$A^t_{U,r}=\begin{cases} W^t_r, & t=1\\ W^t_{\ge0,r}\alpha^{t-1}_U\odot A^{t-1}_U+W^t_{<0,r}\alpha^{t-1}_L\odot A^{t-1}_U, & t\ge2\end{cases}$$

$$B^t_{U,r}=\begin{cases} b^t_r, & t=1\\ W^t_{\ge0,r}(\alpha^{t-1}_U\odot B^{t-1}_U+\beta^{t-1}_U)+\\ W^t_{<0,r}(\alpha^{t-1}_L\odot B^{t-1}_U+\beta^{t-1}_U)+b^t_r, & t\ge2\end{cases}$$

where $\odot$ denotes Hadamard production.

The solutions to all $\alpha_L, \alpha_U, \beta_L, \beta_U$ are the linear bounds to all the activation functions in the network, and their composition is the network-wise tightest approximation. Note that the solutions may not guarantee that the approximation to an individual activation function is the tightest with respect to existing tightness definitions.

4 ALGORITHM FOR 1-HIDDEN-LAYER NETWORKS

Given a neural network, we can compute the network-wise tightest approximation by instantiating and solving the optimization problems (7) and (8). For a one-hidden-layer network, the optimization problems can be simplified as follows:

$$\max(\min_{x\in\mathbb{B}_\infty(x_0,\epsilon)}(A_{L,s}x+B_{L,s})), \tag{9}$$

$$\min(\max_{x\in\mathbb{B}_\infty(x_0,\epsilon)}(A_{U,s}x+B_{U,s})), \tag{10}$$

$$s.t.\quad \forall r\in\mathbb{Z}, 1\le r\le n,$$

$$\begin{cases} \alpha_{L,r}z_r+\beta_{L,r}\le\sigma(z_r)\le\alpha_{U,r}z_r+\beta_{U,r};\\ \min\limits_{x\in\mathbb{B}_\infty(x_0,\epsilon)} W^1x+b^1\le z_r\le \max\limits_{x\in\mathbb{B}_\infty(x_0,\epsilon)} W^1x+b^1. \end{cases}$$

where $A_{L,s}=W^2_{ge0,s}(\alpha_L\odot W^1)+W^2_{<0,s}(\alpha_U\odot W^1)$, $B_{L,s}=W^2_{\ge0,s}(\alpha_L\odot b^1+\beta_L)+W^2_{<0,s}(\alpha_U\odot b^1+\beta_U)+b^2_s$, n denotes the amount of neurons in the hidden layer. The above optimization problems are the instances of the problems (7) and (8).

Algorithm 1: A gradient descent-based searching algorithm for the tightest approximations of 1-hidden-layer networks.

Input : N: a network; x_0: an input to N; ϵ: a ℓ_∞-norm radius
Output: $\alpha_{L,r}, \beta_{L,r}, \alpha_{U,r}, \beta_{U,r}$ for each hidden neuron r

1 **for** *each neuron* r **do**
2 Evaluate input range $[l_r, u_r]$ for r;
3 Let ω denote the line connecting $(l_r, \sigma(l_r))$ and $(u_r, \sigma(u_r))$;
4 $R_L \leftarrow \emptyset, R_U \leftarrow \emptyset$; *// Empty the sets of optimizable neurons.*
5 **if** *ω can be an upper bound of σ* **then**
6 Let $\alpha_{U,r}, \beta_{U,r}$ be the slope and intercept of ω;
7 Add $(r, [l_r, u_r])$ to R_L; *// r's lower bound is optimizable.*
8 **else if** *ω can be a lower bound of σ* **then**
9 Let $\alpha_{L,r}, \beta_{L,r}$ be the slope and intercept of ω;
10 Add $(r, [l_r, u_r])$ to R_U; *// r's upper bound is optimizable.*
11 **else**
12 Let $z_{U,r}, z_{L,r}$ be the cut-off points of the tangent lines of σ crossing $(l_r, \sigma(l_r))$ and $(u_r, \sigma(u_r))$;
13 Add $(r, [z_{U,r}, u_r])$ to R_U, and $(r, [l_r, z_{L,r}])$ to R_L;
14 Randomize the cut-off points for optimizable bounds of r;
15 Let $\alpha_{L,r}, \beta_{L,r}, \alpha_{U,r}, \beta_{U,r}$ be the slope and intercept of tangent line of σ at chosen cut-off points;
16 **for** $1, \ldots, k$ **do** *// k is the preset optimization round*
17 Compute $A_{L,s}, B_{L,s}$ of the lower bound of output neuron s;
18 Let $G := \min\limits_{x\in\mathbb{B}_\infty(x_0,\epsilon)}(A_{L,s}x+B_{L,s})$;
19 Update the cut-off points for r's bound through $-\nabla(G)$;
20 Update $\alpha_{L,r}, \beta_{L,r}, \alpha_{U,r}, \beta_{U,r}$ at chosen cut-off points;

The optimization problem is a convex variant and thus efficiently solvable by leveraging the gradient descent-based searching algorithm [28]. Algorithm 1 shows the pseudo code of the algorithm for calculating the optimal solution for the optimization problem with objective function (9). A solution represents a network-wise tightest lower bound to the 1-hidden-layer networks. For each activation function on a hidden neuron, it first determines whether the line ω crossing the two endpoints can be an upper (Lines 4-6) or lower bound (Lines 7-9). If those are the cases, the tangent line of the activation function is chosen to be lower bound (*resp.* upper bound), and its cut-off point can be an optimization variable. Otherwise (Lines 11-13), the lower and upper bounds can both be optimized. The optimizing ranges for those cases are calculated.

Let $G := \min\limits_{x\in\mathbb{B}_\infty(x_0,\epsilon)}(A_{L,s}x+B_{L,s})$ and $ANS = \max\limits_{\alpha_L,\alpha_U,\beta_L,\beta_U}(G)$. We use gradient descent steps (Lines 15-18) to optimize the target ANS. We conduct gradient descent and modify the value of $\alpha_L, \alpha_U, \beta_L, \beta_U$ if the ANS achieves a larger result under the new bounds.

The optimization problem with objective function (10) can be solved by the same algorithm, with ANS replaced by (10).

Example 4. Let us revisit Example 2. With Algorithm 1, we compute the network-wise tightest approximations for the network in Figure 2. Figure 5 shows the upper (*resp.* lower) bounds, denoted by $x''_{U,3}, x''_{U,4}$ (*resp.* $x''_{L,3}, x''_{L,4}$). By Definition 3, $x''_{U,3}, x''_{U,4}$ are tighter than the other two approximations. The resulting output range of neuron x_7 is $[0.307, 5.693]$, which is much tighter than both $[-0.079, 6.073]$ and $[0.177, 5.817]$ computed in Example 2.

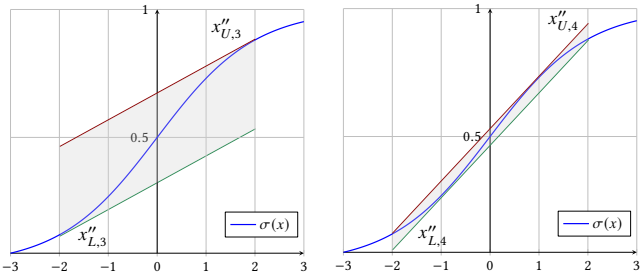


Figure 5: The network-wise tightest approximation to the activation functions on x_3 (left) and x_4 (right) in Example 2.

Note that the network-wise tightest approximations in the above example are a hybrid of both kinds of approximations in Example 2, i.e., $x''_{L,3}$ and $x''_{L,4}$, which cannot be the tightest under a single tightness criterion in [18, 51]. This echoes our advocation that solely pursuing neuron-wise tightness under existing tightness definitions may not guarantee that they are the network-wise tightest, and consequently cannot achieve precise robust verification results.

5 APPROACH FOR MULTI-HIDDEN-LAYER NETWORKS

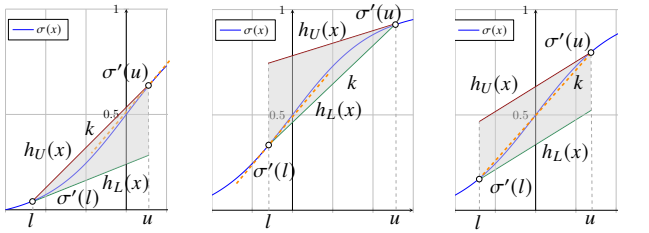
For the networks with two or more hidden layers, solving the optimization problems (7) and (8) becomes impractical due to its non-convexity. In [28], Lyu *et al.* proved that it is even non-convex to separately compute the tightest approximation for each neuron. The intractability lies in the accumulated constraints throughout the network: for any hidden layer, the input intervals of the activation functions are constrained by the approximations to the activation functions for the previous hidden layer. Neither can the optimization problems be solved on a layer basis because the objective function are network-wise. To our knowledge, no efficient algorithms or tools exist for such optimization problems. In this section, we propose computable neuron-wise tightest approximations and identify the condition when all the weights in a neural network are non-negative, the neuron-wise tightest approximations lead to being network-wise tightest.

5.1 The Neuron-Wise Tightest Approximation

Our empirical analysis in Section 3.1 reveals an insight that preserving tighter intermediate intervals during layer-by-layer propagation usually produces larger certified lower robust bounds. In the same spirit, we heuristically define the tightness of an approximation to an individual activation function in terms of the over-estimation caused by the approximation. Smaller over-estimation implies a tighter approximation. Particularly, an approximation is the *neuron-wise tightest* if it results in no overestimation of the output range of the activation function.

Definition 4 (Neuron-wise Tightness). *Let $\sigma(x)$ be an activation function with $x \in [l,u]$, and $h_U(x), h_L(x)$ be its upper and lower bounds, with α_U, α_L their slopes. $h_U(x)$ (resp. $h_L(x)$) is the neuron-wise tightest if $h_U(u) = \sigma(u)$ (resp. $h_L(l) = \sigma(l)$) and $\int_l^u h_U(x) - \sigma(x)dx$ (resp. $\int_l^u \sigma(x) - h_L(x)dx$) is minimal.*

By Definition 4, we identify three cases of defining the neuron-wise tightest approximation for each individual activation function.



(a) $\sigma'(l) < k < \sigma'(u)$ **(b)** $\sigma'(l) > k > \sigma'(u)$ **(c)** $k > \sigma'(l) \wedge k > \sigma'(u)$

Figure 6: The neuron-wise tightest linear approximation.

The three cases are defined according to the relation between the slopes of activation function at two endpoints and the slope of the line crossing the two endpoints, as classified in [18, 51]. Given an input interval $[l,u]$ for $\sigma(x)$, the slopes of $\sigma(x)$ at $(l,\sigma(l))$ and $(u,\sigma(u))$ are represented by $\sigma'(l)$ and $\sigma'(u)$, respectively; the slope of the line crossing $(l,\sigma(l))$ and $(u,\sigma(u))$ is $k = \frac{\sigma(u)-\sigma(l)}{u-l}$. Figure 6 depicts the neuron-wise tightest approximations in the following three different cases:

Case 1. When $\sigma'(l) < k < \sigma'(u)$ (Figure 6a), the line that connects the two endpoints is chosen as the upper bound, while the tangent line of $\sigma(x)$ at $(l,\sigma(l))$ as the lower bound. We then have $h_U(x) = k(x-l) + \sigma(l)$ and $h_L(x) = \sigma'(l)(x-l) + \sigma(l)$.

Case 2. When $\sigma'(u) < k < \sigma'(l)$ (Figure 6b), the tangent line of $\sigma(x)$ at $(u,\sigma(u))$ and the line crossing two endpoints are considered as the upper and lower bounds, respectively. We then have $h_U(x) = \sigma'(u)(x-u) + \sigma(u)$ and $h_L(x) = k(x-u) + \sigma(u)$.

Case 3. When $\sigma'(l) < k$ and $\sigma'(u) < k$ (Figure 6c), the tangent line of $\sigma(x)$ at $(u,\sigma(u))$ is taken as the upper bound, while the tangent line of $\sigma(x)$ at $(l,\sigma(l))$ as the lower bound. We then have $h_U(x) = \sigma'(u)(x-u) + \sigma(u)$ and $h_L(x) = \sigma'(l)(x-l) + \sigma(l)$.

Note that, Definition 4 also considers the tightness characterizations in [18, 28]. It is easy to prove that any other linear bound crossing the endpoints is less tight than the one defined in the above three cases according to the tightness definitions in [18, 28].

5.2 Neuron-Wise vs. Network-Wise

In this section, we study the relation between neuron-wise tightness and network-wise definition. Although the neuron-wise tightest approximation does not overestimate the output range of a single neuron, it cannot guarantee that the composition for all the neurons is the network-wise tightest because the monotonicity of a neuron cannot be preserved by the next layer. The monotonicity may be altered by the weights between layers because the input function of each neuron in any hidden layer is compounded by the output functions in the previous layer multiplied by the weights. Hence, a sufficient condition of passing neuron-wise tightness to the network is to avoid breaking monotonicity during the propagation.

