Tightening Robustness Verification of MaxPool-based Neural Networks via Minimizing the Over-Approximation Zone

Yuan Xiao[1,3], Yuchen Chen[1], Shiqing Ma[2], Chunrong Fang[1,3*], Tongtong Bai[1], Mingzheng Gu[1], Yuxin Cheng[1], Yanwei Chen[1], Zhenyu Chen[1,3*]
[1] State Key Laboratory for Novel Software Technology, Nanjing University, China
[2] University of Massachusetts Amherst, United States
[3] Shenzhen Research Institute, Nanjing University, China

Abstract

The robustness of neural network classifiers is important in the safety-critical domain and can be quantified by robustness verification. At present, efficient and scalable verification techniques are always sound but incomplete, and thus, the improvement of verified robustness results is the key criterion to evaluate the performance of incomplete verification approaches. The multi-variate function MaxPool is widely adopted yet challenging to verify. In this paper, we present ***Ti-Lin****, a robustness verifier for MaxPool-based CNNs with* ***Ti****ght* ***Lin****ear Approximation. Following the sequel of minimizing the over-approximation zone of the non-linear function of CNNs, we are the first to propose the provably neuron-wise tightest linear bounds for the MaxPool function. By our proposed linear bounds, we can certify larger robustness results for CNNs. We evaluate the effectiveness of Ti-Lin on different verification frameworks with open-sourced benchmarks, including LeNet, PointNet, and networks trained on the MNIST, CIFAR-10, Tiny ImageNet and ModelNet40 datasets. Experimental results show that Ti-Lin significantly outperforms the state-of-the-art methods across all networks with up to 78.6% improvement in terms of the certified accuracy with almost the same time consumption as the fastest tool. Our code is available at* `https://github.com/xiaoyuanpigo/Ti-Lin-Hybrid-Lin`.

1. Introduction

Although neural networks achieve remarkable success in many difficult classification tasks, researchers discover non-robust neural networks are vulnerable to perturbation from the environment and adversarial attacks [28, 43, 55, 73]. In some safety-critical domains, such as autonomous driving [22] and face recognition [23], some subtle adversarial perturbation is extremely imperceptible and harmful and may cause disastrous consequences. Therefore, there is an urgent need to certify model robustness guarantees against adversarial attacks, which can provide certified defense against any possible attacks [5, 14, 51]. Mathematically, verifiers aim to prove or disprove the correctness of the output neurons in response to input values with minor perturbations.

The methodology of robustness verification can be divided into complete verifiers and incomplete verifiers. As complete verification is NP-complete even for the simple ReLU-based fully-connected networks [31], it is necessary to risk some verification precision loss to improve efficiency [16, 29, 60]. State-of-the-art verifiers use advanced branching [17, 45, 59] and relaxation techniques [27, 53, 57, 58] to reduce the precision loss while maintaining efficiency and scalability. Linear approximation is an efficient relaxation technique that can be used to handle complex functions beyond univariate functions, such as $Sigmoid(x)Tanh(y), x \cdot Sigmoid(y)$ [14, 33, 47] and MaxPool [5, 26, 38, 53]. Furthermore, the flexibility of this technique allows it to be combined with other incomplete or complete verification algorithms [49, 59, 67]. Linear approximation, in essence, provides a pair of upper and lower linear constraints $u(\boldsymbol{x}), l(\boldsymbol{x})$ to bound the output of the network's non-linear layer $f(\boldsymbol{x})$ according to the layer's input region $\{\boldsymbol{x}|\boldsymbol{x} \in [\boldsymbol{l}, \boldsymbol{u}]\}$. Considerable efforts have been devoted to tightening linear approximation for various non-linear functions [27, 38, 39, 47, 63, 67], which have proven successful in certifying more precise results. Some of these works propose the linear bounds that can produce the smallest over-approximation zone, a.k.a., the neuron-wise tightest linear bounds [72], for the S-shaped [27], $Sigmoid(x) \cdot Tanh(y), x \cdot Sigmoid(y)$ [33, 47], and ReLU functions [5, 53]. These neuron-wise tightest linear bounds can empirically give rise to certified robustness results and stand for the highest precision among other relevant

*Chunrong Fang and Zhenyu Chen are the corresponding authors.

work [42]. However, they focus on the ReLU and Sigmoid layer and ignore multi-variate functions like MaxPool, which is widely adopted in CNNs [37, 66] yet is far more complex to verify.

Until recently, some works attempt to make non-trivial linear relaxation for MaxPool [5, 38, 53] or transform MaxPool into a sequence of affine and ReLU layers to verify [26]. Unfortunately, they both certify robustness results coarsely and some consume much time to gain non-optimal results [26, 38]. These works fail to capture the neuron-wise tightest linear bounds for MaxPool, leaving ample room for the linear bounds to be further adapted to tighten. Recently, a block-wise tightest linear upper bound was proposed for MaxPool [65], which minimizes the over-approximation for the ReLU+MaxPool block. However, this upper bound is the block-wise tightest only when the verifier uses a single upper linear constraint to bound the ReLU. When ReLU is bounded by BaB techniques and has no precision loss, the neuron-wise linear bounds achieve higher precision than the block-wise tightest linear upper bound [65] for the ReLU+MaxPool block.

To address the above challenges, we propose Ti-Lin, a robustness verifier for MaxPool-based CNNs with tight linear approximation. To the best of our knowledge, we are the first to propose the neuron-wise tightest linear bounds for MaxPool. Specifically, we introduce the notion of neuron-wise tightest and utilize the characteristic of piece-wise linearity of MaxPool $f(\boldsymbol{x})$. We make the upper linear bound $u(\boldsymbol{x})$ passing two points $f(\boldsymbol{a})$ and $f(\boldsymbol{b})$, where $\boldsymbol{a}$ and $\boldsymbol{b}$ are the endpoints of the space diagonal for MaxPool's input region, and the lower linear bound passing $f(\boldsymbol{m})$. A simple example are shown in Figure 1, where $\boldsymbol{a} = (l_1, u_2)$ and $\boldsymbol{b} = (u_1, l_2)$. We prove that our proposed linear approximation is the neuron-wise tightest and its computation is efficient with the time complexity $\mathcal{O}(pool_size \times log3)$, where $pool_size$ is the size of inputs to be pooled. Moreover, Ti-Lin can be easily integrated into different state-of-the-art verifiers, such as ERAN [38, 45, 48, 51, 53, 54] and α, β-CROWN [56, 59, 67, 69–71], the VNN-COMP 2021-2023 winner [3, 6, 44]. The integration allows Ti-Lin to certify different types of MaxPool-based networks (e.g., CNNs and PointNets) against l_1, l_2, l_∞-norm perturbations.

To illustrate the superiority of Ti-Lin, we integrate Ti-Lin into four verification frameworks, including CNN-Cert [5], DeepPoly [53], ERAN, and α, β-CROWN, and evaluate Ti-Lin with open-sourced benchmarks on the MNIST [35], CIFAR-10 [34], Tiny ImageNet [34], and ModelNet40 [64] datasets. The experiment results show that Ti-Lin outperforms the state-of-the-art verifiers for MaxPool-based networks, including CNN-Cert [5], DeepPoly [53], and 3DCertify [38], MN-BaB, ERAN, and α, β-CROWN, MaxLin [65], OSIP [26], with up to 69.0%, 39.4%, 7.1%, 39.4%, 7.1%, 78.6%, 28.6% and 65.0% improvement in terms of tightness, respectively. Meanwhile, having almost the same time cost as the fastest tool, Ti-Lin incurs almost no additional time cost and has up to 5.3×, 1.7×, and 420.9× speedup over 3DCertify, ERAN, and OSIP, respectively.

In summary, our work proposes a neuron-wise tightest linear approximation for MaxPool, Ti-Lin, with high time efficiency, which works for various CNNs and different verification frameworks. By tightening linear approximation for MaxPool, our approach outperforms the state-of-the-art tools with up to 78.6% improvement to the certified results and has the same time efficiency as the fastest baseline method.

2. Related work

Adversarial attacks and defenses Many research studies have shown that machine learning models are vulnerable to adversarial examples [9–11, 30, 36, 50, 62, 68], which pose severe concerns for their deployment in security and safety-critical applications such as autonomous driving. With the existence of adversarial attacks, a plethora of research on adversarial defense has been created [4, 12, 20, 27, 31, 32, 39, 41, 59, 63, 67]. However, empirical defenses [7, 46] cannot provide a formal robustness guarantee and are often compromised by adaptive, unforeseen attacks [2, 8, 19]. Therefore, our work focuses on certified defense which can provide a provably safe guarantee, and targets MaxPool-based convolutional neural networks, which are widely used for image classification.