Definition 5 (Network-wise monotonous). *Given a k-layer neural network $f : \mathbb{R}^n \to \mathbb{R}^m$ and its input $x = [x_1, ..., x_n]$, f is called network-wise monotonous if the following three conditions hold:*

(1) $\forall t_1, t_2 \in \mathbb{Z}, 1 \le t_1 \le k \wedge 1 \le t_2 \le k$,
(2) $\forall r_1, r_2 \in \mathbb{Z}, 1 \le r_1 \le n^{t_1} \wedge 1 \le r_2 \le n^{t_2}$,
(3) $\forall i \in \mathbb{Z}, 1 \le i \le n, \phi^{t_1}_{r_1}(x_i), \phi^{t_2}_{r_2}(x_i)$ *are both either monotonically increasing or decreasing.*

Table 2: Performance comparison on non-negative Sigmoid networks between NeWise (NW) and existing tools, DeepCert (DC), VeriNet (VN), and RobustVerifier (RV). $t \times n$ refers to an FNN with t layers and n neurons per layer. CNN_{t-c} denotes a CNN with t layers and c filters of size 3×3.

Dataset	Model	#Neur.	Certified Lower Bound														Time (s)
			Average							Standard Deviation							
			NW	DC	Impr. (%)	VN	Impr. (%)	RV	Impr. (%)	NW	DC	Impr. (%)	VN	Impr. (%)	RV	Impr. (%)	
MNIST	5x100	510	0.0091	0.0071	28.15 ↑	0.0071	27.25 ↑	0.0064	40.68 ↑	0.0057	0.0042	37.11 ↑	0.0042	35.48 ↑	0.0034	68.34 ↑	4.40 ±0.06
	3x700	2,110	0.0037	0.0030	24.92 ↑	0.0030	22.85 ↑	0.0029	26.62 ↑	0.0018	0.0013	41.86 ↑	0.0014	34.56 ↑	0.0013	40.77 ↑	117.25 ±0.18
	CNN_{6-5}	12,300	0.0968	0.0788	22.82 ↑	0.0778	24.37 ↑	0.0699	38.48 ↑	0.0372	0.0280	32.92 ↑	0.0276	35.09 ↑	0.0212	75.70 ↑	5.69 ±0.43
	3x50	160	0.0105	0.0088	19.23 ↑	0.0088	19.50 ↑	0.0080	31.42 ↑	0.0051	0.0038	32.72 ↑	0.0038	32.72 ↑	0.0030	71.86 ↑	0.14 ±0.00
	3x100	310	0.0139	0.0120	15.46 ↑	0.0120	15.56 ↑	0.0111	25.47 ↑	0.0071	0.0057	24.82 ↑	0.0058	23.30 ↑	0.0046	53.13 ↑	2.21 ±0.02
	CNN_{5-5}	10,680	0.0801	0.0708	13.14 ↑	0.0704	13.75 ↑	0.0683	17.32 ↑	0.0238	0.0200	18.87 ↑	0.0198	20.50 ↑	0.0180	32.35 ↑	2.89 ±0.30
	CNN_{3-2}	2,514	0.0521	0.0483	7.82 ↑	0.0483	7.94 ↑	0.0478	8.88 ↑	0.0180	0.0161	12.13 ↑	0.0160	12.41 ↑	0.0156	15.44 ↑	0.17 ±0.04
	CNN_{4-5}	8,680	0.0505	0.0473	6.68 ↑	0.0471	7.26 ↑	0.0464	8.81 ↑	0.0207	0.0186	11.26 ↑	0.0183	12.84 ↑	0.0175	17.80 ↑	1.18 ±0.20
	CNN_{3-4}	5,018	0.0448	0.0422	6.09 ↑	0.0421	6.24 ↑	0.0419	6.98 ↑	0.0156	0.0142	9.71 ↑	0.0142	10.18 ↑	0.0139	12.56 ↑	0.30 ±0.08
Fashion MNIST	4x100	410	0.0312	0.0188	65.48 ↑	0.0194	60.62 ↑	0.0159	96.22 ↑	0.0403	0.0210	92.28 ↑	0.0220	83.20 ↑	0.0176	129.33 ↑	3.27 ±0.04
	3x100	310	0.0326	0.0263	24.02 ↑	0.0270	21.03 ↑	0.0238	36.87 ↑	0.0335	0.0262	27.67 ↑	0.0282	18.92 ↑	0.0234	43.22 ↑	2.19 ±0.02
	CNN_{5-5}	10,680	0.1303	0.1155	12.81 ↑	0.1151	13.22 ↑	0.1088	19.73 ↑	0.0830	0.0714	16.23 ↑	0.0721	15.10 ↑	0.0636	30.53 ↑	2.91 ±0.32
	CNN_{3-2}	2,514	0.0790	0.0713	10.79 ↑	0.0713	10.74 ↑	0.0695	13.68 ↑	0.0497	0.0416	19.55 ↑	0.0418	19.06 ↑	0.0385	29.01 ↑	0.18 ±0.05
	CNN_{4-5}	8,680	0.0959	0.0868	10.40 ↑	0.0864	10.90 ↑	0.0840	14.19 ↑	0.0562	0.0486	15.51 ↑	0.0482	16.52 ↑	0.0453	24.03 ↑	1.19 ±0.20
	CNN_{3-4}	5,018	0.0747	0.0694	7.52 ↑	0.0693	7.72 ↑	0.0681	9.70 ↑	0.0465	0.0410	13.32 ↑	0.0409	13.59 ↑	0.0391	18.88 ↑	0.31 ±0.08
CIFAR10	9x100	910	0.0315	0.0211	49.03 ↑	0.0214	46.94 ↑	0.0192	63.84 ↑	0.0280	0.0183	52.70 ↑	0.0186	50.32 ↑	0.0132	111.50 ↑	4.92 ±0.05
	6x100	610	0.0222	0.0174	27.08 ↑	0.0176	26.14 ↑	0.0170	30.22 ↑	0.0165	0.0118	40.05 ↑	0.0120	37.82 ↑	0.0111	48.38 ↑	3.05 ±0.02
	5x100	510	0.0200	0.0167	19.76 ↑	0.0167	19.47 ↑	0.0163	22.47 ↑	0.0137	0.0104	31.80 ↑	0.0104	31.42 ↑	0.0099	38.59 ↑	2.44 ±0.01
	3x50	160	0.0206	0.0178	15.43 ↑	0.0179	14.66 ↑	0.0176	16.88 ↑	0.0144	0.0113	27.57 ↑	0.0115	25.24 ↑	0.0110	31.30 ↑	0.42 ±0.00
	4x100	410	0.0161	0.0140	15.23 ↑	0.0140	15.23 ↑	0.0138	16.56 ↑	0.0111	0.0089	24.61 ↑	0.0090	23.63 ↑	0.0087	27.76 ↑	1.84 ±0.01
	CNN_{3-4}	6,746	0.0187	0.0181	3.38 ↑	0.0181	3.32 ↑	0.0181	3.43 ↑	0.0109	0.0103	5.93 ↑	0.0103	5.83 ↑	0.0103	6.13 ↑	0.56 ±0.08
	CNN_{3-2}	3,378	0.0185	0.0180	2.49 ↑	0.0180	2.55 ↑	0.0180	2.67 ↑	0.0125	0.0120	4.34 ↑	0.0120	4.34 ↑	0.0120	4.60 ↑	0.31 ±0.06

Intuitively, a monotonous network requires all the neurons to share the same monotonicity w.r.t. the input so that they can achieve the maximum or minimum on the same input.

Theorem 2. *The composition of the neuron-wise tightest approximations is the network-wise tightest if the network satisfies the following two conditions:*

(1) *For the first layer, the items in each column of the weight matrix are all positive or negative;*
(2) *Every item in weights between remained layers is non-negative.*

Theorem 2 holds as both conditions guarantee the monotonicity of the network. In particular, starting from the second layer, the neuron-wise tightness is preserved with only non-negative weights during the layer-wise propagation (the second condition).

Example 5. Assume that we replace the three negative weights of the neural network in Figure 2 with 1, 5, and 1, respectively. The tightest approximations to x_3 and x_4, returned by Algorithm 1, are exactly the same as those returned by the neuron-wise tightest approximations in Section 5.1 (i.e., $[1.430, 10.570]$).

6 EXPERIMENTS

We evaluate our approximation method concerning both precision and efficiency in the robustness verification of Sigmoid-like neural networks. Our goal is threefold:

(1) To validate our mathematical proof of Theorem 2 via extensive experimental results (i.e., always returning the largest certified lower bounds for non-negative networks);
(2) To demonstrate that Algorithm 1 can always compute tighter lower bounds for 1-hidden-layer networks;
(3) To explore our approach's effectiveness under general neural networks with *mixed* weights.

6.1 Benchmarks and Experimental Setup

Competitors. We consider three representative approximations in the literature: DeepCert [51], VeriNet [18], and RobustVerifier [26]. For a fair comparison, we implemented in Python all the competing approaches including our new algorithm called NeWise.

Datasets and Networks. We have conducted three sets of experiments on fully connected (FNNs) and convolutional (CNNs) networks: We focus on CNNs due to their effectiveness in a wide range of visual recognition applications [22, 27, 30, 34, 43]; we also consider FNNs to expand the architecture variety. We trained all the networks on the image databases MNIST [23], Fashion MNIST [53], and CIFAR10 [21]. We chose the first 100 images from the test set of each dataset as in [5, 51, 55], among which only correctly-classified images by the neural network are considered in our experiments.

For each network architecture, we trained three variant neural networks using the Sigmoid, Tanh, and Arctan activation functions, respectively. In Experiment **I** the networks contain only non-negative weights. We used Adam or SGD optimizer with at least 50 epochs of batch size 128. The test set accuracy of networks trained on MNIST, Fashion MNIST, and CIFAR10 is around 0.9, 0.85, and 0.4, respectively. In Experiment **II** we used pre-trained models [2, 40, 51] (as in Table 1, Section 3.2) with no constraint on the weights.

Note that, to reduce space complexity, the number of neurons in FNNs can be considerably fewer than that in CNNs [24]. Nonetheless, FNNs can still achieve up to 0.99 test accuracy.

Metrics. We use certified lower bound to assess *effectiveness*, and $(\epsilon' - \epsilon)/\epsilon$ to quantify the precision improvement, where ϵ' and ϵ denote the lower bounds certified by NeWise and each competing approach, respectively. We consider both *average* and *standard deviation* (SD) of certified lower bounds; in particular, SD is a suitable measure of sensitivity of the approximations to input images: A larger SD implies a better sensitivity [51]. For *efficiency*, we record the average computation time over the (correctly-classified) images.

Experimental Setup. All the experiments were conducted on a workstation running Ubuntu 18.04 with a 2.35GHz 32-core AMD EPYC 7452 CPU and 128 GB memory.

6.2 Experimental Results

Experiment I. Table 2 shows the comparison results for 22 Sigmoid models with non-negative weights. Regarding the precision of verification results, our NeWise computes significantly larger certified lower bounds than the competitors for *all* the models. In

Table 3: Comparison on non-negative Tanh networks.

Dataset	Model	Certified Lower Bound (Standard Deviation)						
		NW	DC	Impr. (%)	VN	Impr. (%)	RV	Impr. (%)
MNIST	5x100	0.0018	0.0003	436.36 ↑	0.0006	195.00 ↑	0.0006	216.07 ↑
	3x700	0.0027	0.0006	380.70 ↑	0.0009	191.49 ↑	0.0009	191.49 ↑
	3x400	0.0018	0.0005	291.11 ↑	0.0008	128.57 ↑	0.0007	137.84 ↑
	CNN_{6-5}	0.0243	0.0066	269.35 ↑	0.0131	85.94 ↑	0.0087	179.13 ↑
	3x50	0.0012	0.0007	67.12 ↑	0.0010	27.08 ↑	0.0008	45.24 ↑
	CNN_{3-4}	0.0067	0.0041	64.69 ↑	0.0058	14.80 ↑	0.0055	20.83 ↑
	CNN_{5-5}	0.0108	0.0069	56.67 ↑	0.0092	17.50 ↑	0.0087	24.54 ↑
	CNN_{4-5}	0.0074	0.0050	49.80 ↑	0.0063	17.19 ↑	0.0060	23.63 ↑
Fashion MNIST	3x100	0.0156	0.0077	104.05 ↑	0.0110	42.04 ↑	0.0091	70.79 ↑
	CNN_{4-5}	0.0188	0.0097	95.03 ↑	0.0139	35.83 ↑	0.0129	45.82 ↑
	CNN_{6-5}	0.0329	0.0180	82.76 ↑	0.0242	35.67 ↑	0.0180	82.66 ↑
	2x100	0.0109	0.0060	81.56 ↑	0.0085	28.59 ↑	0.0077	41.95 ↑
	2x200	0.0102	0.0058	76.91 ↑	0.0080	27.06 ↑	0.0076	33.55 ↑
	CNN_{5-5}	0.0201	0.0117	71.83 ↑	0.0158	27.03 ↑	0.0125	60.69 ↑
CIFAR10	3x200	0.0176	0.0048	271.37 ↑	0.0092	90.91 ↑	0.0080	119.40 ↑
	3x50	0.0111	0.0048	129.96 ↑	0.0077	44.73 ↑	0.0071	56.32 ↑
	3x100	0.0435	0.0227	91.80 ↑	0.0270	61.35 ↑	0.0256	69.66 ↑
	3x400	0.0441	0.0233	89.15 ↑	0.0291	51.58 ↑	0.0253	74.69 ↑
	CNN_{3-2}	0.0106	0.0094	12.95 ↑	0.0102	4.01 ↑	0.0102	4.21 ↑
	CNN_{3-4}	0.0062	0.0058	7.04 ↑	0.0061	1.80 ↑	0.0061	1.96 ↑
	CNN_{3-5}	0.0066	0.0062	6.47 ↑	0.0065	1.86 ↑	0.0065	2.02 ↑

particular, for *average*, NeWise achieves up to 96.22% improvement, i.e., FNNs with 4 hidden layers trained on Fashion MNIST. NeWise improves the precision even more with *standard deviation* (up to 129.33%). This indicates that our approach is more sensitive to input images compared to the other approaches: The more the certified lower bound is improved, the larger deviation the network exhibits.