Robustness verification for MaxPool-based neural networks Recently, Some work based on mixed integer linear programming and satisfiability modulo theory [1, 15, 32] proposed a complete verification framework that supports CNNs with arbitrary piece-wise linear activation functions. However, it cannot apply to MaxPool-based networks with other activation functions, such as Sigmoid. Recently, PRIMA [45] proposes a novel verifier based on multi-neuron relaxation for arbitrary, bounded, multivariate non-linear activation functions. MN-BaB [17] propose a state-of-the-art complete neural network verifier that builds on the tight multi-neuron constraints proposed in PRIMA. Another line of research based on linear approximation has the advantage of high time efficiency and, meanwhile, preserves high precision of certifying the robustness of CNNs. DeepPoly [53] gives non-trivial linear bounds to MaxPool and its parallel work, CNN-Cert [5] leverages a hyperplane containing n different particular points to give the upper and lower bound. 3DCertify [38] gives tighter bounds than DeepPoly by the double description method [18]. Recently, OSIP [26] verifies MaxPool-based networks by transforming the MaxPool layer into a sequence of affine and ReLU

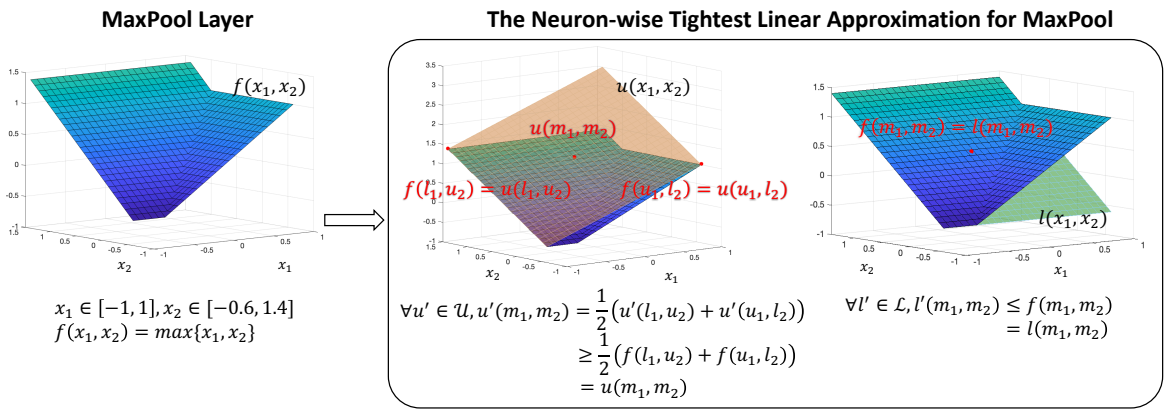


Figure 1. An illustration for the neuron-wise tightest linear bounds for bivariate MaxPool function. From left to right, the subfigures are the MaxPool function, upper linear bound plane $u(x_1, x_2)$, and lower linear bound plane $l(x_1, x_2)$, respectively. The red dots are $u(l_1, u_2), u(u_1, l_2), u(m_1, m_2)$,and $l(m_1, m_2)$, where $m_i = \frac{1}{2}(l_i + u_i), i = 1, 2$. $\mathcal{L}$ and $\mathcal{U}$ are the sets of all lower and upper linear bounds for MaxPool, respectively.

layers. However, the above techniques verify the robustness results coarsely and some methods waste much time to gain sub-optimal results. Recently, MaxLin [65] proposed provably block-wise tightest upper linear constraints for ReLU+MaxPool blocks. However, this block-wise tightest property holds only when the ReLU is bounded by a single upper linear constraint, rather than by much tighter constraints, such as those provided by BaB techniques.

3. Preliminaries

3.1. Certified robustness bound

Let $F(\boldsymbol{x}) : \mathbb{R}^{n_0} \rightarrow \mathbb{R}^{n_K}$ be a neural network classifier function with (K+1) layers and $\boldsymbol{x_0}$ be an input data point. t denotes the true label of $\boldsymbol{x_0}$ in F. Let $\mathbb{B}_p(\boldsymbol{x_0}, \epsilon)$ denotes $\boldsymbol{x_0}$ perturbed within an l_p-norm ball with radius ϵ, that is $\mathbb{B}_p(\boldsymbol{x_0}, \epsilon) = \{\boldsymbol{x} | \|\boldsymbol{x} - \boldsymbol{x_0}\|_p \leq \epsilon\}$. In this work, $p = 1, 2, \infty$. Next, we introduce the notion of local robustness bound ϵ_r, which can be certified by complete verifiers.

Definition 1 (Local robustness bound)**.** *If $\epsilon_r (\epsilon_r \geq 0)$ is the local robustness bound of an input $\boldsymbol{x_0}$ in the neural network, if and only if these two conditions (i) $argmax_i F_i(\boldsymbol{x}) = t, \forall \boldsymbol{x} \in \mathbb{B}_p(\boldsymbol{x_0}, \epsilon_r)$ and (ii) $\forall \delta > 0, \exists \boldsymbol{x_a} \in \mathbb{B}_p(\boldsymbol{x_0}, \epsilon + \delta) s.t. \arg\max_i F_i(\boldsymbol{x_a}) \neq t$ are satisfied.*

Apparently, ϵ_r is the maximum absolute safe radius of $\boldsymbol{x_0}$. Although computing ϵ_r is an essential problem, it is computationally expensive and NP-complete [31]. Thus, it is practical to compute a bound that satisfies condition (ii) of Definition 1 but is lower than ϵ_r.

Definition 2 (Certified lower bound)**.** *If $\epsilon_{cert} (\epsilon_{cert} \geq 0)$ is a certified lower bound of input $\boldsymbol{x_0}$ in a neural network, if and only if these two conditions (i) $\epsilon_{cert} \leq \epsilon_r$ and (ii) $\arg\max_i F_i(\boldsymbol{x}) = t, \forall \boldsymbol{x} \in \mathbb{B}_p(\boldsymbol{x_0}, \epsilon_r)$ are satisfied.*

Robustness lower bound can be certified by incomplete verifiers. As incomplete verifiers risk precision loss to efficiently certify the robustness of more types of networks, the value of ϵ_{cert} is a key criterion to evaluate the tightness of robustness verification methods.

3.2. Linear approximation

Define $\boldsymbol{l}, \boldsymbol{u}$ are the lower and upper bounds of the input of the intermediate layer, whose function is $f(\cdot)$; that is, $\boldsymbol{x} \in [\boldsymbol{l}, \boldsymbol{u}]$. The essence of the linear approximation technique is giving linear bounds to every layer, that is

$$\forall \boldsymbol{x} \in [\boldsymbol{l}, \boldsymbol{u}], l(\boldsymbol{x}) \leq f(\boldsymbol{x}) \leq u(\boldsymbol{x}) \tag{1}$$

Definition 3 (Upper/Lower linear bound)**.** *Let $f_i(\boldsymbol{x})$ be the function of the i-th neuron in the intermediate layer $f(\boldsymbol{x})$ of neural network F, with $\boldsymbol{x} \in [\boldsymbol{l}, \boldsymbol{u}] \subset \mathbb{R}^n, \boldsymbol{a_u}, \boldsymbol{a_l} \in \mathbb{R}^n, b_u, b_l \in \mathbb{R}$ and*

$$u_i(\boldsymbol{x}) = \boldsymbol{a_u}\boldsymbol{x} + b_u, l_i(\boldsymbol{x}) = \boldsymbol{a_l}\boldsymbol{x} + b_l$$

$u_i(\boldsymbol{x})$ and $l_i(\boldsymbol{x})$ are linear upper and lower bounds of $f_i(x)$ if $l_i(\boldsymbol{x}) \leq f_i(\boldsymbol{x}) \leq u_i(\boldsymbol{x}), \forall \boldsymbol{x} \in [\boldsymbol{l}, \boldsymbol{u}]$.

Here, $\boldsymbol{a_u}$ and b_u are the slope and intercept of the upper linear bound $u_i(\boldsymbol{x})$, respectively. n equals to the input size of $f_i(\boldsymbol{x})$, e.g., when $f_i(\boldsymbol{x})$ is MaxPool and the pool size is 2×2, n equals to 4.

3.3. Computing certified results

Robustness is typically measured by certified accuracy and certified robustness bounds.

3.3.1. Certified accuracy

Certified accuracy is widely used in most verifiers [26, 45, 53, 70] and represents the percentage of the instances verified as safe. In essence, the robustness verifier of neural networks aims to compute the bounds of the output neurons within a perturbed input region. That is, after verification, we would get

$$\boldsymbol{L} \leq F(\boldsymbol{x}) \leq \boldsymbol{U}, \forall x \in \mathbb{B}_p(\boldsymbol{x_0}, \epsilon)$$

where $\boldsymbol{L}$ and $\boldsymbol{U}$ are the lower and upper bounds of the whole network $F(\boldsymbol{x})$, respectively. As $F(\boldsymbol{x}) : \mathbb{R}^{n_0} \rightarrow \mathbb{R}^{n_K}$, we assume $\boldsymbol{L} = (L_1, L_2, \cdots, L_{n_K})$ and $\boldsymbol{U} = (U_1, U_2, \cdots, U_{n_K})$. $[n]$ denotes $\{1, 2, \cdots, n\}$. Then, if

$$L_t > U_i, \forall i \neq t, i \in [n_K] \tag{2}$$

we can conclude that ϵ is a safe perturbation radius, i.e., $\boldsymbol{x_0}$ is verified as safe under the perturbation whose radius is ϵ and form is l_p norm. Otherwise, $\boldsymbol{x_0}$ is verified as unknown. The certified accuracy is computed as the proportion of inputs verified as safe relative to the total number of inputs.

3.3.2. Certified robustness bounds

Some verification methods [5, 63] would use the binary search algorithm to compute the maximal certified lower bounds for input $\boldsymbol{x_0}$, a.k.a. the certified robustness bounds, as their robustness results. Concretely, the perturbation range is initialized as ϵ. Given an input $\boldsymbol{x_0}$ and neural network F, the verification framework computes the bounds for the output layers within the input region $\boldsymbol{x} \in \mathbb{B}_p(\boldsymbol{x_0}, \epsilon)$. $\boldsymbol{L}$ and $\boldsymbol{U}$ represent the lower and upper bounds of neural network F, respectively. When the perturbation range is certified safe, the perturbation ranges ϵ would be increased. When ϵ cannot be certified safe, ϵ would be decreased. Finally, when the above verification process is repeated, the search range is gradually reduced. After repeated finite times, the algorithm terminates and returns ϵ as the maximal certified lower bound that the verifier can verify for the input $\boldsymbol{x_0}$.

4. Ti-Lin: A robustness verifier for MaxPool-based networks

In this section, we present Ti-Lin, the neuron-wise tightest and easy-to-integrate linear approximation for MaxPool.

4.1. Tight linear approximation for MaxPool

We assume that n is the size of inputs that will be pooled. Then, we use $f(x_1, \cdots, x_n) = max\{x_1, \cdots, x_n\}, x_i \in [l_i, u_i]$ to represent the MaxPool function without loss of generality.