Regarding efficiency, all the approaches incur similar overhead as expected (they share the same complexity, i.e., $O(1)$ on each neuron). We use $p \pm q$ to denote their average time cost p and the size of the interval $2q$.

Table 3 presents the results on the Tanh models. NeWise computes *even larger* certified lower bounds, e.g., with up to 436.36%. We omit the similar time overheads in Table 3; see Appendix C for the complete results, including those on the Arctan models.

All these experimental results provide strong independent validation of our mathematical proof of Theorem 2.

Experiment II. We evaluate the performance of Algorithm 1 on 1-hidden-layer networks. Table 4 shows the comparison results with the other three tools. We only show the metric of standard deviation due to space limit. Complete results are available in the appendix Appendix C . As shown in the table, Algorithm 1 can compute larger bounds with up to 143.24% improvement.

Regarding efficiency, the searching algorithm needs more time because it is in polynomial time, unlike the constant-time approach for the non-negative modes. Nevertheless, the gradient-decent-based approach has been proven an efficient and practical solution to such convex optimization problems [17]. When the size of a network is reasonably small, such overhead is acceptable, compared with the improvement of the verification results.

Experiment III. Despite the infeasibility of network-wise tightest approximations in the general case (Section 3.3), we have explored the performance of our approximation method and the competitors on the networks of mixed weights. Table 5 shows the certified lower bounds returned by each approach for 11 CNNs and 9 FNNs.

First, the performance of each approach as compared with the others varies under different mixed-weight models. This coincides with our analysis in Section 3.3: Pure neuron-wise tightness does not imply a network-wise tightness. Moreover, we observe that our NeWise performs surprisingly better than other approaches on all the experimented CNNs, while DeepCert and VeriNet return

Table 4: Performance comparison of Algorithm 1 with DeepCert (DC), VeriNet (VN), and RobustVerifier (RV) on 1-hidden-layer Sigmoid networks. $t \times n$ refers to an FNN with t layers and n neurons per layer. CNN_{t-c-f} denotes a CNN with t layers and c filters of size $f \times f$. $*$ and $+$ mark the models trained on MNIST and Fashion MNIST, respectively.

Arch.	Model	Certified Lower Bound (Standard Deviation)						
		Alg.1	DC	Impr. (%)	VN	Impr. (%)	RV	Impr. (%)
CNN	CNN_{2-1-5}^{*}	0.0312	0.0145	114.58 ↑	0.0143	118.64 ↑	0.0137	127.90↑
	CNN_{2-2-5}^{*}	0.0321	0.0208	54.12 ↑	0.0207	55.32 ↑	0.0187	71.42 ↑
	CNN_{2-3-5}^{*}	0.0308	0.0197	56.16 ↑	0.0196	56.64 ↑	0.0176	74.99 ↑
	CNN_{2-4-5}^{*}	0.0489	0.0233	109.96 ↑	0.0232	110.96 ↑	0.0210	132.97↑
	CNN_{2-5-3}^{*}	0.0408	0.0182	123.79 ↑	0.0182	124.65 ↑	0.0176	131.93↑
	CNN_{2-1-5}^{+}	0.0484	0.0385	25.71 ↑	0.0386	25.38 ↑	0.0349	38.80 ↑
	CNN_{2-2-5}^{+}	0.0489	0.0353	38.39 ↑	0.0355	37.65 ↑	0.0311	57.12 ↑
	CNN_{2-3-5}^{+}	0.0511	0.0371	37.55 ↑	0.0374	36.55 ↑	0.0344	48.46 ↑
	CNN_{2-4-5}^{+}	0.0546	0.0366	49.19 ↑	0.0367	48.71 ↑	0.0337	62.23 ↑
	CNN_{2-5-3}^{+}	0.0378	0.0340	11.28 ↑	0.0340	11.12 ↑	0.0313	20.97 ↑
FNN	1x50 $*$	0.0152	0.0082	85.10 ↑	0.0085	78.56 ↑	0.0062	143.24 ↑
	1x100$*$	0.0119	0.0064	86.44 ↑	0.0066	79.14 ↑	0.0050	138.86 ↑
	1x150$*$	0.0125	0.0083	50.87 ↑	0.0085	46.27 ↑	0.0064	94.35 ↑
	1x200$*$	0.0122	0.0074	64.86 ↑	0.0076	61.38 ↑	0.0058	111.44 ↑
	1x250$*$	0.0091	0.0075	22.15 ↑	0.0076	20.21 ↑	0.0060	52.43 ↑
	1x50 $^{+}$	0.0147	0.0117	25.54 ↑	0.0122	20.90 ↑	0.0089	65.41 ↑
	$1x100^{+}$	0.0167	0.0119	39.65 ↑	0.0123	36.12 ↑	0.0088	90.13 ↑
	$1x150^{+}$	0.0171	0.0120	42.52 ↑	0.0123	39.27 ↑	0.0090	89.87 ↑
	$1x200^{+}$	0.0207	0.0129	60.06 ↑	0.0132	56.55 ↑	0.0096	115.41 ↑
	$1x250^{+}$	0.0192	0.0126	52.45 ↑	0.0128	50.42 ↑	0.0096	101.24 ↑

larger certified lower bounds on the FNNs. The results evidenced network architecture is another factor influencing the verification. One possible reason is that a convolutional neural network is more possible to be monotonic based on the fact that the neurons' weights on the same layer are constrained to be identical [24].

Finally, we observe that average and standard deviation share the same increase/decrease trends. This indicates that a tighter approximation is more sensitive to the input images, which conforms to our conclusion in Experiment **I**.

6.3 Threats to Validity

We discuss potential threats to the validity of our approach in terms of its application domains.

Neural networks with ReLU activation functions. Despite the focus on the Sigmoid-like activation functions, our approach is also applicable to the ReLU activation functions. A ReLU function $\sigma(x) = max(x, 0)$, with $x \in [l, u]$, only needs approximation when $l < 0$ and $u > 0$; the upper, resp. lower, linear bound would be then $y = \frac{u}{u-l}(x - l)$, resp. $y = 0$. Hence, the approximation is the tightest for non-negative neural networks. However, linear approximation is not a necessity for ReLU due to its piece-wise linearity. There are more precise (both sound and complete) verification approaches (by using, e.g., SMT [20] and Mixed Integer Linear Programming [6]) which could compute larger certified lower bounds.

Fully Connected Neural Networks (FNNs) with Mixed Weights. For such networks, it is generally unpredictable which approach would compute the most precise verification result (despite a 10% decrease on average in our approach). To the best of our knowledge, the only feasible way to examine a proposed approximation under non-trivial FNNs is by empirical analysis. Tackling this fundamentally and efficiently remains to be an open research problem.

7 RELATED WORK

This work is a sequel to many pioneering efforts, which we classify into the following three categories.

Table 5: Performance comparison with DeepCert, VeriNet, and RobustVerifier on mixed-weights Sigmoid networks. ∗, +, and # mark the models trained on MNIST, Fashion MNIST, and CIFAR10, respectively.

Arch.	Model	#Neur.	Certified Lower Bound													Time (s)
			Average							Standard Deviation						
			NW	DC	Impr. (%)	VN	Impr. (%)	RV	Impr. (%)	NW	DC	Impr. (%)	VN	Impr. (%)	RV	Impr. (%)	
CNN	$CNN_{3\_2}^{*}$	2,514	0.0607	0.0579	4.92 ↑	0.0580	4.67 ↑	0.0569	6.84 ↑	0.0219	0.0202	8.06 ↑	0.0204	7.37 ↑	0.0192	13.85 ↑	0.18 ±0.04
	$CNN_{3\_4}^{*}$	5,018	0.0478	0.0472	1.17 ↑	0.0472	1.29 ↑	0.0464	2.95 ↑	0.0155	0.0153	1.11 ↑	0.0153	1.44 ↑	0.0147	5.87 ↑	0.31 ±0.09
	$CNN_{4\_5}^{*}$	8,680	0.0570	0.0539	5.64 ↑	0.0543	5.03 ↑	0.0522	9.16 ↑	0.0157	0.0145	8.64 ↑	0.0146	7.52 ↑	0.0132	19.36 ↑	1.18 ±0.20
	$CNN_{5\_5}^{*}$	10,680	0.0581	0.0548	6.06 ↑	0.0550	5.63 ↑	0.0513	13.42 ↑	0.0157	0.0142	10.48 ↑	0.0144	8.80 ↑	0.0120	30.81 ↑	3.00 ±0.38
	$CNN_{6\_5}^{*}$	12,300	0.0624	0.0590	5.71 ↑	0.0588	6.00 ↑	0.0541	15.27 ↑	0.0171	0.0153	12.03 ↑	0.0153	11.96 ↑	0.0123	39.72 ↑	5.72 ±0.45
	$CNN_{8\_5}^{*}$	14,570	0.1191	0.0878	35.58 ↑	0.0882	35.02 ↑	0.0685	73.75 ↑	0.0361	0.0248	45.60 ↑	0.0255	41.66 ↑	0.0163	122.22↑	15.39±0.83
	$CNN_{4\_5}^{+}$	8,680	0.0747	0.0720	3.73 ↑	0.0720	3.79 ↑	0.0666	12.16 ↑	0.0413	0.0376	9.85 ↑	0.0378	9.12 ↑	0.0313	31.73 ↑	1.19 ±0.21
	$CNN_{5\_5}^{+}$	10,680	0.0704	0.0676	4.14 ↑	0.0676	4.14 ↑	0.0605	16.45 ↑	0.0347	0.0318	9.03 ↑	0.0320	8.41 ↑	0.0245	41.76 ↑	3.01 ±0.39
	$CNN_{6\_5}^{+}$	12,300	0.0735	0.0695	5.77 ↑	0.0691	6.37 ↑	0.0627	17.32 ↑	0.0368	0.0341	7.97 ↑	0.0340	8.35 ↑	0.0278	32.61 ↑	5.83 ±0.53
	$CNN_{3\_2}^{\#}$	3,378	0.0314	0.0312	0.58 ↑	0.0312	0.61 ↑	0.0311	1.06 ↑	0.0172	0.0169	1.65 ↑	0.0169	1.84 ↑	0.0168	2.69 ↑	0.31 ±0.06
	$CNN_{6\_5}^{\#}$	17,110	0.0229	0.0224	2.19 ↑	0.0223	2.46 ↑	0.0213	7.72 ↑	0.0158	0.0153	3.20 ↑	0.0153	3.20 ↑	0.0141	12.05 ↑	10.34±0.70
FNN	$3x50^{*}$	160	0.0069	0.0076	-8.82 ↓	0.0077	-9.77 ↓	0.0065	6.62 ↑	0.0025	0.0027	-6.37 ↓	0.0028	-9.42 ↓	0.0021	19.05 ↑	0.14 ±0.00
	$3x100^{*}$	310	0.0078	0.0086	-9.44 ↓	0.0087	-10.79↓	0.0074	4.58 ↑	0.0026	0.0029	-10.14↓	0.0030	-12.88↓	0.0023	10.30 ↑	2.21 ±0.00
	$3x200^{*}$	610	0.0080	0.0091	-11.69↓	0.0091	-12.36↓	0.0079	1.01 ↑	0.0026	0.0030	-14.14↓	0.0031	-16.39↓	0.0024	4.94 ↑	7.49 ±0.06
	$5x100^{*}$	510	0.0058	0.0061	-5.27 ↓	0.0062	-6.66 ↓	0.0052	10.79↑	0.0024	0.0025	-5.98 ↓	0.0026	-8.88 ↓	0.0021	12.38 ↑	4.38 ±0.01
	$6x500^{*}$	3,010	0.0685	0.0778	-11.95↓	0.0776	-11.73↓	0.0665	3.04 ↑	0.0186	0.0210	-11.56↓	0.0211	-11.69↓	0.0152	21.98 ↑	153.82 ±0.59
	$3x50^{+}$	160	0.0092	0.0101	-9.67 ↓	0.0102	-10.29↓	0.0086	6.64 ↑	0.0035	0.0037	-6.74 ↓	0.0038	-9.90 ↓	0.0030	15.72 ↑	0.14 ±0.00
	$5x100^{+}$	510	0.0071	0.0078	-8.51 ↓	0.0079	-10.01↓	0.0066	8.23 ↑	0.0036	0.0040	-8.79 ↓	0.0041	-11.68↓	0.0033	11.69 ↑	4.40 ±0.03
	$3x50^{\#}$	160	0.0041	0.0045	-10.57↓	0.0045	-10.18↓	0.0042	-2.17↓	0.0018	0.0021	-14.90↓	0.0021	-14.49↓	0.0017	2.91 ↑	0.42 ±0.01
	$5x100^{\#}$	510	0.0033	0.0037	-10.60↓	0.0037	-10.60↓	0.0033	-0.60↓	0.0014	0.0017	-15.06↓	0.0017	-14.55↓	0.0013	5.22 ↑	2.45 ±0.01

Linear Approximations of Sigmoid-like activation functions. NeVer [33] uses piece-wise linear constraints for approximation and is therefore unscalable. Both CROWN [55] and CNN-Cert [5] consider the tangent line at the midpoint of $[l, u]$ as one of the linear bounds. DeepCert [51] defines a fine-grained approximation strategy by calculating the slopes of the two linear constraints according to l and u. RobustVerifier [26] leverages Taylor expansion at the midpoint of $[l, u]$. These approximations are intuitive but lack rigorous justifications or proofs for their better performance.