Theorem 1. *Given $f(x_1, \cdots, x_n) = max\{x_1, \cdots, x_n\}, x_i \in [l_i, u_i]$, we first select the first, second, and third largest elements of $\{u_i | i = 1, \cdots, n\}$, whose indexes are denoted as i, j, k, respectively. We then choose the largest element of $\{l_i | i = 1, \cdots, n\}$ and denote it as l_{max}. Define $\boldsymbol{m} = (m_1, \cdots, m_n) = (\frac{l_1+u_1}{2}, \cdots, \frac{l_n+u_n}{2}) \in \mathbb{R}^n$. Then the linear bounds of the MaxPool function are:*

Upper linear bound:

$u(x_1, \cdots, x_n) = \sum_i a_i(x_i - l_i) + b$. Specifically, there are four different cases:

Case 1: if $(l_i = l_{max}) \wedge (l_i \geq u_j)$, then $a_i = 1; b = l_i; a_q = 0, q \neq i$.

Case 2: if $(l_i = l_{max}) \wedge (u_j > l_i \geq u_k)$, then $a_i = 1; a_j = \frac{u_j - l_i}{u_j - l_j}; b = l_i; a_q = 0, q \neq i, j$.

Case 3: if $(l_j = l_{max}) \wedge (l_i \neq l_{max}) \wedge (u_j \geq l_j \geq u_k)$, then $a_i = \frac{u_i - l_j}{u_i - l_i}; a_j = 1; b = l_j; a_q = 0, q \neq i, j$.

Case 4: otherwise, $a_i = \frac{u_i - u_k}{u_i - l_i}; a_j = \frac{u_j - u_k}{u_j - l_j}; b = u_k; a_q = 0, q \neq i, j$.

Lower linear bound:

$l(x_1, \cdots, x_n) = x_q, q = \arg\max_i (m_i)$

We prove that the linear bounds in Theorem 1 are sound in the Appendix, that is, $l(\boldsymbol{x}) \leq f(\boldsymbol{x}) \leq u(\boldsymbol{x}), \forall \boldsymbol{x} \in [\boldsymbol{l}, \boldsymbol{u}]$.

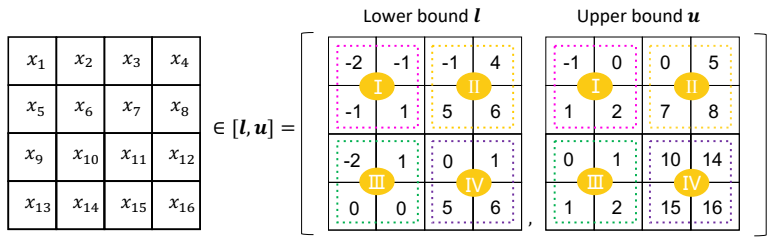
4.2. Neuron-wise tightest property

Linear approximation technique inevitably introduces an over-approximation zone of the intermediate layer's output value to keep sound, according to Equation 1. As for single non-linear layer positioned between linear layers, the over-approximation zone between the upper and lower linear constraints will be propagated throughout the verification process, and affect the whole precision loss. Therefore, it is necessary to minimize the over-approximation zone for every non-linear layer, such as, MaxPool. Furthermore, for cases where two non-linear layers, such as an adjacent ReLU and MaxPool block, are positioned between linear layers, it is also more effective to minimize the over-approximation zone for MaxPool when the activation is bounded by the precise constraints provided by BaB techniques.

We now introduce the concept of neuron-wise tightest [72], which denotes the characteristic of linear bounds that yield the most compact over-approximation zone for the non-linear neuron.

Definition 4 (Neuron-wise Tightest)**.** *Upper linear bound $u : [\boldsymbol{l}, \boldsymbol{u}] \rightarrow \mathbb{R}$ and lower linear bound $l : [\boldsymbol{l}, \boldsymbol{u}] \rightarrow \mathbb{R}$ are the neuron-wise tightest if and only if both $\iint_{[\boldsymbol{l}, \boldsymbol{u}]} (u(\boldsymbol{x}) - l(\boldsymbol{x})) d\boldsymbol{x}$ are the minimal.*

The upper and lower bounds of the intermediate layer's input value rely on the verification process of the previous layers and decide the input region of the current layer's linear approximation. Based on this, we can get the general



Region	Case	Linear bounds	Diagonal of $[\boldsymbol{l}, \boldsymbol{u}]$
I	1	$u(x_1, x_2, x_5, x_6) = x_6$ $l(x_1, x_2, x_5, x_6) = x_6$	$\boldsymbol{a} = (l_1, l_2, l_5, l_6)$ $\boldsymbol{b} = (u_1, u_2, u_5, u_6)$
II	2	$u(x_3, x_4, x_7, x_8) = 0.5x_7 + x_8 - 2.5$ $l(x_3, x_4, x_7, x_8) = x_8$	$\boldsymbol{a} = (l_3, l_4, l_7, u_8)$, $\boldsymbol{b} = (u_3, u_4, u_7, l_8)$
III	3	$u(x_9, x_{10}, x_{13}, x_{14}) = x_{10} + 0.5x_{14} = 1 + 0.5x_{14}$ $l(x_9, x_{10}, x_{13}, x_{14}) = x_{14}$	$\boldsymbol{a} = (l_9, l_{10}, l_{13}, u_{14})$, $\boldsymbol{b} = (u_9, u_{10}, u_{13}, l_{14})$
IV	4	$u(x_{11}, x_{12}, x_{15}, x_{16}) = 0.1x_{15} + 0.2x_{16} + 12.3$ $l(x_{11}, x_{12}, x_{15}, x_{16}) = x_{16}$	$\boldsymbol{a} = (l_{11}, l_{12}, l_{15}, u_{16})$, $\boldsymbol{b} = (u_{11}, u_{12}, u_{15}, l_{16})$

Figure 2. Toy examples of the neuron-wise tightest linear bounds for the four cases in Theorem 1.

form of the neuron-wise tightest linear bounds for all non-linear, bounded, and continuous functions in neural networks.

Theorem 2. *Given f is continuous and bounded, we define $\boldsymbol{m} = \frac{\boldsymbol{l}+\boldsymbol{u}}{2}$. Then, (i) if $u(\boldsymbol{m})$ reaches the minimum, then the upper linear bound for f is the neuron-wise tightest; (ii) if $l(\boldsymbol{m})$ reaches the maximum, then the lower linear bound for f is the neuron-wise tightest.*

The proof of Theorem 2 is put in the Appendix due to page limit.

Existing methods [5, 26, 38, 53] ignore the piece-wise linear property of MaxPool and fail to capture the tightest linear approximation for MaxPool. Their resulting coarse linear approximation leads to non-optimal robustness results. In this paper, we can prove that our proposed linear approximation in Theorem 1 is the neuron-wise tightest and produces the smallest over-approximation zone.

Theorem 3. *The MaxPool linear bounds from Theorem 1 are the neuron-wise tightest.*

The proof of Theorem 3 is put in the Appendix due to page limit.

To explain the provably neuron-wise tightness clearly, we provide a toy example for Theorem 1 in Figure 2. In this example, we consider a MaxPool layer with a pool size of 2×2, a padding size of P=[0 0 0 0], and a stride S=[2 2] to clarify the use of Ti-Lin. The MaxPool operation is applied on four regions: I, II, III, and IV, and we give an example for each case in Theorem 1, as shown in Figure 2. According to the values of $\boldsymbol{l}$ and $\boldsymbol{u}$, we can give the linear bounds for each region as shown in Figure 2. We list all $\boldsymbol{a}$ and $\boldsymbol{b}$ for each case, which are the endpoints of the space diagonal of the input region and can be formally defined as:

$$\boldsymbol{a} := \mathbb{I} \cdot \boldsymbol{l} + (\mathbf{1} - \mathbb{I}) \cdot \boldsymbol{u}$$

$$\boldsymbol{b} := \boldsymbol{l} + \boldsymbol{u} - \boldsymbol{a} = (\mathbf{1} - \mathbb{I}) \cdot \boldsymbol{l} + \mathbb{I} \cdot \boldsymbol{u}$$

where $\cdot$ is dot product and $\mathbb{I}$ represents a indicator matrix that has the same shape as $\boldsymbol{l}$. From Figure 2, it is easy to get $f(\boldsymbol{a}) = u(\boldsymbol{a}), f(\boldsymbol{b}) = u(\boldsymbol{b})$. Then, leveraging the linearity of the upper linear bounds $u(\boldsymbol{x})$, we have $\forall u' \in \mathcal{U}, u(\boldsymbol{m}) = \frac{1}{2}(u(\boldsymbol{a}) + u(\boldsymbol{b})) = \frac{1}{2}(f(\boldsymbol{a}) + f(\boldsymbol{b})) \leq \frac{1}{2}(u'(\boldsymbol{a}) + u'(\boldsymbol{b})) = u'(\frac{1}{2}(\boldsymbol{a}+\boldsymbol{b})) = u'(\boldsymbol{m})$. Then, we prove that the upper linear bound is neuron-wise tightest, according to Theorem 2. In terms of lower linear bound, we have $\forall l' \in \mathcal{L}, l(\boldsymbol{m}) = f(\boldsymbol{m}) \geq l'(\boldsymbol{m})$. Then, we can prove that the lower linear bound is neuron-wise tightest, according to Theorem 2.

The time complexity of Theorem 1 depends on selecting the three largest upper bounds. The time complexity of computing the linear bound for one region is $\mathcal{O}(pool_size \times log3)$, where $pool_size$ is the size of inputs to be pooled. For example, in Figure 2, the MaxPool layer operates on a kernel of 2×2 and thus, $pool_size = 2 \times 2 = 4$.

5. Experimental evaluation

We conduct extensive experiments on CNNs by comparing Ti-Lin to seven SOTA verifiers, including CNN-Cert [5], DeepPoly [53], 3DCertify [38], MN-BaB [17], ERAN [38, 45, 51, 53, 54], OSIP [26], α, β-CROWN [56, 59, 67, 69–71], the VNN-COMP 2021-2023 winner [3, 6, 44], and MaxLin [65]. All experiments were conducted on a server with a 48core Intel Xeon Silver 4310 CPU and 125 GB RAM.

Table 1. The performance of Ti-Lin on DeepPoly verifier (backsubstitution-based).

Dataset	Network	Certified Accuracy (%)			UB (%)[‡]	Average Runtime(s)			Speedup
		DeepPoly	3DCertify[†]	Ti-Lin[†]		DeepPoly	3DCertify	Ti-Lin	vs. 3DCertify
MNIST	Conv_MaxPool	39.5	50.6	**53.1**	93.8	0.9	2.6	1.4	**1.9**
	Convnet_MaxPool	3.1	3.1	**4.2**	71.9	0.2	0.3	0.2	**1.5**
	CNN, 4 layers	79.8	86.2	**89.4**	98.9	2.5	6.7	4.6	**1.5**
	CNN, 5 layers	90.9	90.9	**92.9**	100.0	13.5	23.2	13.8	**1.7**
	CNN, 6 layers	78.8	**85.9**	**85.9**	99.0	23.9	56.0	27.0	**2.1**
	CNN, 7 layers	81.3	86.8	**90.1**	100.0	32.7	71.1	44.7	**1.6**
	CNN, 8 layers	38.4	70.7	**77.8**	99.0	30.1	76.8	41.7	**1.8**
CIFAR-10	Conv_MaxPool	0.0	0.0	**2.1**	93.9	19.7	30.3	17.8	**1.7**
	CNN, 4 layers	59.3	66.7	**70.4**	100.0	8.6	14.2	10.9	**1.3**
	CNN, 5 layers	68.8	68.8	**75.0**	87.5	36.2	55.5	35.4	**1.6**
	CNN, 6 layers	**85.7**	**85.7**	**85.7**	100.0	58.2	115.5	55.4	**2.1**
	CNN, 7 layers	13.8	34.5	**37.9**	72.7	115.1	107.7	95.6	**1.1**
	CNN, 8 layers	41.7	**50.0**	**50.0**	91.7	60.5	128.9	55.8	**2.3**
ModelNet40	16p_Natural	74.6	74.6	**76.3**	-	7.3	13.0	7.7	**1.7**
	32p_Natural	56.5	56.5	**58.1**	-	13.2	25.4	14.0	**1.8**
	64p_Natural	28.1	28.1	**29.7**	-	32.1	81.2	30.8	**2.6**
	64p_FGSM	12.1	12.1	**13.6**	-	27.8	177.0	42.3	**4.2**
	64p_IBP	**88.3**	**88.3**	**88.3**	-	29.3	174.9	32.8	**5.3**

[†] 3DCertify is built atop the DeepPoly framework, which is why we integrate Ti-Lin into DeepPoly for comparison. This integration allows us to evaluate their performance within the same framework.

[‡] UB represents the upper bounds for robustness verification computed by PGD attacks [40]. As PGD attacks is not applicable to PointNets, the UB results of PointNets is - .

5.1. Experimental setup

Framework. Our proposed neuron-wise tightest linear bounds for MaxPool are independent of the concrete verifier. For a fair comparison, we compare Ti-Lin to the baseline methods on the same frameworks. Concretely, we instantiate Ti-Lin on three types of verifiers, including the backsubstitution-based verifiers (CNN-Cert, DeepPoly, and 3DCertify), the multi-neuron abstraction (MN) based verifier (ERAN), and the Branch and Bound (BaB) based verifier (α, β-CROWN). Concretely, backsubstitution [53], a.k.a. bound propagation, uses linear approximation to compute the bounds of the output neurons and achieves a good balance between precision and efficiency. MN [45, 52] leverages the interdependencies between neurons and computes optimal convex relaxations for the output of k-ReLU operations jointly for improved precision. BaB [70] computes the sound linear bounds for the nonlinear layers and recursively splits the layer's input region with added constraints to achieve increasingly tighter bounds.

Benchmarks and experimental settings. We follow the settings used in the baseline methods. MNIST [35], CIFAR-10 [34], Tiny ImageNet [13], and ModelNet40 [64] are used in experiments. All networks in experiments are open-sourced and come from 3DCertify, ERAN, VNN-COMP2021 [3], and CNN-Cert. We follow the timeout setting in VNN-COMP2021 and α, β-CROWN, timeout is 180 seconds for all experiments. Other detailed experiment settings are in the Appendix.

Metric. We follow the metrics used in the baseline methods. In terms of effectiveness, we use the certified accuracy as our metric. In terms of efficiency, we record the average runtime over all instances. We use t'/t to represent the relative time improvement of Ti-Lin over the baseline methods, where t and t' denote the average computation time of Ti-Lin and the baseline methods. On the α, β-CROWN framework, we also record the timeout rate. ***It is worth noting that, we only compute the speedup of Ti-Lin over 3DCertify, ERAN, and OSIP***, as both 3DCertify and ERAN MaxPool linear bounds rely on the double description method [18] to search for optimal linear bounds, which has high time complexity for computing its MaxPool linear approximation compared to other methods. While OSIP bounds MaxPool by transforming it into a sequence of ReLU and affine layers. However, the additional layers introduced by this transformation significantly increase the computational time. As for other methods, whose linear approximation for MaxPool is directly obtained by sorting the upper and lower bounds, operate similarly to our proposed method, and the speedup of Ti-Lin is primarily due to its high precision rather than its low time complexity. Therefore, we only bold the minimal runtime results.

5.2. Performance on backsubstitution-based verifier

CNN-Cert and DeepPoly are two state-of-the-art and popular backsubstitution-based verifiers. 3DCertify, built atop DeepPoly, is a tight verifier for point cloud models. We conduct addtional experiments to compare Ti-Lin to CNN-Cert in Appendix due to the page limit.

Table 2. The performance of Ti-Lin on ERAN verifier (MN-based).

Dataset	Network	Certified Accuracy (%)			UB (%)	Average Runtime(min)			Speedup
		MN-BaB	ERAN	Ti-Lin†		MN-BaB	ERAN	Ti-Lin	vs. ERAN
MNIST	Conv_MaxPool	38.3	56.8	**58.0**	93.8	1.8	1.2	0.7	**1.7**
	ConvNet_MaxPool	3.1	**6.3**	**6.3**	71.9	2.5	0.6	0.4	**1.6**
	CNN, 4 layers	79.8	87.2	**89.4**	98.9	1.8	0.4	0.3	**1.4**
	CNN, 5 layers	90.9	91.9	**92.9**	100.0	1.5	0.8	0.5	**1.4**
	CNN, 6 layers	78.8	**85.9**	**85.9**	99.0	1.9	2.3	2.1	**1.1**
	CNN, 7 layers	81.3	86.8	**90.1**	100.0	1.6	3.0	1.9	**1.5**
	CNN, 8 layers	38.4	70.7	**77.8**	99.0	5.0	7.4	6.1	**1.2**
CIFAR-10	Conv_MaxPool	0.0	0.0	**2.1**	93.9	5.0	17.3	13.1	**1.3**
	CNN, 4 layers	59.3	66.7	**70.4**	100.0	3.7	5.8	5.6	**1.0**
	CNN, 5 layers	68.8	68.8	**75.0**	87.5	2.9	2.9	2.0	**1.4**
	CNN, 6 layers	**85.7**	**85.7**	**85.7**	100.0	2.9	5.1	3.1	**1.6**
	CNN, 7 layers	13.8	34.5	**37.9**	72.7	6.8	9.0	7.1	**1.3**
	CNN, 8 layers	41.7	50.0	**58.3**	91.7	4.9	11.1	9.2	**1.2**

† MN-BaB uses PRIMA's BaB Bound for MaxPool and employs a dual solver in the BaB solver, which makes it difficult to integrate Ti-Lin directly. However, both MN-BaB and ERAN use MN bounding for ReLU. Therefore, in this table, we integrate Ti-Lin into ERAN for comparison.

In this experiment, we integrate Ti-Lin into DeepPoly and compare Ti-Lin to both DeepPoly and 3DCertify across various networks, including CNNs trained on MNIST and CIFAR-10 and PointNets trained on ModelNet40. The PointNets we use are from 3DCertify. 16p, Natural, FGSM, and IBP represent the number point of inputs, naturally trained model, adversarial trained model by FGSM [21, 61], and verified trained model by IBP [24], respectively. The results are shown in Table 1. The results demonstrate that Ti-Lin provides higher certified accuracy in all settings, improving by up to 39.4% and 7.1% over DeepPoly and 3DCertify on MNIST_CNN_8layers, respectively. As for time efficiency, 3DCertify computes the upper bound of every possible upper linear bound for the MaxPool function to choose the best upper linear bound, while Ti-Lin has provably neuron-wise tightest linear bounds and offers the best upper linear bound directly. Therefore, Ti-Lin is more efficient than 3DCertify with up to 5.3× speedup. Because both Ti-Lin and DeepPoly give the linear bounds directly, they share almost the same time efficiency.

5.3. Performance on MN-based verifier

MN-BaB is a state-of-the-art complete neural network verifier that builds on the tight multi-neuron constraints in PRIMA [45], and ERAN is the cutting-edge MN-based verifier. To evaluate the performance of Ti-Lin on the MN-based verifier, we compare Ti-Lin to both MN-BAB and ERAN in this experiment. We integrate Ti-Lin into ERAN to compare the performance of Ti-Lin to MN-BaB and ERAN. Concretely, we use the group-wise joint neuron abstractions [45] for ReLU and MILP, and LP [54] solvers to analyze.

We compute the results of Ti-Lin built atop ERAN, which are shown in Table 2. The results show that Ti-Lin verifies higher certified accuracy across all networks on different datasets with up to 39.4% and 7.1% improvement on MNIST_CNN_8layers, respectively. In terms of efficiency, as ERAN uses MaxPool linear bounds of 3DCertify, which is time-consuming, Ti-Lin has less time cost in all cases than ERAN with up to 1.7× speedup on MNIST_Conv_MaxPool. However, although Ti-Lin computes faster than MN-BaB on most networks trained on the MNIST dataset, both Ti-Lin and ERAN are comparatively slower than MN-BaB on networks trained on the CIFAR-10 dataset, with a maximum slowdown of 2.6 × compared to MN-BaB on CIFAR_Conv_MaxPool. The reason is that although the timeout thresholds of these methods are all 180 seconds, the average runtime can vary across different frameworks. For example, ERAN may try its DeepPoly verifier first and then try using multi-neuron constraint to further tighten the output bounds, which may cause additional time consumption. For a fair comparison, we only compare the time efficiency of Ti-Lin to ERAN.