Lyu *et al.* [28] characterized the tightness of approximations in terms of the overestimation of output range of each hidden neuron. But they observed and admitted that by their definition tighter bounding lines do not ensure more precise results. By our definition, we show that in the case of one hidden layer, their definition also guarantees to be network-wise tightest. They proposed a gradient-based searching algorithm for near-tightest approximations under their definition. However, the algorithm has been shown difficult to scale up to large-size networks because it needs to perform on every neuron, compared with other existing constant-time approaches [18, 51]. Our work is, to the best of our knowledge, the first provably tightest, constant-time linear approximation.

Defining Tightness for Linear Approximations. There has been a shift of focus from individual neurons to multiple neurons w.r.t. defining tightness for linear approximations, but most of the work only concerns about the ReLU networks. Tjandraatmadja *et al.* [41] experimentally show that the success of approximations hinges on how closely they approximate the object that they are relaxing. Salman *et al.* [36] reveal an inherent barrier of the approximation-based approaches for the ReLU networks and require for the tightest LP-relaxed verification the tightest pre-activation upper and lower bounds of all the neurons in networks. Singh *et al.* [38] approximate multiple neurons simultaneously to obtain the tighter bounds. In contrast to this line of research, we have defined both neuron-wise and network-wise tightness to characterize linear approximations of Sigmoid-like activation functions.

Other Robustness Verification Approaches. In addition to approximation, other techniques have also been used for the robustness verification of neural networks. Abstract interpretation [11], a technique that was originally proposed for program verification, has been proven both effective and efficient in neural network verification [16, 39, 40]. These approaches also rely on over-approximation but to transform the original verification problem into dedicated abstract domains. We believe that our approximation approach is also applicable to produce more precise verification results for non-negative neural networks. Other verification methods leverage the Lipschitz continuity feature of neural networks to estimate the output ranges [10, 25, 35]. Although the approximation to an activation function can be bypassed using the Lipschitz constant, it would still be helpful to compute tighter Lipschitz constants by estimating the input range of the activation function via the approximation.

8 CONCLUDING REMARKS

We have presented *network-wise tightness*, a novel and unified characterization of the tightness of linear approximations in robustness verification of Sigmoid-like neural networks. We have shown that (i) to achieve precise verification results, activation functions in a network should *not* be approximated with the same existing neuron-wise tightness criterion; (ii) computing the network-wise tightest approximation is computationally expensive and impractical due to its non-convexity; and (iii) how to bypass the complexity barrier via a neuron-wise tightest approximation. The experimental results demonstrate that our approximation approach outperforms state-of-the-art approaches under three scenarios, i.e., non-negative networks, 1-hidden-layer networks, and convolutional networks.

Our work sheds light on the pursuit of robust neural networks via tightening linear approximations. The ineffectiveness of neuron-wise tightness on general networks calls for new, potentially hybrid, approximation strategies. The intrinsic high complexity in computing the network-wise tightest approximation motivates us to rethink of both fundamental and the heuristic trade-offs between precision and efficiency in neural network verification. For mixed-weight neural networks, there may be latent factors that could influence the tightness of approximations. One promising direction is to explore possible combinations of existing tightness characterizations to achieve network-wise tightness while taking into account the features of weight distributions and network architectures.

REFERENCES

[1] Afan Ali and Fan Yangyu. 2017. Automatic modulation classification using deep learning based on sparse autoencoders with nonnegativity constraints. *IEEE signal processing letters* 24, 11 (2017), 1626–1630.

[2] aptx4869tjx. 2021. Pretrained Models. https://github.com/aptx4869tjx/train_network.

[3] Christel Baier and Joost-Pieter Katoen. 2008. *Principles of model checking*. MIT press.

[4] Teodora Baluta, Zheng Leong Chua, Kuldeep S. Meel, and Prateek Saxena. 2021. Scalable Quantitative Verification For Deep Neural Networks. In *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*. IEEE, 312–323.

[5] Akhilan Boopathy, Tsui-Wei Weng, Pin-Yu Chen, Sijia Liu, and Luca Daniel. 2019. CNN-Cert: An Efficient Framework for Certifying Robustness of Convolutional Neural Networks. In *AAAI Conference on Artificial Intelligence (AAAI)*. 3240–3247.

[6] Elena Botoeva, Panagiotis Kouvaros, Jan Kronqvist, Alessio Lomuscio, and Ruth Misener. 2020. Efficient Verification of ReLU-Based Neural Networks via Dependency Analysis. In *AAAI Conference on Artificial Intelligence (AAAI)*. AAAI Press, 3291–3299.

[7] Nicholas Carlini and David Wagner. 2017. Towards evaluating the robustness of neural networks. In *IEEE symposium on security and privacy (S&P)*. IEEE, 39–57.

[8] Fabrício Ceschin, Marcus Botacin, Heitor Murilo Gomes, Luiz S Oliveira, and André Grégio. 2019. Shallow security: On the creation of adversarial variants to evade machine learning-based malware detectors. In *Proceedings of the 3rd Reversing and Offensive-oriented Trends Symposium*. 1–9.

[9] Edmund M Clarke. 1997. Model checking. In *International Conference on Foundations of Software Technology and Theoretical Computer Science*. Springer, 54–56.

[10] Patrick L Combettes and Jean-Christophe Pesquet. 2020. Lipschitz certificates for layered network structures driven by averaged activation operators. *SIAM Journal on Mathematics of Data Science* 2, 2 (2020), 529–557.

[11] Patrick Cousot and Radhia Cousot. 1977. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *ACM Symposium on Principles of Programming Languages (POPL)*. 238–252.

[12] Isaac Dunn, Hadrien Pouget, Daniel Kroening, and Tom Melham. 2021. Exposing previously undetectable faults in deep neural networks. In *30th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*. ACM, 56–66.

[13] Souradeep Dutta, Susmit Jha, Sriram Sankaranarayanan, and Ashish Tiwari. 2018. Output Range Analysis for Deep Feedforward Neural Networks. In *NASA Formal Methods Symposium (NFM)*. Springer, 121–138.

[14] Ruediger Ehlers. 2017. Formal verification of piece-wise linear feed-forward neural networks. In *International Symposium on Automated Technology for Verification and Analysis*. Springer, 269–286.

[15] William Fleshman, Edward Raff, Jared Sylvester, et al. 2018. Non-Negative Networks Against Adversarial Attacks. *CoRR* abs/1806.06108 (2018). http://arxiv.org/abs/1806.06108

[16] 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 *IEEE Symposium on Security and Privacy (S&P)*. IEEE, 3–18.

[17] Saad Hikmat Haji and Adnan Mohsin Abdulazeez. 2021. Comparison of optimization techniques based on gradient descent algorithm: A review. *PalArch's Journal of Archaeology of Egypt/Egyptology* 18, 4 (2021), 2715–2743.

[18] Patrick Henriksen and Alessio R. Lomuscio. 2020. Efficient Neural Network Verification via Adaptive Refinement and Adversarial Search. In *European Conference on Artificial Intelligence (ECAI)*. IOS Press, 2513–2520.

[19] Omid Kargarnovin, Amir Mahdi Sadeghzadeh, and Rasool Jalili. 2021. Mal2GCN: A Robust Malware Detection Approach Using Deep Graph Convolutional Networks With Non-Negative Weights. *arXiv preprint arXiv:2108.12473* (2021).

[20] 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 *International Conference on Computer Aided Verification (CAV)*. Springer, 97–117.

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

[22] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2017. ImageNet classification with deep convolutional neural networks. *Commun. ACM* 60, 6 (2017), 84–90.

[23] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-Based Learning Applied to Document Recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

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

[25] Sungyoon Lee, Jaewook Lee, and Saerom Park. 2020. Lipschitz-certifiable training with a tight outer bound. *Annual Conference on Neural Information Processing Systems (NeurIPS)* 33 (2020), 16891–16902.

[26] 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 *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 11418–11427.

[27] Jonathan Long, Evan Shelhamer, and Trevor Darrell. 2015. Fully convolutional networks for semantic segmentation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, 3431–3440.

[28] Zhaoyang Lyu, Ching-Yun Ko, Zhifeng Kong, Ngai Wong, Dahua Lin, and Luca Daniel. 2020. Fastened CROWN: Tightened Neural Network Robustness Certificates. In *AAAI Conference on Artificial Intelligence (AAAI)*. 5037–5044.

[29] Ana Neacsu, Jean-Christophe Pesquet, and Corneliu Burileanu. 2020. Accuracy-Robustness Trade-Off for Positively Weighted Neural Networks. In *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 8389–8393.

[30] Tianxiang Pan, Bin Wang, Guiguang Ding, and Jun-Hai Yong. 2017. Fully Convolutional Neural Networks with Full-Scale-Features for Semantic Segmentation. In *AAAI Conference on Artificial Intelligence (AAAI)*. AAAI Press, 4240–4246.

[31] Harsh Nilesh Pathak and Randy Clinton Paffenroth. 2020. Non-convex Optimization Using Parameter Continuation Methods for Deep Neural Networks. *Deep Learning Applications, Volume 2* 1232 (2020), 273–298.

[32] Brandon Paulsen, Jingbo Wang, Jiawei Wang, and Chao Wang. 2020. NEUROD-IFF: Scalable Differential Verification of Neural Networks using Fine-Grained Approximation. In *International Conference on Automated Software Engineering (ASE)*. IEEE, 784–796.

[33] Luca Pulina and Armando Tacchella. 2010. An Abstraction-Refinement Approach to Verification of Artificial Neural Networks. In *International Conference on Computer Aided Verification (CAV)*. Springer, 243–257.

[34] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. 2017. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39, 6 (2017), 1137–1149.

[35] Wenjie Ruan, Xiaowei Huang, and Marta Kwiatkowska. 2018. Reachability analysis of deep neural networks with provable guarantees. In *International Joint Conference on Artificial Intelligence(IJCAI)*. 2651–2659.

[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 *Annual Conference on Neural Information Processing Systems (NeurIPS)*. 9832–9842.

[37] Marco Sälzer and Martin Lange. 2021. Reachability Is NP-Complete Even for the Simplest Neural Networks. *CoRR* abs/2108.13179 (2021).

[38] Gagandeep Singh, Rupanshu Ganvir, Markus Püschel, and Martin T. Vechev. 2019. Beyond the Single Neuron Convex Barrier for Neural Network Certification. In *Annual Conference on Neural Information Processing Systems (NeurIPS)*. 15072–15083.

[39] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin T Vechev. 2018. Fast and Effective Robustness Certification.. In *Advances in Neural Information Processing Systems (NeurIPS)*. 10825–10836.

[40] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. 2019. An Abstract Domain for Certifying Neural Networks. *Proceedings of the ACM on Programming Languages (POPL)* (2019), 1–30.

[41] 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 *Annual Conference on Neural Information Processing Systems (NeurIPS)*.

[42] Vincent Tjeng, Kai Xiao, and Russ Tedrake. 2019. Evaluating Robustness of Neural Networks with Mixed Integer Programming. In *International Conference on Learning Representations (ICLR)*.

[43] Alexander Toshev and Christian Szegedy. 2014. DeepPose: Human Pose Estimation via Deep Neural Networks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, 1653–1660.

[44] Hoang-Dung Tran, Xiaodong Yang, Diego Manzanas Lopez, Patrick Musau, Luan Viet Nguyen, Weiming Xiang, Stanley Bak, and Taylor T Johnson. 2020. NNV: the neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In *International Conference on Computer Aided Verification(CAV)*. Springer, 3–17.

[45] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. 2018. Efficient formal safety analysis of neural networks. In *Annual Conference on Neural Information Processing Systems (NeurIPS)*. 6369–6379.

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

[47] Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. 2021. Beta-crown: Efficient bound propagation with per-neuron split constraints for neural network robustness verification. *Annual Conference on Neural Information Processing Systems (NeurIPS)* 34 (2021).

[48] Lily Weng, Huan Zhang, Hongge Chen, Zhao Song, et al. 2018. Towards Fast Computation of Certified Robustness for ReLU Networks. In *International Conference on Machine Learning (ICML)*. PMLR, 5276–5285.

[49] Tsui-Wei Weng, Huan Zhang, Hongge Chen, Zhao Song, et al. 2018. Towards Fast Computation of Certified Robustness for ReLU Networks. In *International Conference on Machine LearningICML*, Vol. 80. PMLR, 5273–5282.
[50] Jeannette M Wing. 2021. Trustworthy AI. *Commun. ACM* 64, 10 (2021), 64–71.
[51] Yiting Wu and Min Zhang. 2021. Tightening Robustness Verification of Convolutional Neural Networks with Fine-Grained Linear Approximation. In *AAAI Conference on Artificial Intelligence (AAAI)*. 11674–11681.
[52] Weiming Xiang, Hoang-Dung Tran, and Taylor T Johnson. 2018. Output reachable set estimation and verification for multilayer neural networks. *IEEE transactions on neural networks and learning systems* 29, 11 (2018), 5777–5783.
[53] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *CoRR* abs/1708.07747 (2017).
[54] Ming Yan, Junjie Chen, Xiangyu Zhang, Lin Tan, Gan Wang, and Zan Wang. 2021. Exposing numerical bugs in deep learning via gradient back-propagation. In *29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, 627–638.
[55] 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 *Annual Conference on Neural Information Processing Systems (NeurIPS)*. 4944–4953.

A A COUNTEREXAMPLE FOR ARCTAN

This section shows a counterexample for Arctan, where a tighter linear bound line for the activation function actually produces a less precise verification result.

The counterexample is shown in Figure 7. For simplicity, weights of edges are integers and biases are ignored. Let x_1 and x_2 represent the possible input values in $[-2,-1]$ and $[-1,0]$, respectively. In the output layer, if the value of x_5 is always greater than the one of x_6, we can claim that all x_1 in $[-2,-1]$ and x_2 in $[-1,0]$ are classified to the label of x_5. We introduce an auxiliary node x_7 to represent $x_5 - x_6$.

The two linear approximations in Figure 7 (a) and (b) share the same upper bounds. The lower bounds of x_3, x_4 (the red in (a)) are closer to the Arctan function than the ones (the blue in (b)). However, the output range $[-2.496, -1.572]$ of x_7 computed by the right approximation is more precise than $[-2.614, -1.572]$ computed by the left approximation.

B PROOF OF THEOREM 2

For k-layer neural network $f : \mathbb{R}^n \to \mathbb{R}^m$ satisfying the conditions in Theorem 2, we continue to use the symbol definition proposed. We assume that the upper and lower bounds of $\phi_r^t(x)$ are u_r^t and l_r^t, *i.e.*, $l_r^t \le \phi_r^t(x) \le u_r^t$. We consider the linear approximation of $\sigma(\phi_r^t(x))$ between $[l_r^t, u_r^t]$, where $\sigma(x)$ is a Sigmoid-like function. According to Definition 4, we denote the upper and lower linear bounds of $\sigma(\phi_r^t(x))$ obtained by our approach as $h_{U,r}^t(x)$ and $h_{L,r}^t(x)$, respectively.

First, we illustrate that the two conditions in the Theorem 2 guarantee that the network is monotonous.

Theorem 3. *A neural network is network-wise monotonous if the network satisfies the following two conditions:*

1. *For the first layer, for any selected $i \in \mathbb{Z}, 1 \le i \le n$, items in the ith column of W^1 are all positive or all negative;*
2. *Every item in weights from the second layer to the last layer is non-negative.*

Proof. For any selected item of input x_i, items in the i-th column of W^1 are all positive or all negative. Suppose that they are all positive, which means $\forall r \in \mathbb{Z}, 1 \le r \le n^1, w_{r,i}^1 > 0$. First we consider $t = 1$, for any value $x_i' > x_i''$ of x_i, $\phi_r^1(x_i') = w_{r,i}^1 x_i' > w_{r,i}^1 x_i'' = \phi_r^1(x_i'')$.
Suppose that for any $t \le T, 1 \le T$, $\phi_r^t(x_i') > \phi_r^t(x_i'')$ holds. Then we consider layer $T+1$, Since σ is monotonously increasing, and all items in W^{T+1} are non-negative, we have:

$$\phi_r^{T+1}(x_i') = \sum_p w_{r,p}^{T+1} \sigma(\phi_p^T(x_i')) \tag{11}$$

$$> \sum_p w_{r,p}^{T+1} \sigma(\phi_p^T(x_i'')) \tag{12}$$

$$= \phi_r^{T+1}(x_i'') \tag{13}$$

In summary, by mathematical induction, $\forall r, t, \phi_r^t(x_i)$ is monotonously increasing.

Similarly, if the items in the i-th column of W^1 are all negative, we can deduce that $\forall r, t, \phi_r^t(x_i)$ is monotonously decreasing. The network satisfies Definition 5. □

Lemma 1. *Given a k-layer neural network f satisfying Definition 5 and its input $x = [x_1, ..., x_n]$, $\forall t \in \mathbb{Z}, 1 \le t \le k$, the j-th item in layer $t+1$ satisfies:*

$$\min_{x \in \mathbb{B}_\infty(x_0,\epsilon)} \phi_j^{t+1}(x) = \sum_r w_{j,r}^{t+1} \min_{x \in \mathbb{B}_\infty(x_0,\epsilon)} \sigma(\phi_r^t(x))$$

Proof. Suppose that f is network-wise monotonously increasing. $\forall$ selected i, from Definition 5, $\forall t$, $\phi_r^t(x_i)$ and $\phi_r^{t+1}(x_i)$ are both monotonously increasing. Suppose that $x_i \in [l_i^0, u_i^0]$, $x_{min} := [l_1^0, ..., l_n^0]$, $x_{max} := [u_1^0, ..., u_n^0]$, then:

$$\begin{aligned}
\min_{x \in \mathbb{B}_\infty(x_0,\epsilon)} \phi_j^{t+1}(x) &= \phi_j^{t+1}(x_{min}) \\
&= \sum_r w_{j,r}^{t+1} \sigma(\phi_r^t(x_{min})) \\
&= \sum_r w_{j,r}^{t+1} \sigma(\min_{x \in \mathbb{B}_\infty(x_0,\epsilon)} \phi_r^t(x)) \\
&= \sum_r w_{j,r}^{t+1} \min_{x \in \mathbb{B}_\infty(x_0,\epsilon)} \sigma(\phi_r^t(x))
\end{aligned}$$ □

Next, we prove the optimality of our approximation approach from the perspective of a single neuron.

Lemma 2. *Given function $f : \mathbb{R}^n \to \mathbb{R}$, let f's definition domain be D and its range be $[l,u]$, if the definition domain of function $g : \mathbb{R} \to \mathbb{R}$ is $[l,u]$, we can define compound function $g \circ f : \mathbb{R}^n \to \mathbb{R}$. Suppose that g is a monotonically increasing function, then we have:*

$$\min_{x \in D} g \circ f(x) = g(l), \quad \max_{x \in D} g \circ f(x) = g(u).$$

Proof. Because g is monotonically increasing, there is $\forall x \in [l,u], g(l) \le g(x) \le g(u)$. Then

$$\begin{aligned}
&\min_{x \in D} g \circ f(x) = g(\min_{x \in D} f(x)) = g(l), \text{and,} \\
&\max_{x \in D} g \circ f(x) = g(\max_{x \in D} f(x)) = g(u)
\end{aligned}$$ □

Theorem 4. *Given a neural network f, let $\phi^t : \mathbb{R}^n \to \mathbb{R}^{n^t}$ be the compound function of the first t layers of f. The r-th row of ϕ^t can be denoted as $\phi_r^t : \mathbb{R}^n \to \mathbb{R}$. When the input x is bounded in a norm-ball $\mathbb{B}_\infty(x_0,\epsilon)$, ϕ_r^t ranges in $[l_r^t, u_r^t]$, which is the definition domain of $h_{U,r}^t$ and $h_{L,r}^t$. We have:*

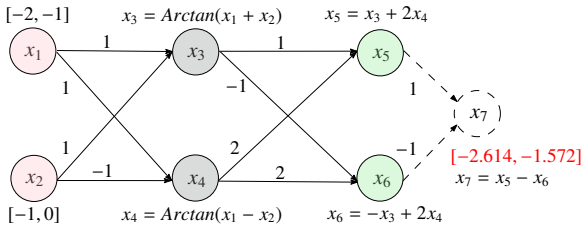
$$\min_{x \in \mathbb{B}_\infty(x_0,\epsilon)} h_{L,r}^t \circ \phi_r^t(x) = h_{L,r}^t(l_r^t), \text{and,}$$

$$\max_{x \in \mathbb{B}_\infty(x_0,\epsilon)} h_{U,r}^t \circ \phi_r^t(x) = h_{U,r}^t(u_r^t).$$

Proof. According to Definition 4, $h_{U,r}^t(x) = \alpha_{U,r}^t x + \beta_{U,r}^t$, $h_{L,r}^t(x) = \alpha_{L,r}^t x + \beta_{L,r}^t$. In our approximation, $h_{L,r}^t$ and $h_{L,r}^t$ are always monotonically increasing. By replacing f and g in Lemma 2 with ϕ_r^t and $h_{L,r}^t$ (*resp.* $h_{U,r}^t$), we can easily get the conclusion. □

Theorem 4 says that our approximation method does not overestimate the output range of current neuron. That is because $h_{L,r}^t(l_r^t) = \sigma(l_r^t)$ and $h_{U,r}^t(u_r^t) = \sigma(u_r^t)$. That is, the output range of $\sigma(\phi_r^t(x))$ after the linear approximation is still $[\sigma(l_r^t), \sigma(u_r^t)]$.

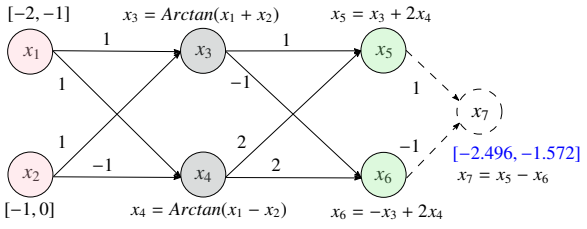
When dealing with the lower bound of $\phi_j^{t+1}(x)$ - j-th output of layer $t+1$ under the condition that $x \in \mathbb{B}_\infty(x_0,\epsilon)$, we have:



$x_{3,U} = 0.232(x_1 + x_2) - 0.554$
$x_{3,L} = 0.200(x_1 + x_2) - 0.707$
$x_{3,L} \le x_3 \le x_{3,U}$

$x_{4,U} = 0.554(x_1 - x_2)$
$x_{4,L} = 0.500(x_1 - x_2) - 0.285$
$x_{4,L} \le x_4 \le x_{4,U}$

(a) Two lower bounds (red) closer to the Arctan function.



$x_{3,U} = 0.232(x_1 + x_2) - 0.554$
$x_{3,L} = 0.100(x_1 + x_2) - 0.949$
$x_{3,L} \le x_3 \le x_{3,U}$

$x_{4,U} = 0.554(x_1 - x_2)$
$x_{4,L} = 0.200(x_1 - x_2) - 0.707$
$x_{4,L} \le x_4 \le x_{4,U}$

(b) Two lower bounds (blue) farther from the Arctan function.

Figure 7: Output ranges calculated by different linear approximations on the neural networks with Arctan.

$$\min_{x\in\mathbb{B}_\infty(x_0,\epsilon)} \phi_j^{t+1}(x) = \sum_r w_{j,r}^{t+1} \min_{x\in\mathbb{B}_\infty(x_0,\epsilon)} \sigma(\phi_r^t(x)) \tag{14}$$

$$= \sum_r w_{j,r}^{t+1} \min_{x\in\mathbb{B}_\infty(x_0,\epsilon)} h_{L,r}^t \circ \phi_r^t(x) \tag{15}$$

$$= \sum_r w_{j,r}^{t+1} h_{L,r}^t(l_r^t) \tag{16}$$

where (14) is the conclusion of Lemma 1, and (16) is derived from (15) according to Theorem 4.