5.4. Performance on BaB-based verifier

α, β-CROWN is an efficient linear bound propagation framework and uses BaB for ReLU to tighten robustness results. To illustrate the superiority of Ti-Lin, which provides neuron-wise tightest linear bounds, over MaxLin, which offers block-wise tightest linear bounds, we integrate both Ti-Lin and MaxLin into the α, β-CROWN framework. Concretely, in addition to the default settings of the α, β-CROWN framework, we allocate more computational time for the BaB procedure in the ReLU layers and set the timeout for BaB to 3600 seconds, ensuring the verifier can explore the feasible region for ReLU more thoroughly during the verification process.

The results, shown in Table 3, demonstrate that Ti-Lin

Table 3. The performance of Ti-Lin on α, β-CROWN verifier (BaB-based).

Dataset	Network	Certified Accuracy (%)			UB (%)	Timeout Rate(%)			Average Runtime(min)		
		α, β-CROWN	MaxLin*	Ti-Lin**		α, β-CROWN	MaxLin	Ti-Lin	α, β-CROWN	MaxLin	Ti-Lin
MNIST	Conv_MaxPool	45.7	60.5	**66.7**	93.8	48.1	33.3	**27.2**	1.7	1.8	**1.4**
	Convnet_MaxPool	4.2	10.4	**19.8**	71.9	21.9	15.6	**6.3**	0.7	0.5	**0.3**
	CNN, 4 layers†	79.8	**92.6**	**92.6**	98.9	18.1	**5.3**	**5.3**	0.8	0.7	**0.6**
	CNN, 5 layers	9.1	68.7	**85.9**	100.0	90.9	28.3	**26.3**	6.2	3.2	**2.1**
	CNN, 6 layers	4.04	71.7	**73.7**	99.0	94.9	30.3	**13.1**	9.5	3.8	**3.3**
	CNN, 7 layers	**0.0**	**0.0**	**0.0**	100.0	**98.9**	**98.9**	**98.9**	11.8	10.5	**8.1**
	CNN, 8 layers	0.0	0.0	**3.0**	99.0	98.0	98.0	**94.9**	14.2	13.2	**12.5**
CIFAR-10	Conv_MaxPool	2.1	2.1	**6.3**	93.9	58.3	58.3	**54.2**	55.1	50.2	**11.9**
	CNN, 4 layers	63.6	74.1	**90.9**	100.0	36.4	11.1	**9.1**	2.3	3.7	**2.1**
	CNN, 5 layers	0.0	0.0	**6.3**	87.5	81.3	81.3	**75.0**	3.0	4.9	**2.5**
	CNN, 6 layers	7.1	57.1	**85.7**	100.0	92.9	42.9	**15.3**	8.3	6.6	**5.9**
	CNN, 7 layers	0.0	**3.4**	**3.4**	72.7	86.2	**82.8**	**82.8**	8.3	14.2	**6.7**
	CNN, 8 layers	**0.0**	**0.0**	**0.0**	91.7	**83.3**	**83.3**	**83.3**	12.5	11.6	**11.2**

† CNNs with 4-8 layers on MNIST and CIFAR-10, are initially saved in the ".h5" format and are converted into ".onnx" to be tested in this table. The transformation preserve the same predict accuracy but its robustness is affected [25]. Thus, only the certified accuracy in the same table, rather than in different result tables, can effectively illustrate the performance of verifiers.

* MaxLin, as proposed in its original paper, is designed to be integrated into various verification frameworks. In this work, we follow the approach in MaxLin to integrate it into α, β-CROWN and evaluate its performance in the BaB-bound setting for ReLU.

** In this table, Ti-Lin is also integrated into α, β-CROWN for comparison.

can improve the certified accuracy by up to 78.6% compared to α, β-CROWN, and by 28.6% compared to MaxLin implemented in α, β-CROWN on the CIFAR_CNN_6layer model. This reveals that verifiers with the neuron-wise linear bounds for MaxPool can greatly reduce the overestimation and compute higher verified accuracy. As for efficiency, our method can accelerate the computation process in all cases in Table 3. The speedup over α, β-CROWN is due to its adaptive linear bounds for MaxPool, which are gradually optimized during the propagation process, resulting in more time consumption. While the speedup over MaxLin is attributed to Ti-Lin's higher precision when using BaB for ReLU, allowing Ti-Lin to handle fewer timeout samples, which leads to faster computation.

5.5. Additional experiments

We conduct additional experiments to further demonstrate the superiority of Ti-Lin. The detailed experiment settings are in the Appendix, and we perform the following experiments: (I) We conduct extensive experiments by comparing Ti-Lin to CNN-Cert, a backsubstitution-based verifier, by integrating our neuron-wise tightest linear approximation into CNN-Cert. (II) We compare Ti-Lin to OSIP to illustrate the superiority of tight linear approximation for MaxPool over the MaxPool2ReLU transformation. (III) We analyze the global lower bounds of the last layer computed by Ti-Lin and other methods, including DeepPoly, 3DCertify, optimized linear bounds, used in α, β-CROWN, and MaxLin to further illustrate the advantages of the neuron-wise tightness over other linear bounds. (IV) To illustrate the difference between MaxLin and Ti-Lin, we introduce Hybrid-Lin in the Appendix, Hybrid-Lin applies MaxLin when the ReLU upper linear bound is $u(x) = \frac{u}{u-l}(x-l)$ and Ti-Lin when the ReLU upper linear bound is $u(x) = 0$ or $u(x) = x$. We compare the CROWN and α-CROWN results of Ti-Lin, MaxLin, and Hybrid-Lin, demonstrating that ***rather than competing, MaxLin and Ti-Lin complement each other in achieving optimal bounding precision.***

6. Conclusion

In this paper, we propose Ti-Lin, a robustness verifier for MaxPool-based CNNs. We are the first to propose the neuron-wise tightest linear bounds for the MaxPool function. We implemented Ti-Lin in various networks on the MNIST, CIFAR-10, Tiny ImageNet, and ModelNet40 datasets. Experimental results show the effectiveness of the neuron-wise tightest techniques with up to 78.6% improvement to the certified accuracy over the state-of-the-art linear approximation methods with the same time consumption as the fastest tools. The results also show that transforming MaxPool to ReLU can lead to more time consumption and more coarse verification results. Moreover, as Ti-Lin is simple to be integrated into other verifiers, the results after integration strongly show that Ti-Lin reduce the overestimation and compute tighter certified robustness results.

Acknowledgements. The authors would like to thank the anonymous reviewers for their insightful comments. This work is partially supported by the National Natural Science Foundation of China (U24A20337, 62372228), the Shenzhen-Hong Kong-Macau Technology Research Programme (Type C) (Grant No. SGDX20230821091559018), and the Fundamental Research Funds for the Central Universities (14380029).

References

[1] Ross Anderson, Joey Huchette, Will Ma, Christian Tjandraatmadja, and Juan Pablo Vielma. Strong mixed-integer programming formulations for trained neural networks. *Math. Program.*, 183:3–39, 2020. 2

[2] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *International conference on machine learning (ICML)*, pages 274–283, 2018. 2

[3] Stanley Bak, Changliu Liu, and Taylor Johnson. The second international verification of neural networks competition (vnn-comp 2021): Summary and results. *arXiv preprint arXiv:2109.00498*, 2021. 2, 5, 6

[4] Mislav Balunovic, Maximilian Baader, Gagandeep Singh, Timon Gehr, and Martin Vechev. Certifying geometric robustness of neural networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 32:1–11, 2019. 2

[5] Akhilan Boopathy, Tsui-Wei Weng, Pin-Yu Chen, Sijia Liu, and Luca Daniel. Cnn-cert: an efficient framework for certifying robustness of convolutional neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 3240–3247, 2019. 1, 2, 4, 5, 18

[6] Christopher Brix, Stanley Bak, Changliu Liu, and Taylor T. Johnson. The fourth international verification of neural networks competition (vnn-comp 2023): Summary and results. *arXiv preprint arXiv:2312.16760*, 2023. 2, 5

[7] Jacob Buckman, Aurko Roy, Colin Raffel, and Ian Goodfellow. Thermometer encoding: One hot way to resist adversarial examples. In *International conference on learning representations (ICLR)*, 2018. 2

[8] Nicholas Carlini and David Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In *Proceedings of the 10th ACM workshop on artificial intelligence and security*, pages 3–14, 2017. 2

[9] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *IEEE S & P*, pages 39–57, 2017. 2

[10] Jianbo Chen, Michael I Jordan, and Martin J Wainwright. Hopskipjumpattack: A query-efficient decision-based attack. In *IEEE S & P*, pages 1277–1294, 2020.