Likewise, we have:

$$\max_{x\in\mathbb{B}_\infty(x_0,\epsilon)} \phi_j^{t+1}(x) = \sum w_{j,r}^t h_{U,r}^t(u_r^t). \tag{17}$$

Theorem 5. *For networks satisfying conditions in Theorem 2, our linear approximation guarantees that (16) is the largest and (17) is the smallest among all possible approximations.*

Proof. Given an arbitrary other lower and upper linear bounds $\hat{h}_{L,r}^t$ and $\hat{h}_{U,r}^t$, there are $\hat{h}_{L,r}^t(x) \le \sigma(x)$ and $\hat{h}_{U,r}^t(x) \ge \sigma(x)$ for all $x \in [l_r^t, u_r^t]$, which implies that $\hat{h}_{L,r}^t(l_r^t) \le \sigma(l_r^t)$ and $\hat{h}_{U,r}^t(u_r^t) \ge \sigma(u_r^t)$. According to our linear approximation:

$$h_{L,r}^t(l_r^t) = \sigma(l_r^t) \ge \hat{h}_{L,r}^t(l_r^t), \text{and,}$$
$$h_{U,r}^t(u_r^t) = \sigma(u_r^t) \le \hat{h}_{U,r}^t(u_r^t).$$

As a result, for $w_{j,r}^t \ge 0$, we have the following formula hold:

$$\sum w_{j,r}^t h_{L,r}^t(l_r^t) \ge \sum w_{j,r}^t \hat{h}_{L,r}^t(l_r^t), \text{and,}$$
$$\sum w_{j,r}^t h_{U,r}^t(u_r^t) \le \sum w_{j,r}^t \hat{h}_{U,r}^t(u_r^t).$$

So our linear approximation obtains the supremum of (16) and the infimum of (17). □

According to Theorem 5, our approximation obtains the upper bound of $\min_{x\in\mathbb{B}_\infty(x_0,\epsilon)} \phi_j^{t+1}(x)$. Next, we prove the network-wise tightness of our approach through Definition 3. Let (f_L, f_U) be our linear approximation of f with f_U and f_L the upper and lower bounds, respectively. Suppose that there exists another linear approximation $(\hat{f}_L, \hat{f}_U)$ and label s s.t.

$$\min_{x\in\mathbb{B}_\infty(x_0,\epsilon)} f_{L,s}(x) < \min_{x\in\mathbb{B}_\infty(x_0,\epsilon)} \hat{f}_{L,s}(x). \tag{18}$$

If the two approaches share the same bounds of nodes of (k-1)-th layer, then:

$$\min_{x\in\mathbb{B}_\infty(x_0,\epsilon)} f_{L,s}(x) = \min_{x\in\mathbb{B}_\infty(x_0,\epsilon)} \phi_{L,s}^k(x)$$
$$= \sum_r w_{s,r}^k h_{L,r}^{k-1}(l_r^{k-1})$$
$$\ge \sum_r w_{s,r}^k \hat{h}_{L,r}^{k-1}(l_r^{k-1})$$

which contradicts with (18). So the two approaches cannot share the same bounds, and there must be $\hat{l}_r^{k-1} > l_r^{k-1}$. Otherwise, $\hat{h}_{L,r}^{k-1}(\hat{l}_r^{k-1}) \le \sigma(\hat{l}_r^{k-1}) < \sigma(l_r^{k-1}) \le h_{L,r}^{k-1}(l_r^{k-1})$, meaning that (18) cannot be achieved. We have:

$$l_r^{k-1} = \min_{x\in\mathbb{B}_\infty(x_0,\epsilon)} \phi_{L,r}^{k-1}(x)$$
$$= \sum_j w_{r,j}^{k-1} h_{L,j}^{k-2}(l_j^{k-2})$$
$$< \hat{l}_r^{k-1}$$
$$= \min_{x\in\mathbb{B}_\infty(x_0,\epsilon)} \hat{\phi}_{L,r}^{k-1}(x)$$
$$= \sum_j w_{r,j}^{k-1} \hat{h}_{L,j}^{k-2}(\hat{l}_j^{k-2})$$

Similarly, there must be $\hat{l}_j^{k-2} > l_j^{k-2}$ and so on. In the end, we can deduce that $\forall i \in [1, n], \hat{l}_i^0 > l_i^0$, which are the lower bounds of x_i. However, the input range of x_i is the same for the two approaches, which is a contradiction.

To conclude, there does not exist a $(\hat{f}_L, \hat{f}_U)$ satisfying (18). Theorem 2 is proven.

C ADDITIONAL EXPERIMENTAL RESULTS

This section presents the precision and efficiency comparison of NeWise with DeepCert, VeriNet and RobustVerifier on 27 Sigmoid, 30 Tanh, and 32 Arctan models with non-negative weights. In addition, we also demonstrated the effectiveness of Algorithm 1 on 40 1-hidden-layer models with mixed weights.

Table 6: Performance comparison on non-negative Sigmoid networks between NeWise (NW) and existing tools, DeepCert (DC), VeriNet (VN), and RobustVerifier (RV). $t \times n$ refers to an FNN with t layers and n neurons per layer. CNN_{t-c} denotes a CNN with t layers and c filters of size 3×3.

Dataset	Model	#Neur.	Certified Lower Bound														Time (s)
			Average							Standard Deviation							
			NW	DC	Impr. (%)	VN	Impr. (%)	RV	Impr. (%)	NW	DC	Impr. (%)	VN	Impr. (%)	RV	Impr. (%)	
MNIST	5x100	510	0.0091	0.0071	28.15 ↑	0.0071	27.25 ↑	0.0064	40.68 ↑	0.0057	0.0042	37.11 ↑	0.0042	35.48 ↑	0.0034	68.34 ↑	4.40 ±0.06
	3x700	2,110	0.0037	0.0030	24.92 ↑	0.0030	22.85 ↑	0.0029	26.62 ↑	0.0018	0.0013	41.86 ↑	0.0014	34.56 ↑	0.0013	40.77 ↑	117.25 ±0.18
	CNN_{6-5}	12,300	0.0968	0.0788	22.82 ↑	0.0778	24.37 ↑	0.0699	38.48 ↑	0.0372	0.0280	32.92 ↑	0.0276	35.09 ↑	0.0212	75.70 ↑	5.69 ±0.43
	3x50	160	0.0105	0.0088	19.23 ↑	0.0088	19.50 ↑	0.0080	31.42 ↑	0.0051	0.0038	32.72 ↑	0.0038	32.72 ↑	0.0030	71.86 ↑	0.14 ±0.00
	3x100	310	0.0139	0.0120	15.46 ↑	0.0120	15.56 ↑	0.0111	25.47 ↑	0.0071	0.0057	24.82 ↑	0.0058	23.30 ↑	0.0046	53.13 ↑	2.21 ±0.02
	CNN_{5-5}	10,680	0.0801	0.0708	13.14 ↑	0.0704	13.75 ↑	0.0683	17.32 ↑	0.0238	0.0200	18.87 ↑	0.0198	20.50 ↑	0.0180	32.35 ↑	2.89 ±0.30
	3x200	610	0.0080	0.0071	12.54 ↑	0.0071	12.85 ↑	0.0068	17.16 ↑	0.0046	0.0037	26.43 ↑	0.0037	25.41 ↑	0.0034	37.28 ↑	10.08 ±1.40
	3x400	1,210	0.0061	0.0056	9.66 ↑	0.0056	9.86 ↑	0.0054	12.89 ↑	0.0035	0.0030	16.78 ↑	0.0030	16.39 ↑	0.0028	26.55 ↑	39.20 ±0.22
	CNN_{3-2}	2,514	0.0521	0.0483	7.82 ↑	0.0483	7.94 ↑	0.0478	8.88 ↑	0.0180	0.0161	12.13 ↑	0.0160	12.41 ↑	0.0156	15.44 ↑	0.17 ±0.04
	CNN_{4-5}	8,680	0.0505	0.0473	6.68 ↑	0.0471	7.26 ↑	0.0464	8.81 ↑	0.0207	0.0186	11.26 ↑	0.0183	12.84 ↑	0.0175	17.80 ↑	1.18 ±0.20
	CNN_{3-4}	5,018	0.0448	0.0422	6.09 ↑	0.0421	6.24 ↑	0.0419	6.98 ↑	0.0156	0.0142	9.71 ↑	0.0142	10.18 ↑	0.0139	12.56 ↑	0.30 ±0.08
Fashion MNIST	4x100	410	0.0312	0.0188	65.48 ↑	0.0194	60.62 ↑	0.0159	96.22 ↑	0.0403	0.0210	92.28 ↑	0.0220	83.20 ↑	0.0176	129.33 ↑	3.27 ±0.04
	3x100	310	0.0326	0.0263	24.02 ↑	0.0270	21.03 ↑	0.0238	36.87 ↑	0.0335	0.0262	27.67 ↑	0.0282	18.92 ↑	0.0234	43.22 ↑	2.19 ±0.02
	2x100	210	0.0306	0.0250	22.49 ↑	0.0254	20.51 ↑	0.0230	33.03 ↑	0.0287	0.0211	36.04 ↑	0.0228	25.88 ↑	0.0194	47.76 ↑	1.11 ±0.01
	3x200	610	0.0224	0.0184	21.80 ↑	0.0187	19.45 ↑	0.0170	31.63 ↑	0.0220	0.0159	38.66 ↑	0.0170	29.38 ↑	0.0143	54.42 ↑	8.16 ±2.76
	2x200	410	0.0263	0.0220	19.52 ↑	0.0223	17.86 ↑	0.0204	28.77 ↑	0.0279	0.0200	39.55 ↑	0.0212	31.96 ↑	0.0176	58.76 ↑	4.07 ±1.37
	CNN_{5-5}	10,680	0.1303	0.1155	12.81 ↑	0.1151	13.22 ↑	0.1088	19.73 ↑	0.0830	0.0714	16.23 ↑	0.0721	15.10 ↑	0.0636	30.53 ↑	2.91 ±0.32
	CNN_{3-2}	2,514	0.0790	0.0713	10.79 ↑	0.0713	10.74 ↑	0.0695	13.68 ↑	0.0497	0.0416	19.55 ↑	0.0418	19.06 ↑	0.0385	29.01 ↑	0.18 ±0.05
	CNN_{4-5}	8,680	0.0959	0.0868	10.40 ↑	0.0864	10.90 ↑	0.0840	14.19 ↑	0.0562	0.0486	15.51 ↑	0.0482	16.52 ↑	0.0453	24.03 ↑	1.19 ±0.20
	CNN_{3-4}	5,018	0.0747	0.0694	7.52 ↑	0.0693	7.72 ↑	0.0681	9.70 ↑	0.0465	0.0410	13.32 ↑	0.0409	13.59 ↑	0.0391	18.88 ↑	0.31 ±0.08
CIFAR10	9x100	910	0.0315	0.0211	49.03 ↑	0.0214	46.94 ↑	0.0192	63.84 ↑	0.0280	0.0183	52.70 ↑	0.0186	50.32 ↑	0.0132	111.50 ↑	4.92 ±0.05
	6x100	610	0.0222	0.0174	27.08 ↑	0.0176	26.14 ↑	0.0170	30.22 ↑	0.0165	0.0118	40.05 ↑	0.0120	37.82 ↑	0.0111	48.38 ↑	3.05 ±0.02
	5x100	510	0.0200	0.0167	19.76 ↑	0.0167	19.47 ↑	0.0163	22.47 ↑	0.0137	0.0104	31.80 ↑	0.0104	31.42 ↑	0.0099	38.59 ↑	2.44 ±0.01
	3x50	160	0.0206	0.0178	15.43 ↑	0.0179	14.66 ↑	0.0176	16.88 ↑	0.0144	0.0113	27.57 ↑	0.0115	25.24 ↑	0.0110	31.30 ↑	0.42 ±0.00
	4x100	410	0.0161	0.0140	15.23 ↑	0.0140	15.23 ↑	0.0138	16.56 ↑	0.0111	0.0089	24.61 ↑	0.0090	23.63 ↑	0.0087	27.76 ↑	1.84 ±0.01
	CNN_{3-4}	6,746	0.0187	0.0181	3.38 ↑	0.0181	3.32 ↑	0.0181	3.43 ↑	0.0109	0.0103	5.93 ↑	0.0103	5.83 ↑	0.0103	6.13 ↑	0.56 ±0.08
	CNN_{3-2}	3,378	0.0185	0.0180	2.49 ↑	0.0180	2.55 ↑	0.0180	2.67 ↑	0.0125	0.0120	4.34 ↑	0.0120	4.34 ↑	0.0120	4.60 ↑	0.31 ±0.06

Table 7: Comparison on non-negative Tanh networks.

Dataset	Model	#Neur.	Certified Lower Bound														Time (s)
			Average							Standard Deviation							
			NW	DC	Impr. (%)	VN	Impr. (%)	RV	Impr. (%)	NW	DC	Impr. (%)	VN	Impr. (%)	RV	Impr. (%)	
MNIST	5x100	510	0.0032	0.0007	366.18 ↑	0.0013	145.74 ↑	0.0010	207.77 ↑	0.0018	0.0003	436.36 ↑	0.0006	195.00 ↑	0.0006	216.07 ↑	4.28 ± 0.04
	3x700	2,110	0.0044	0.0011	288.50 ↑	0.0018	141.21 ↑	0.0016	174.38 ↑	0.0027	0.0006	380.70 ↑	0.0009	191.49 ↑	0.0009	191.49 ↑	117.13 ± 0.03
	3x400	1,210	0.0038	0.0012	222.03 ↑	0.0019	95.88 ↑	0.0017	130.30 ↑	0.0018	0.0005	291.11 ↑	0.0008	128.57 ↑	0.0007	137.84 ↑	39.17 ± 0.05
	CNN_{6-5}	12,300	0.0601	0.0226	166.37 ↑	0.0381	57.67 ↑	0.0303	98.61 ↑	0.0243	0.0066	269.35 ↑	0.0131	85.94 ↑	0.0087	179.13 ↑	5.68 ± 0.31
	3x50	160	0.0027	0.0018	49.45 ↑	0.0023	17.24 ↑	0.0022	25.35 ↑	0.0012	0.0007	67.12 ↑	0.0010	27.08 ↑	0.0008	45.24 ↑	0.14 ± 0.00
	3x100	310	0.0033	0.0023	42.74 ↑	0.0029	14.78 ↑	0.0028	21.45 ↑	0.0014	0.0009	56.98 ↑	0.0011	19.47 ↑	0.0010	33.66 ↑	2.16 ± 0.03
	3x200	610	0.0031	0.0022	42.13 ↑	0.0027	14.13 ↑	0.0026	18.99 ↑	0.0013	0.0008	59.52 ↑	0.0011	19.64 ↑	0.0010	31.37 ↑	8.02 ± 0.97
	CNN_{3-4}	5,018	0.0234	0.0169	38.45 ↑	0.0213	9.94 ↑	0.0209	12.21 ↑	0.0067	0.0041	64.69 ↑	0.0058	14.80 ↑	0.0055	20.83 ↑	0.32 ± 0.07
	CNN_{5-5}	10,680	0.0329	0.0246	33.62 ↑	0.0296	10.93 ↑	0.0290	13.42 ↑	0.0108	0.0069	56.67 ↑	0.0092	17.50 ↑	0.0087	24.54 ↑	2.88 ± 0.22
	CNN_{4-5}	8,680	0.0272	0.0209	30.22 ↑	0.0244	11.51 ↑	0.0239	14.03 ↑	0.0074	0.0050	49.80 ↑	0.0063	17.19 ↑	0.0060	23.63 ↑	1.17 ± 0.14
	CNN_{3-2}	2,514	0.0327	0.0254	28.46 ↑	0.0294	11.35 ↑	0.0288	13.63 ↑	0.0100	0.0067	48.89 ↑	0.0085	17.33 ↑	0.0081	23.10 ↑	0.17 ± 0.03
Fashion MNIST	3x100	310	0.0153	0.0085	80.66 ↑	0.0117	30.72 ↑	0.0104	48.02 ↑	0.0156	0.0077	104.05 ↑	0.0110	42.04 ↑	0.0091	70.79 ↑	2.17 ± 0.01
	2x200	410	0.0099	0.0065	52.62 ↑	0.0083	19.23 ↑	0.0079	25.57 ↑	0.0102	0.0058	76.91 ↑	0.0080	27.06 ↑	0.0076	33.55 ↑	4.93 ± 1.39
	CNN_{6-5}	12,300	0.0408	0.0269	51.63 ↑	0.0343	19.18 ↑	0.0297	37.67 ↑	0.0329	0.0180	82.76 ↑	0.0242	35.67 ↑	0.0180	82.66 ↑	5.81 ± 0.34
	2x100	210	0.0108	0.0072	49.24 ↑	0.0092	17.60 ↑	0.0086	25.85 ↑	0.0109	0.0060	81.56 ↑	0.0085	28.59 ↑	0.0077	41.95 ↑	1.10 ± 0.01
	CNN_{5-5}	10,680	0.0305	0.0205	48.34 ↑	0.0264	15.55 ↑	0.0239	27.28 ↑	0.0201	0.0117	71.83 ↑	0.0158	27.03 ↑	0.0125	60.69 ↑	2.92 ± 0.22
	CNN_{4-5}	8,680	0.0276	0.0190	45.57 ↑	0.0240	15.22 ↑	0.0223	24.07 ↑	0.0188	0.0097	95.03 ↑	0.0139	35.83 ↑	0.0129	45.82 ↑	1.21 ± 0.15
	CNN_{3-2}	2,514	0.0271	0.0188	43.65 ↑	0.0242	11.68 ↑	0.0234	15.45 ↑	0.0111	0.0068	65.04 ↑	0.0091	22.42 ↑	0.0085	31.37 ↑	0.18 ± 0.03
	CNN_{3-4}	5,018	0.0302	0.0216	39.92 ↑	0.0271	11.32 ↑	0.0263	14.88 ↑	0.0160	0.0106	51.09 ↑	0.0130	23.36 ↑	0.0121	32.45 ↑	0.31 ± 0.06
	3x200	610	0.0530	0.0389	35.98 ↑	0.0440	20.23 ↑	0.0354	49.70 ↑	0.0662	0.0536	23.48 ↑	0.0563	17.43 ↑	0.0454	45.67 ↑	7.93 ± 1.84
CIFAR10	3x200	610	0.0226	0.0079	185.50 ↑	0.0137	65.86 ↑	0.0120	88.82 ↑	0.0176	0.0048	271.37 ↑	0.0092	90.91 ↑	0.0080	119.40 ↑	6.37 ± 0.02
	4x100	410	0.0198	0.0071	178.65 ↑	0.0106	88.06 ↑	0.0091	118.98 ↑	0.0147	0.0078	88.79 ↑	0.0101	45.63 ↑	0.0097	51.19 ↑	1.85 ± 0.01
	3x700	2,110	0.0688	0.0287	140.06 ↑	0.0391	76.09 ↑	0.0385	78.93 ↑	0.0512	0.0363	41.08 ↑	0.0386	32.64 ↑	0.0406	26.33 ↑	102.74 ± 0.73
	3x400	1,210	0.0484	0.0211	129.25 ↑	0.0295	64.15 ↑	0.0257	88.85 ↑	0.0441	0.0233	89.15 ↑	0.0291	51.58 ↑	0.0253	74.69 ↑	35.51 ± 0.15
	3x100	310	0.0429	0.0208	106.35 ↑	0.0295	45.33 ↑	0.0269	59.42 ↑	0.0435	0.0227	91.80 ↑	0.0270	61.35 ↑	0.0256	69.66 ↑	1.26 ± 0.01
	5x100	510	0.0462	0.0272	69.75 ↑	0.0315	46.45 ↑	0.0300	54.12 ↑	0.0400	0.0371	7.84 ↑	0.0378	5.96 ↑	0.0378	5.87 ↑	2.46 ± 0.02
	3x50	160	0.0170	0.0102	67.39 ↑	0.0138	23.30 ↑	0.0134	27.26 ↑	0.0111	0.0048	129.96 ↑	0.0077	44.73 ↑	0.0071	56.32 ↑	0.42 ± 0.00
	CNN_{3-2}	3,378	0.0136	0.0130	5.17 ↑	0.0134	1.49 ↑	0.0134	1.56 ↑	0.0106	0.0094	12.95 ↑	0.0102	4.01 ↑	0.0102	4.21 ↑	0.30 ± 0.04
	CNN_{3-4}	6,746	0.0102	0.0098	3.46 ↑	0.0101	0.99 ↑	0.0101	0.99 ↑	0.0062	0.0058	7.04 ↑	0.0061	1.80 ↑	0.0061	1.96 ↑	0.57 ± 0.08
	CNN_{3-5}	8,430	0.0094	0.0091	3.30 ↑	0.0093	0.97 ↑	0.0093	0.97 ↑	0.0066	0.0062	6.47 ↑	0.0065	1.86 ↑	0.0065	2.02 ↑	0.73 ± 0.10

Table 8: Comparison on non-negative Arctan networks.

Dataset	Model	#Neur.	Certified Lower Bound														Time (s)
			Average							Standard Deviation							
			NW	DC	Impr. (%)	VN	Impr. (%)	RV	Impr. (%)	NW	DC	Impr. (%)	VN	Impr. (%)	RV	Impr. (%)	
MNIST	CNN_{6-5}	12,300	0.0484	0.0227	113.44 ↑	0.0367	32.11 ↑	0.0325	49.15 ↑	0.0189	0.0061	210.16 ↑	0.0124	52.21 ↑	0.0098	92.67 ↑	5.64 ±0.30
	5x100	510	0.0020	0.0010	99.01 ↑	0.0016	29.68 ↑	0.0014	41.55 ↑	0.0012	0.0010	20.79 ↑	0.0008	48.78 ↑	0.0007	67.12 ↑	4.44 ±0.05
	3x700	2,110	0.0023	0.0013	70.45 ↑	0.0019	20.32 ↑	0.0017	29.31 ↑	0.0012	0.0006	98.31 ↑	0.0009	28.57 ↑	0.0008	42.68 ↑	116.93 ±0.28
	3x50	160	0.0051	0.0032	58.57 ↑	0.0043	18.10 ↑	0.0040	26.62 ↑	0.0026	0.0014	90.58 ↑	0.0020	32.83 ↑	0.0017	52.91 ↑	0.14 ±0.00
	3x400	1,210	0.0024	0.0015	58.39 ↑	0.0020	17.41 ↑	0.0019	24.21 ↑	0.0011	0.0006	92.98 ↑	0.0009	26.44 ↑	0.0008	37.50 ↑	39.22 ±0.02
	3x200	610	0.0028	0.0018	55.80 ↑	0.0024	16.53 ↑	0.0023	22.08 ↑	0.0012	0.0006	95.24 ↑	0.0010	28.13 ↑	0.0009	41.38 ↑	9.78 ±2.25
	3x100	310	0.0031	0.0020	50.98 ↑	0.0027	14.93 ↑	0.0026	19.84 ↑	0.0013	0.0007	76.06 ↑	0.0010	22.55 ↑	0.0010	31.58 ↑	2.18 ±0.02
	CNN_{5-5}	10,680	0.0218	0.0154	41.93 ↑	0.0196	11.22 ↑	0.0191	14.02 ↑	0.0070	0.0039	76.84 ↑	0.0059	18.80 ↑	0.0055	26.59 ↑	2.84 ±0.20
	CNN_{3-2}	2,514	0.0138	0.0100	38.99 ↑	0.0127	8.90 ↑	0.0125	11.08 ↑	0.0042	0.0026	64.20 ↑	0.0037	12.83 ↑	0.0036	17.88 ↑	0.17 ±0.03
	CNN_{4-5}	8,680	0.0203	0.0151	34.37 ↑	0.0186	9.05 ↑	0.0182	11.20 ↑	0.0068	0.0042	62.47 ↑	0.0059	15.15 ↑	0.0057	20.85 ↑	1.16 ±0.13
	CNN_{3-4}	5,018	0.0162	0.0127	27.46 ↑	0.0152	6.51 ↑	0.0151	7.57 ↑	0.0053	0.0036	46.65 ↑	0.0048	9.83 ↑	0.0047	12.66 ↑	0.30 ±0.05
Fashion MNIST	4x100	410	0.0179	0.0066	172.71 ↑	0.0108	66.26 ↑	0.0092	94.88 ↑	0.0279	0.0093	200.32 ↑	0.0150	85.31 ↑	0.0134	107.68 ↑	3.28 ±0.03
	3x100	310	0.0120	0.0062	92.91 ↑	0.0091	31.07 ↑	0.0081	47.17 ↑	0.0127	0.0057	124.07 ↑	0.0089	42.41 ↑	0.0078	63.14 ↑	2.17 ±0.02
	3x200	610	0.0097	0.0053	83.46 ↑	0.0077	25.49 ↑	0.0071	36.49 ↑	0.0107	0.0052	106.78 ↑	0.0081	31.89 ↑	0.0070	52.43 ↑	9.85 ±2.64
	2x100	210	0.0105	0.0068	55.54 ↑	0.0091	16.35 ↑	0.0086	23.01 ↑	0.0095	0.0045	109.27 ↑	0.0074	28.11 ↑	0.0069	37.99 ↑	1.11 ±0.00
	CNN_{6-5}	12,300	0.0324	0.0214	51.87 ↑	0.0282	14.95 ↑	0.0258	25.88 ↑	0.0228	0.0132	72.69 ↑	0.0180	27.19 ↑	0.0134	69.87 ↑	5.74 ±0.33
	2x200	410	0.0087	0.0058	49.14 ↑	0.0076	13.97 ↑	0.0073	18.82 ↑	0.0079	0.0044	81.61 ↑	0.0065	20.98 ↑	0.0061	29.08 ↑	4.53 ±0.90
	CNN_{3-2}	2,514	0.0266	0.0180	47.64 ↑	0.0235	13.34 ↑	0.0225	17.97 ↑	0.0179	0.0089	101.01 ↑	0.0127	41.29 ↑	0.0115	55.37 ↑	0.17 ±0.03
	CNN_{4-5}	8,680	0.0241	0.0168	43.44 ↑	0.0217	10.82 ↑	0.0207	16.06 ↑	0.0117	0.0075	55.51 ↑	0.0099	17.93 ↑	0.0090	30.40 ↑	1.18 ±0.14
	CNN_{5-5}	10,680	0.0266	0.0189	40.75 ↑	0.0241	10.50 ↑	0.0228	16.95 ↑	0.0121	0.0083	47.15 ↑	0.0107	13.88 ↑	0.0099	22.75 ↑	2.88 ±0.22
	CNN_{3-4}	5,018	0.0258	0.0196	31.65 ↑	0.0237	8.64 ↑	0.0231	11.60 ↑	0.0158	0.0103	53.54 ↑	0.0135	17.61 ↑	0.0124	27.97 ↑	0.30 ±0.06
CIFAR10	4x100	410	0.0173	0.0130	33.67 ↑	0.0158	9.70 ↑	0.0156	11.25 ↑	0.0128	0.0082	56.37 ↑	0.0110	16.11 ↑	0.0107	19.59 ↑	1.85 ±0.02
	5x100	510	0.0248	0.0191	30.05 ↑	0.0229	8.28 ↑	0.0227	9.38 ↑	0.0195	0.0139	40.20 ↑	0.0176	11.16 ↑	0.0173	13.09 ↑	2.44 ±0.03
	3x50	160	0.0186	0.0144	29.16 ↑	0.0171	8.41 ↑	0.0169	9.82 ↑	0.0112	0.0077	45.60 ↑	0.0099	13.42 ↑	0.0096	17.45 ↑	0.42 ±0.00
	6x100	610	0.0171	0.0133	28.61 ↑	0.0159	7.69 ↑	0.0158	8.44 ↑	0.0157	0.0115	36.79 ↑	0.0144	9.19 ↑	0.0142	10.42 ↑	3.04 ±0.02
	3x100	310	0.0138	0.0107	28.21 ↑	0.0127	8.17 ↑	0.0126	9.29 ↑	0.0113	0.0077	46.38 ↑	0.0100	13.30 ↑	0.0098	15.85 ↑	1.25 ±0.01
	3x200	610	0.0108	0.0088	22.30 ↑	0.0101	6.23 ↑	0.0101	6.86 ↑	0.0073	0.0054	36.26 ↑	0.0066	10.12 ↑	0.0065	11.47 ↑	6.48 ±0.02
	3x700	2,110	0.0079	0.0066	21.07 ↑	0.0075	5.73 ↑	0.0074	6.59 ↑	0.0053	0.0040	33.08 ↑	0.0049	9.03 ↑	0.0048	10.63 ↑	105.61 ±2.25
	3x400	1,210	0.0105	0.0087	20.83 ↑	0.0099	5.74 ↑	0.0099	6.38 ↑	0.0068	0.0051	34.45 ↑	0.0062	9.46 ↑	0.0062	10.88 ↑	37.78 ±0.14
	CNN_{3-2}	3,378	0.0115	0.0111	3.24 ↑	0.0114	0.88 ↑	0.0114	0.88 ↑	0.0078	0.0074	5.80 ↑	0.0077	1.55 ↑	0.0077	1.69 ↑	0.30 ±0.04
	CNN_{3-4}	6,746	0.0080	0.0077	3.10 ↑	0.0079	0.88 ↑	0.0079	0.88 ↑	0.0051	0.0048	5.63 ↑	0.0050	1.40 ↑	0.0050	1.60 ↑	0.56 ±0.08
	CNN_{3-5}	8,430	0.0095	0.0093	2.49 ↑	0.0094	0.64 ↑	0.0094	0.64 ↑	0.0061	0.0058	4.49 ↑	0.0060	1.17 ↑	0.0060	1.34 ↑	0.72 ±0.10

Table 9: Performance comparison of Alg. 1 with DeepCert (DC), VeriNet (VN), and RobustVerifier (RV) on 1-hidden-layer Sigmoid and Tanh networks with mixed weights.