[11] Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *Proceedings of the 10th ACM workshop on artificial intelligence and security*, pages 15–26, 2017. 2

[12] Jeremy Cohen, Elan Rosenfeld, and Zico Kolter. Certified adversarial robustness via randomized smoothing. In *International conference on machine learning (ICML)*, pages 1310–1320, 2019. 2

[13] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255, 2009. 6, 18

[14] Tianyu Du, Shouling Ji, Lujia Shen, Yao Zhang, Jinfeng Li, Jie Shi, Chengfang Fang, Jianwei Yin, Raheem Beyah, and Ting Wang. Cert-rnn: Towards certifying the robustness of recurrent neural networks. In *ACM Conference on Computer and Communications Security (CCS)*, pages 516–534, 2021. 1

[15] Rüdiger Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In *Automated Technology for Verification and Analysis(ATVA)*, pages 269–286. Springer, 2017. 2

[16] Mahyar Fazlyab, Manfred Morari, and George J Pappas. Probabilistic verification and reachability analysis of neural networks via semidefinite programming. In *IEEE 58th Conference on Decision and Control (CDC)*, pages 2726–2731, 2019. 1

[17] Claudio Ferrari, Mark Niklas Mueller, Nikola Jovanović, and Martin Vechev. Complete verification via multi-neuron relaxation guided branch-and-bound. In *International Conference on Learning Representations (ICLR)*, 2022. 1, 2, 5

[18] Komei Fukuda and Alain Prodon. Double description method revisited. In *Combinatorics and Computer Science*, pages 91–111, 2005. 2, 6

[19] Amin Ghiasi, Ali Shafahi, and Tom Goldstein. Breaking certified defenses: Semantic adversarial examples with spoofed robustness certificates. *arXiv preprint arXiv:2003.08937*, pages 1–16, 2020. 2

[20] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014. 2

[21] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations (ICLR)*, 2015. 7

[22] Divya Gopinath, Guy Katz, Corina S Păsăreanu, and Clark Barrett. Deepsafe: a data-driven approach for assessing robustness of neural networks. In *International symposium on automated technology for verification and analysis (ATVA)*, pages 3–19, 2018. 1

[23] Gaurav Goswami, Nalini Ratha, Akshay Agarwal, Richa Singh, and Mayank Vatsa. Unravelling robustness of deep learning based face recognition against adversarial attacks. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 1–10, 2018. 1

[24] Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Relja Arandjelovic, Timothy Arthur Mann, and Pushmeet Kohli. Scalable verified training for provably robust image classification. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 4841–4850, 2019. 7

[25] Qianyu Guo, Sen Chen, Xiaofei Xie, Lei Ma, Qiang Hu, Hongtao Liu, Yang Liu, Jianjun Zhao, and Xiaohong Li. An empirical study towards characterizing deep learning development and deployment across different frameworks and platforms. In *IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 810–822, 2019. 8

[26] Vahid Hashemi, Panagiotis Kouvaros, and Alessio Lomuscio. OSIP: tightened bound propagation for the verification of relu neural networks. In *Software Engineering and Formal Methods (SEFM)*, pages 463–480, 2021. 1, 2, 4, 5, 19

[27] Patrick Henriksen and Alessio Lomuscio. Efficient neural network verification via adaptive refinement and adversarial

search. In *European Conference on Artificial Intelligence (ECAI)*, pages 2513–2520. 2020. 1, 2

[28] Pei Huang, Yuting Yang, Minghao Liu, Fuqi Jia, Feifei Ma, and Jian Zhang. ϵ-weakened robustness of deep neural networks. In *International Symposium on Software Testing and Analysis (ISSTA)*, pages 126–138, 2022. 1

[29] Jinyuan Jia, Binghui Wang, Xiaoyu Cao, and Neil Zhenqiang Gong. Certified robustness of community detection against adversarial structural perturbation via randomized smoothing. In *Proceedings of The Web Conference (WWW)*, pages 2718–2724, 2020. 1

[30] Jinyuan Jia, Zhuowen Yuan, Dinuka Sahabandu, Luyao Niu, Arezoo Rajabi, Bhaskar Ramasubramanian, Bo Li, and Radha Poovendran. Fedgame: A game-theoretic defense against backdoor attacks in federated learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. 2

[31] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: an efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification (CAV)*, pages 97–117, 2017. 1, 2, 3

[32] Guy Katz, Derek A Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljić, et al. The marabou framework for verification and analysis of deep neural networks. In *International Conference on Computer Aided Verification (CAV)*, pages 443–452, 2019. 2

[33] Ching-Yun Ko, Zhaoyang Lyu, Lily Weng, Luca Daniel, Ngai Wong, and Dahua Lin. Popqorn: quantifying robustness of recurrent neural networks. In *International Conference on Machine Learning (ICML)*, pages 3468–3477, 2019. 1

[34] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. pages 1–60, 2009. 2, 6

[35] Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, page 1, 1998. 2, 6

[36] Han Liu, Zhi Xu, Xiaotong Zhang, Feng Zhang, Fenglong Ma, Hongyang Chen, Hong Yu, and Xianchao Zhang. Hqa-attack: Toward high quality black-box hard-label adversarial attack on text. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. 2

[37] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (CVPR)*, pages 11976–11986, 2022. 2

[38] Tobias Lorenz, Anian Ruoss, Mislav Balunović, Gagandeep Singh, and Martin Vechev. Robustness certification for point cloud models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 7608–7618, 2021. 1, 2, 5

[39] Zhaoyang Lyu, Ching-Yun Ko, Zhifeng Kong, Ngai Wong, Dahua Lin, and Luca Daniel. Fastened crown: tightened neural network robustness certificates. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 5037–5044, 2020. 1, 2

[40] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *6th International Conference on Learning Representations (ICLR)*, 2018. 6

[41] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations (ICLR)*, 2018. 2

[42] Mark Huasong Meng, Guangdong Bai, Sin Gee Teo, Zhe Hou, Yan Xiao, Yun Lin, and Jin Song Dong. Adversarial robustness of deep neural networks: a survey from a formal verification perspective. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 1:1–18, 2022. 2

[43] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 1765–1773, 2017. 1

[44] Mark Niklas Müller, Christopher Brix, Stanley Bak, Changliu Liu, and Taylor T Johnson. The third international verification of neural networks competition (vnn-comp 2022): summary and results. *arXiv preprint arXiv:2212.10376*, 2022. 2, 5

[45] Mark Niklas Müller, Gleb Makarchuk, Gagandeep Singh, Markus Püschel, and Martin Vechev. Prima: general and precise neural network certification via scalable convex hull approximations. *Proceedings of the ACM on Programming Languages (POPL)*, pages 1–33, 2022. 1, 2, 4, 5, 6, 7

[46] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *IEEE S & P*, pages 582–597, 2016. 2

[47] Wonryong Ryou, Jiayu Chen, Mislav Balunovic, Gagandeep Singh, Andrei Dan, and Martin Vechev. Scalable polyhedral verification of recurrent neural networks. In *Computer Aided Verification: 33rd International Conference (CAV)*, pages 225–248. Springer, 2021. 1

[48] Hadi Salman, Greg Yang, Huan Zhang, Cho-Jui Hsieh, and Pengchuan Zhang. A convex relaxation barrier to tight robustness verification of neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 9832–9842, 2019. 2

[49] Zhouxing Shi, Qirui Jin, Huan Zhang, Zico Kolter, Suman Jana, and Cho-Jui Hsieh. Formal verification for neural networks with general nonlinearities via branch-and-bound. In *2nd Workshop on Formal Verification of Machine Learning (WFVML)*, 2023. 1

[50] Samuel Henrique Silva and Peyman Najafirad. Opportunities and challenges in deep learning adversarial robustness: A survey, 2020. 2

[51] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. Fast and effective robustness certification. *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1–12, 2018. 1, 2, 5

[52] Gagandeep Singh, Rupanshu Ganvir, Markus Püschel, and Martin Vechev. Beyond the single neuron convex barrier for neural network certification. *Advances in Neural Information Processing Systems (NeurIPS)*, 32:1–12, 2019. 6

[53] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages (POPL)*, pages 1–30, 2019. 1, 2, 4, 5, 6

[54] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. Boosting robustness certification of neural networks. In *International Conference on Learning Representations (ICLR)*, pages 1–12, 2019. 2, 5, 7

[55] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *International Conference for Learning Representations (ICLR)*, pages 1–10, 2014. 1

[56] Vincent Tjeng, Kai Y Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. In *International Conference on Learning Representations (ICLR)*, pages 1–21, 2018. 2, 5

[57] Hoang-Dung Tran, Stanley Bak, Weiming Xiang, and Taylor T. Johnson. Verification of deep convolutional neural networks using imagestars. In *Computer Aided Verification - 32nd International Conference (CAV)*, pages 18–42, 2020. 1

[58] Hoang-Dung Tran, Xiaodong Yang, Diego Manzanas Lopez, Patrick Musau, Luan Viet Nguyen, Weiming Xiang, Stanley Bak, and Taylor T. Johnson. NNV: the neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In *Computer Aided Verification - 32nd International Conference (CAV)*, pages 3–17, 2020. 1

[59] Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. Beta-crown: efficient bound propagation with per-neuron split constraints for neural network robustness verification. *Advances in Neural Information Processing Systems (NeurIPS)*, 34:29909–29921, 2021. 1, 2, 5

[60] Lily Weng, Pin-Yu Chen, Lam Nguyen, Mark Squillante, Akhilan Boopathy, Ivan Oseledets, and Luca Daniel. Proven: Verifying robustness of neural networks with a probabilistic approach. In *International Conference on Machine Learning (ICML)*, pages 6727–6736, 2019. 1

[61] Eric Wong, Leslie Rice, and J. Zico Kolter. Fast is better than free: Revisiting adversarial training. In *International Conference on Learning Representations (ICLR)*, 2020. 7

[62] Baoyuan Wu, Li Liu, Zihao Zhu, Qingshan Liu, Zhaofeng He, and Siwei Lyu. Adversarial machine learning: A systematic survey of backdoor attack, weight attack and adversarial example, 2023. 2

[63] Yiting Wu and Min Zhang. Tightening robustness verification of convolutional neural networks with fine-grained linear approximation. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 11674–11681, 2021. 1, 2, 4

[64] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 1912–1920, 2015. 2, 6

[65] Yuan Xiao, Shiqing Ma, Juan Zhai, Chunrong Fang, Jinyuan Jia, and Zhenyu Chen. Towards general robustness verification of maxpool-based convolutional neural networks via tightening linear approximation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 24766–24775, 2024. 2, 3, 5

[66] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 1492–1500, 2017. 2

[67] Kaidi Xu, Huan Zhang, Shiqi Wang, Yihan Wang, Suman Jana, Xue Lin, and Cho-Jui Hsieh. Fast and complete: enabling complete neural network verification with rapid and massively parallel incomplete verifiers. *International Conference on Learning Representations (ICLR)*, pages 1–15, 2021. 1, 2, 5

[68] Aoxiang Zhang, Yu Ran, Weixuan Tang, and Yuan-Gen Wang. Vulnerabilities in video quality assessment models: The challenge of adversarial attacks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. 2

[69] Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1–10, 2018. 2, 5

[70] Huan Zhang, Shiqi Wang, Kaidi Xu, Linyi Li, Bo Li, Suman Sekhar Jana, Cho-Jui Hsieh, and J. Zico Kolter. General cutting planes for bound-propagation-based neural network verification. *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. 4, 6

[71] Huan Zhang, Shiqi Wang, Kaidi Xu, Yihan Wang, Suman Jana, Cho-Jui Hsieh, and Zico Kolter. A branch and bound framework for stronger adversarial attacks of ReLU networks. In *Proceedings of Machine Learning Research (PMLR)*, pages 26591–26604, 2022. 2, 5

[72] Zhaodi Zhang, Yiting Wu, Si Liu, Jing Liu, and Min Zhang. Provably tightest linear approximation for robustness verification of sigmoid-like neural networks. In *IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 1–13, 2022. 1, 4

[73] Zhe Zhao, Guangke Chen, Jingyi Wang, Yiwei Yang, Fu Song, and Jun Sun. Attack as defense: characterizing adversarial examples using robustness. In *International Symposium on Software Testing and Analysis (ISSTA)*, pages 42–55, 2021. 1

7. Appendix

7.1. Proof of Theorem 1

In this subsection, we prove the soundness of our linear bounds in Theorem 1.