Arch.	Model	σ	Certified Lower Bound														Time (s)	
			Average							Standard Deviation							Alg.1	Others
			Alg.1	DC	Impr. (%)	VN	Impr. (%)	RV	Impr. (%)	Alg.1	DC	Impr. (%)	VN	Impr. (%)	RV	Impr. (%)		
CNN	CNN_{2-1-5}^{*}	Sig.	0.0727	0.0437	66.25 ↑	0.0435	67.20 ↑	0.0428	69.86 ↑	0.0312	0.0145	114.58 ↑	0.0143	118.64 ↑	0.0137	127.90 ↑	0.22	0.06 ±0.01
		Tanh	0.0616	0.0270	127.77 ↑	0.0285	116.10 ↑	0.0277	122.66 ↑	0.0313	0.0124	151.99 ↑	0.0132	137.46 ↑	0.0121	158.01 ↑	0.21	0.07 ±0.01
	CNN_{2-2-5}^{*}	Sig.	0.0726	0.0542	34.03 ↑	0.0539	34.70 ↑	0.0521	39.38 ↑	0.0321	0.0208	54.12 ↑	0.0207	55.32 ↑	0.0187	71.42 ↑	0.60	0.08 ±0.01
		Tanh	0.0477	0.0315	51.44 ↑	0.0331	44.17 ↑	0.0321	48.89 ↑	0.0268	0.0113	137.60 ↑	0.0122	120.24 ↑	0.0113	138.03 ↑	0.58	0.08 ±0.01
	CNN_{2-3-5}^{*}	Sig.	0.0707	0.0526	34.51 ↑	0.0524	34.98 ↑	0.0503	40.67 ↑	0.0308	0.0197	56.16 ↑	0.0196	56.64 ↑	0.0176	74.99 ↑	1.03	0.09 ±0.02
		Tanh	0.0486	0.0313	55.27 ↑	0.0327	48.80 ↑	0.0317	53.41 ↑	0.0196	0.0113	74.48 ↑	0.0121	62.90 ↑	0.0111	76.68 ↑	0.80	0.10 ±0.01
	CNN_{2-4-5}^{*}	Sig.	0.0786	0.0647	21.45 ↑	0.0645	21.80 ↑	0.0621	26.63 ↑	0.0489	0.0233	109.96 ↑	0.0232	110.96 ↑	0.0210	132.97 ↑	0.71	0.11 ±0.02
		Tanh	0.0600	0.0262	129.25 ↑	0.0272	120.49 ↑	0.0266	125.71 ↑	0.0222	0.0095	134.51 ↑	0.0103	116.68 ↑	0.0096	132.31 ↑	2.73	0.11 ±0.02
	CNN_{2-5-3}^{*}	Sig.	0.0765	0.0477	60.33 ↑	0.0476	60.67 ↑	0.0471	62.58 ↑	0.0408	0.0182	123.79 ↑	0.0182	124.65 ↑	0.0176	131.93 ↑	2.54	0.09 ±0.03
		Tanh	0.0273	0.0189	44.68 ↑	0.0192	42.34 ↑	0.0191	43.39 ↑	0.0120	0.0071	69.37 ↑	0.0072	65.62 ↑	0.0071	69.13 ↑	0.66	0.09 ±0.02
	CNN_{2-1-5}^{+}	Sig.	0.1292	0.0877	47.35 ↑	0.0876	47.47 ↑	0.0835	54.75 ↑	0.0484	0.0385	25.71 ↑	0.0386	25.38 ↑	0.0349	38.80 ↑	1.34	0.06 ±0.01
		Tanh	0.0676	0.0490	38.08 ↑	0.0554	21.96 ↑	0.0525	28.67 ↑	0.0352	0.0211	67.26 ↑	0.0254	38.73 ↑	0.0227	55.31 ↑	0.20	0.07 ±0.01
	CNN_{2-2-5}^{+}	Sig.	0.1241	0.0823	50.82 ↑	0.0822	50.90 ↑	0.0777	59.74 ↑	0.0489	0.0353	38.39 ↑	0.0355	37.65 ↑	0.0311	57.12 ↑	0.56	0.08 ±0.02
		Tanh	0.0751	0.0482	55.93 ↑	0.0535	40.40 ↑	0.0509	47.69 ↑	0.0388	0.0214	81.81 ↑	0.0260	49.25 ↑	0.0234	66.32 ↑	0.59	0.08 ±0.01
	CNN_{2-3-5}^{+}	Sig.	0.1049	0.0844	24.38 ↑	0.0843	24.39 ↑	0.0811	29.40 ↑	0.0511	0.0371	37.55 ↑	0.0374	36.55 ↑	0.0344	48.46 ↑	0.78	0.09 ±0.02
		Tanh	0.0574	0.0465	23.67 ↑	0.0506	13.59 ↑	0.0479	20.05 ↑	0.0509	0.0214	138.09 ↑	0.0245	108.20 ↑	0.0215	136.76 ↑	0.43	0.10 ±0.01
	CNN_{2-4-5}^{+}	Sig.	0.1304	0.0814	60.09 ↑	0.0814	60.15 ↑	0.0784	66.32 ↑	0.0546	0.0366	49.19 ↑	0.0367	48.71 ↑	0.0337	62.23 ↑	3.23	0.11 ±0.02
		Tanh	0.0794	0.0475	67.24 ↑	0.0518	53.29 ↑	0.0494	60.64 ↑	0.0605	0.0213	184.09 ↑	0.0246	145.97 ↑	0.0219	176.05 ↑	0.49	0.11 ±0.02
	CNN_{2-5-3}^{+}	Sig.	0.0788	0.0700	12.45 ↑	0.0699	12.61 ↑	0.0678	16.15 ↑	0.0378	0.0340	11.28 ↑	0.0340	11.12 ↑	0.0313	20.97 ↑	0.78	0.10 ±0.03
		Tanh	0.0416	0.0322	29.32 ↑	0.0335	24.11 ↑	0.0325	28.16 ↑	0.0184	0.0169	8.39 ↑	0.0184	0.00 ↑	0.0167	10.21 ↑	0.65	0.10 ±0.02
FNN	$1x50^{*}$	Sig.	0.0367	0.0275	33.46 ↑	0.0279	31.73 ↑	0.0235	56.15 ↑	0.0152	0.0082	85.10 ↑	0.0085	78.56 ↑	0.0062	143.24 ↑	0.15	0.03 ±0.00
		Tanh	0.0271	0.0171	58.47 ↑	0.0199	36.31 ↑	0.0167	62.56 ↑	0.0086	0.0051	70.36 ↑	0.0058	49.43 ↑	0.0042	105.65 ↑	0.10	0.03 ±0.00
	$1x100^{*}$	Sig.	0.0362	0.0247	46.61 ↑	0.0250	45.02 ↑	0.0215	68.48 ↑	0.0119	0.0064	86.44 ↑	0.0066	79.14 ↑	0.0050	138.86 ↑	0.17	0.04 ±0.00
		Tanh	0.0233	0.0151	54.21 ↑	0.0179	29.91 ↑	0.0152	52.99 ↑	0.0080	0.0042	90.51 ↑	0.0050	58.00 ↑	0.0037	114.64 ↑	0.16	0.04 ±0.00
	$1x150^{*}$	Sig.	0.0433	0.0290	49.42 ↑	0.0292	48.39 ↑	0.0254	70.28 ↑	0.0125	0.0083	50.87 ↑	0.0085	46.27 ↑	0.0064	94.35 ↑	0.24	0.06 ±0.00
		Tanh	0.0244	0.0169	44.89 ↑	0.0198	23.20 ↑	0.0171	43.19 ↑	0.0072	0.0046	54.80 ↑	0.0056	27.98 ↑	0.0042	69.84 ↑	0.48	0.06 ±0.00
	$1x200^{*}$	Sig.	0.0403	0.0260	54.74 ↑	0.0262	53.97 ↑	0.0230	74.75 ↑	0.0122	0.0074	64.86 ↑	0.0076	61.38 ↑	0.0058	111.44 ↑	0.32	0.07 ±0.01
		Tanh	0.0277	0.0154	79.61 ↑	0.0184	50.76 ↑	0.0162	71.49 ↑	0.0078	0.0034	128.78 ↑	0.0045	72.34 ↑	0.0035	123.55 ↑	0.82	0.07 ±0.00
	$1x250^{*}$	Sig.	0.0355	0.0256	38.69 ↑	0.0257	38.21 ↑	0.0229	54.86 ↑	0.0091	0.0075	22.15 ↑	0.0076	20.21 ↑	0.0060	52.43 ↑	0.22	0.09 ±0.00
		Tanh	0.0235	0.0141	65.91 ↑	0.0166	41.07 ↑	0.0147	59.48 ↑	0.0080	0.0032	151.73 ↑	0.0041	97.04 ↑	0.0031	154.14 ↑	1.01	0.09 ±0.00
	$1x50^{+}$	Sig.	0.0484	0.0351	37.72 ↑	0.0356	35.83 ↑	0.0292	65.48 ↑	0.0147	0.0117	25.54 ↑	0.0122	20.90 ↑	0.0089	65.41 ↑	0.13	0.03 ±0.00
		Tanh	0.0321	0.0206	55.78 ↑	0.0237	35.35 ↑	0.0193	66.53 ↑	0.0139	0.0067	106.93 ↑	0.0081	71.93 ↑	0.0057	145.18 ↑	0.23	0.03 ±0.00
	$1x100^{+}$	Sig.	0.0434	0.0331	31.11 ↑	0.0334	29.78 ↑	0.0276	57.33 ↑	0.0167	0.0119	39.65 ↑	0.0123	36.12 ↑	0.0088	90.13 ↑	0.20	0.04 ±0.00
		Tanh	0.0277	0.0185	49.70 ↑	0.0213	29.84 ↑	0.0177	56.73 ↑	0.0137	0.0059	132.27 ↑	0.0068	101.73 ↑	0.0048	182.76 ↑	0.18	0.04 ±0.00
	$1x150^{+}$	Sig.	0.0463	0.0375	23.59 ↑	0.0377	22.77 ↑	0.0317	46.12 ↑	0.0171	0.0120	42.52 ↑	0.0123	39.27 ↑	0.0090	89.87 ↑	0.21	0.06 ±0.00
		Tanh	0.0320	0.0208	53.65 ↑	0.0239	33.72 ↑	0.0198	61.65 ↑	0.0128	0.0068	88.96 ↑	0.0079	62.96 ↑	0.0056	128.44 ↑	0.19	0.06 ±0.00
	$1x200^{+}$	Sig.	0.0445	0.0378	17.88 ↑	0.0380	17.07 ↑	0.0318	39.92 ↑	0.0207	0.0129	60.06 ↑	0.0132	56.55 ↑	0.0096	115.41 ↑	1.02	0.07 ±0.00
		Tanh	0.0301	0.0199	51.07 ↑	0.0229	31.37 ↑	0.0192	56.74 ↑	0.0120	0.0065	84.27 ↑	0.0075	58.61 ↑	0.0056	115.49 ↑	1.01	0.07 ±0.00
	$1x250^{+}$	Sig.	0.0449	0.0385	16.61 ↑	0.0387	16.09 ↑	0.0327	37.46 ↑	0.0192	0.0126	52.45 ↑	0.0128	50.42 ↑	0.0096	101.24 ↑	1.15	0.09 ±0.00
		Tanh	0.0312	0.0196	59.31 ↑	0.0226	38.48 ↑	0.0191	63.82 ↑	0.0109	0.0059	86.00 ↑	0.0067	62.16 ↑	0.0049	123.88 ↑	1.15	0.09 ±0.00