Proof. Define $\boldsymbol{m} = \frac{\boldsymbol{u}+\boldsymbol{l}}{2} = (m_1, \cdots, m_n) \in \mathbb{R}^n$.

Upper linear bound:

Case 1: When $l_i = l_{max} \wedge l_{max} \geq u_j$, we have$f(x_1, \cdots, x_n) = x_i$ and $u(x_1, \cdots, x_n) = x_i - l_i + l_i = x_i$. Then, we have$u(\boldsymbol{x}) \geq f(\boldsymbol{x})$, that is, the upper linear bound of case 1 is sound.

Case 2: When $l_i = l_{max}, u_j > l_i \geq u_k$, we have $f(x_1, \cdots, x_n) = max(x_i, x_j)$ and $u(x_1, \cdots, x_n) = x_i - l_i + \frac{u_j - l_i}{u_j - l_j}(x_j - l_j) + l_i$.

If $f(x_1, \cdots, x_n) = x_i$,

$$\begin{aligned} u(x_1, \cdots, x_n) - x_i =& x_i - l_i + \frac{u_j - l_i}{u_j - l_j}(x_j - l_j) + l_i - x_i \\ =& \frac{u_j - l_i}{u_j - l_j}(x_j - l_i) \\ \geq& 0 \end{aligned}$$

If $f(x_1, \cdots, x_n) = x_j$,

$$\begin{aligned} u(x_1, \cdots, x_n) - x_j =& x_i - l_i + \frac{u_j - l_i}{u_j - l_j}(x_j - l_j) + l_i - x_j \\ =& (x_i - l_i) + \frac{u_j - l_i}{u_j - l_j}(x_j - l_j) + l_j - x_j - l_j + l_i \\ =& (x_i - l_i) + \frac{l_j - l_i}{u_j - l_j}(x_j - l_j) - l_j + l_i \\ =& (x_i - l_i) + \frac{u_j - x_j}{u_j - l_j}(l_i - l_j) \\ \geq& 0 \end{aligned}$$

Then, we have $u(\boldsymbol{x}) \geq f(\boldsymbol{x})$, that is, the upper linear bound of case 2 is sound.

Case 3: When $l_i = l_{max} \wedge l_i \neq l_{max} \wedge u_j \geq l_j \geq u_k$, we have $f(x_1, \cdots, x_n) = max(x_i, x_j)$ and $u(x_1, \cdots, x_n) = \frac{u_i - l_j}{u_i - l_i} x_i + x_j + l_j$.

If $f(x_1, \cdots, x_n) = x_i$,

$$\begin{aligned} u(x_1, \cdots, x_n) - x_i =& \frac{u_i - l_j}{u_i - l_i}(x_i - l_i) + (x_j - l_j) + l_j - l_i + l_i - x_i \\ =& (x_i - l_i)(\frac{u_i - l_j}{u_i - l_i} - 1) + (x_j - l_j) + l_j - l_i \\ =& (x_i - l_i)\frac{l_i - l_j}{u_i - l_i} + (x_j - l_j) + l_j - l_i \\ =& (l_j - l_i)(1 - \frac{x_i - l_i}{u_i - l_i}) + (x_j - l_j) \\ =& (l_j - l_i)\frac{u_i - x_i}{u_i - l_i} + (x_j - l_j) \\ \geq& 0 \end{aligned}$$

If $f(x_1, \cdots, x_n) = x_j$,

$$\begin{aligned} u(x_1, \cdots, x_n) - x_j =& \frac{u_i - l_j}{u_i - l_i}(x_i - l_i) + (x_j - l_j) + l_j - x_j \\ =& \frac{u_i - l_j}{u_i - l_i}(x_i - l_i) \\ \geq& 0 \end{aligned}$$

Then, we have $u(\boldsymbol{x}) \geq f(\boldsymbol{x})$, that is, the upper linear bound of case 3 is sound.
Case 4: First, we prove that if $\boldsymbol{u}$ and $\boldsymbol{l}$ do not satisfy case 1,2, and 3, then $u_k > max\{l_i, l_j\}$.
We prove this by contradiction.
We assume that $u_k \leq max\{l_i, l_j\}$. Then, as $u_k \geq l_k$ and $u_k \geq max_{s \neq i,j,k}\{u_p\}$, we have $u_k \geq max_{p \neq i,j}\{l_p\}$.
And we have $l_{max} = max\{l_1, \cdots, l_n\} = max\{l_i, l_j, u_k\} = max\{l_i, l_j\}$.
If $l_{max} = l_i$, then we have $l_i < u_j \wedge (l_i < u_k \vee l_i \geq u_j)$, that is, $l_i < u_j \wedge l_i < u_k$. It contradicts $u_k \leq max\{l_i, l_j\}$.
if $l_{max} = l_j \wedge l_{max} \neq l_i$, then we have $l_j > u_j \vee l_j < u_k$, that is $l_j < u_k$. It contradicts $u_k \leq max\{l_i, l_j\}$.
Therefore, in case 4, $u_k > max\{l_i, l_j\}$.
Here, we prove the upper bound in case 4 is sound.
$f(x_1, \cdots, x_n) = max(x_1, \cdots, x_n), \forall (x_1, \cdots, x_n)$. If $f(x_1, \cdots, x_n) = x_i$,

$$\begin{aligned}
u(x_1, \cdots, x_n) - x_i =& \frac{u_i - u_k}{u_i - l_i}(x_i - l_i) + \frac{u_j - u_k}{u_j - l_j}(x_j - l_j) + u_k - x_i \\
=& \frac{u_i - u_k}{u_i - u_k}(x_i - l_i) + \frac{u_j - u_k}{u_j - l_j}(x_j - l_j) + u_k - l_i + l_i - x_i \\
=& (x_i - l_i)(\frac{u_i - u_k}{u_i - l_i} - 1) + \frac{u_j - u_k}{u_j - l_j}(x_j - l_j) + u_k - l_i \\
=& (u_k - l_i)\frac{u_i - x_i}{u_i - l_i} + \frac{u_j - u_k}{u_j - l_j}(x_j - l_j) \\
\geq& 0
\end{aligned}$$

If $f(x_1, \cdots, x_n) = x_j$, the proof is the same as above.

$$\begin{aligned}
u(x_1, \cdots, x_n) - x_j =& \frac{u_i - u_k}{u_i - l_i}(x_i - l_i) + \frac{u_j - u_k}{u_j - l_j}(x_j - l_j) + u_k - x_j \\
=& \frac{u_i - u_k}{u_i - u_k}(x_i - l_i) + \frac{u_j - u_k}{u_j - l_j}(x_j - l_j) + u_k - l_j + l_j - x_j \\
=& \frac{u_i - u_k}{u_i - l_i}(x_i - l_i) + (\frac{u_j - u_k}{u_j - l_j} - 1)(x_j - l_j) + u_k - l_j \\
=& \frac{u_i - u_k}{u_i - l_i}(x_i - l_i) + (1 - \frac{x_j - l_j}{u_j - l_j})(u_k - l_j) \\
=& \frac{u_i - u_k}{u_i - l_i}(x_i - l_i) + \frac{u_j - x_j}{u_j - l_j}(u_k - l_j) \\
\geq& 0
\end{aligned}$$

If $f(x_1, \cdots, x_n) = x_k$,

$$\begin{aligned}
u(x_1, \cdots, x_n) - x_j =& \frac{u_i - u_k}{u_i - l_i}(x_i - l_i) + \frac{u_j - u_k}{u_j - l_j}(x_j - l_j) + (u_k - x_k) \\
\geq& 0
\end{aligned}$$

If $f(x_1, \cdots, x_n) = x_l, l \neq i, j, k$,

$$\begin{aligned}
& u(x_1, \cdots, x_n) - x_l \\
=& \frac{u_i - u_k}{u_i - l_i}(x_i - l_i) + \frac{u_j - u_k}{u_j - l_j}(x_j - l_j) + u_k - x_l \\
=& \frac{u_i - u_k}{u_i - l_i}(x_i - l_i) + \frac{u_j - u_k}{u_j - l_j}(x_j - l_j) + (u_k - u_l) + (u_l - x_l) \\
\geq& 0
\end{aligned}$$

Then, we have $u(\boldsymbol{x}) \geq f(\boldsymbol{x})$, that is, the upper linear bound of case 4 is sound.
Lower linear bound:

$l(x_1, \cdots, x_n) = x_j = argmax_i m_i$, and $\forall (x_1, \cdots, x_n) \in \times_{i=1}^{n} [l_i, u_i]$,

$$\begin{aligned} f(x_1, \cdots, x_n) &= max(x_1, \cdots, x_n) \\ &\geq x_j \\ &= l(x_1, \cdots, x_n) \end{aligned}$$

Then, we have $l(\boldsymbol{x}) \leq f(\boldsymbol{x})$, that is the lower linear bound is sound.

This completes the proof. □

7.2. Proof of Theorem 2

In this subsection, we prove Theorem 2.

Proof. First, $u(\boldsymbol{x})$ and $l(\boldsymbol{x})$ are upper bound and lower bound for $f(\boldsymbol{x})$. Therefore, $u(\boldsymbol{x}) \geq l(\boldsymbol{x}), \forall \boldsymbol{x} \in [\boldsymbol{l}, \boldsymbol{u}]$. Then, we have

$$u(\boldsymbol{x}) - l(\boldsymbol{x}) = (u(\boldsymbol{x}) - f(\boldsymbol{x})) + (f(\boldsymbol{x}) - l(\boldsymbol{x}))$$

$(u(\boldsymbol{x}) - f(\boldsymbol{x}))$ and $(f(\boldsymbol{x}) - l(\boldsymbol{x}))$ are not smaller than 0, when $\forall \boldsymbol{x} \in [\boldsymbol{l}, \boldsymbol{u}]$.

Therefore, minimizing $\iint_{[\boldsymbol{l},\boldsymbol{u}]} (u(\boldsymbol{x}) - l(\boldsymbol{x})) d\boldsymbol{x}$ is equivalent to minimizing both $\iint_{[\boldsymbol{l},\boldsymbol{u}]} (u(\boldsymbol{x}) - f(\boldsymbol{x})) d\boldsymbol{x}$ and $\iint_{[\boldsymbol{l},\boldsymbol{u}]} (f(\boldsymbol{x}) - l(\boldsymbol{x})) d\boldsymbol{x}$. The value of $\iint_{[\boldsymbol{l},\boldsymbol{u}]} f(\boldsymbol{x}) d\boldsymbol{x}$ is constant. Thus, it is also equivalent to minimizing $\iint_{[\boldsymbol{l},\boldsymbol{u}]} u(\boldsymbol{x}) d\boldsymbol{x}$ and $\iint_{[\boldsymbol{l},\boldsymbol{u}]} (-l(\boldsymbol{x})) d\boldsymbol{x}$. Further, we define that lower linear bound $l(\cdot)$ is the neuron-wise tightest when $-l(\boldsymbol{m})$ reaches the minimum, and upper linear bound $u(\cdot)$ is the neuron-wise tightest when $u(\boldsymbol{m})$ reaches the minimum.

Define $\boldsymbol{x} = (x_1, \cdots, x_n)$ and $[n] = \{1, 2, \cdots, n\}$. Because $u(\boldsymbol{x})$ is a linear combination of $x_i, i \in [n]$. Without loss of generality, we assume $u(\boldsymbol{x}) = \sum_{i \in [n]} a_{u,i} x_i + b_u$. Then,

$$\begin{aligned} \iint_{(x_1, \cdots, x_n) \in [\boldsymbol{l},\boldsymbol{u}]} u(x_1, \cdots, x_n) d\boldsymbol{x} &= \iint_{(x_1, \cdots, x_n) \in [\boldsymbol{l},\boldsymbol{u}]} (\sum_{i \in [n]} a_{u,i} x_i + b_u) d\boldsymbol{x} \\ &= \sum_{i=1}^{n} \frac{a_{u,i}}{2} ((u_i)^2 - (l_i)^2) + b_u (u_i - l_i) \\ &= \prod_{i=1}^{n} (u_i - l_i) (\sum_{i \in [n]} a_{u,i} \frac{u_i + l_i}{2} + b_u) \\ &= \prod_{i=1}^{n} (u_i - l_i) u(\boldsymbol{m}) \end{aligned}$$

where $\boldsymbol{m} = (\frac{u_1 + l_1}{2}, \cdots, \frac{u_n + l_n}{2})$. As $u_i, l_i, i \in [n]$ are constant, the minimize target has been transformed into minimizing $u(\boldsymbol{m})$.

Symmetric to the above proof, minimizing $\iint_{(x_1, \cdots, x_n) \in [\boldsymbol{l},\boldsymbol{u}]} u(x_1, \cdots, x_n) d\boldsymbol{x}$ is equivalent to minimize $-l(\boldsymbol{m})$.

This completes the proof. □

7.3. Proof of Theorem 3

In this subsection, we prove Theorem 3, that is, our linear bounds in Theorem 1 are the neuron-wise tightest.

Proof. Define $\boldsymbol{m} = \frac{\boldsymbol{u} + \boldsymbol{l}}{2} = (m_1, \cdots, m_n) \in \mathbb{R}^n$.

Upper linear bound:

Case 1: When $l_i = l_{max} \wedge l_{max} \geq u_j$, we have$f(x_1, \cdots, x_n) = x_i$ and $u(x_1, \cdots, x_n) = x_i - l_i + l_i = x_i$. As $u(m) = f(m) \leq u'(\boldsymbol{m}), \forall u' \in \mathcal{U}$, the upper bound is the neuron-wise tightest.

Case 2: When $l_i = l_{max}, u_j > l_i \geq u_k$, we have $f(x_1, \cdots, x_n) = max(x_i, x_j)$ and $u(x_1, \cdots, x_n) = x_i - l_i + \frac{u_j - l_i}{u_j - l_j}(x_j - l_j) + l_i$.

Because

$$
\begin{aligned}
&u(u_1, \cdots, u_{i-1}, l_i, u_{i+1}, \cdots, u_{j-1}, u_j, u_{j+1}, \cdots, u_n) \\
=&l_i - l_i + \frac{u_j - l_i}{u_j - l_j}(u_j - l_j) + l_i \\
=&u_j \\
=&f(u_1, \cdots, u_{i-1}, l_i, u_{i+1}, \cdots, u_{j-1}, u_j, u_{j+1}, \cdots, u_n)
\end{aligned}
$$

and

$$
\begin{aligned}
&u(l_1, \cdots, l_{i-1}, u_i, l_{i+1}, \cdots, l_{j-1}, l_j, l_{j+1}, \cdots, l_n) \\
=&u_i - l_i + \frac{u_j - l_i}{u_j - l_j}(l_j - l_j) + l_i \\
=&u_i \\
=&f(l_1, \cdots, l_{i-1}, u_i, l_{i+1}, \cdots, l_{j-1}, l_j, l_{j+1}, \cdots, l_n)
\end{aligned}
$$

We notice that
$\boldsymbol{a} := (u_1, \cdots, u_{i-1}, l_i, u_{i+1}, \cdots, u_n)$ and $\boldsymbol{b} := (l_1, \cdots, l_{i-1}, u_i, l_{i+1}, \cdots, l_n)$ are the space diagonal of $\times_{i=1}^{n}[l_i, u_i]$, and $\boldsymbol{m} = \frac{1}{2}(\boldsymbol{a} + \boldsymbol{b})$. Then, as $u(\boldsymbol{x})$ is linear, we have $f(\boldsymbol{a}) + f(\boldsymbol{b}) = f(\boldsymbol{a} + \boldsymbol{b})$.
Then, we have

$$
\begin{aligned}
\forall u' \in \mathcal{U}, u(\boldsymbol{m}) &= u(\frac{1}{2}(\boldsymbol{a} + \boldsymbol{b})) \\
&= \frac{1}{2}(u(\boldsymbol{a}) + u(\boldsymbol{b})) \\
&= \frac{1}{2}(f(\boldsymbol{a}) + f(\boldsymbol{b})) \\
&\leq \frac{1}{2}(u'(\boldsymbol{a}) + u'(\boldsymbol{b})) \\
&= (u'(\frac{1}{2}(\boldsymbol{a} + \boldsymbol{b}))) \\
&= u'(\boldsymbol{m})
\end{aligned}
$$

Therefore, the plane is the neuron-wise tightest upper linear bounding plane.

Case 3: When $l_i = l_{max} \wedge l_i \neq l_{max} \wedge u_j \geq l_j \geq u_k$, we have $f(x_1, \cdots, x_n) = max(x_i, x_j)$ and $u(x_1, \cdots, x_n) = \frac{u_i - l_j}{u_i - l_i} x_i + x_j + l_j$.

Because

$$
\begin{aligned}
&u(u_1, \cdots, u_{i-1}, u_i, u_{i+1}, \cdots, u_{j-1}, l_j, u_{j+1}, \cdots, u_n) \\
=&\frac{u_i - l_j}{u_i - l_i}(u_i - l_i) + (l_j - l_j) + l_j \\
=&f(u_1, \cdots, u_{i-1}, u_i, u_{i+1}, \cdots, u_{j-1}, l_j, u_{j+1}, \cdots, u_n)
\end{aligned}
$$

and

$$
\begin{aligned}
&u(l_1, \cdots, l_{i-1}, l_i, l_{i+1}, \cdots, l_{j-1}, u_j, l_{j+1}, \cdots, l_n) \\
=&\frac{u_i - l_j}{u_i - l_i}(l_i - l_i) + (u_j - l_j) + l_j \\
=&f(l_1, \cdots, l_{i-1}, l_i, l_{i+1}, \cdots, l_{j-1}, u_j, l_{j+1}, \cdots, l_n)
\end{aligned}
$$

We notice that
$\boldsymbol{a} := (u_1, \cdots, u_{j-1}, l_j, u_{j+1}, \cdots, u_n)$ and $\boldsymbol{b} := (l_1, \cdots, l_{j-1}, u_j, l_{j+1}, \cdots, l_n)$ are the space diagonal of $\times_{i=1}^{n}[l_i, u_i]$, and
$\boldsymbol{m} = \frac{1}{2}(\boldsymbol{a} + \boldsymbol{b})$. Similar to the proof in case 2, we can prove that the plane is the neuron-wise tightest linear upper bounding plane.

Case 4: First, as proved in Section 7.1, $u_k > max\{l_i, l_j\}$ in case 4.

Because

$$
\begin{aligned}
&u(u_1, \cdots, u_{i-1}, u_i, u_{i+1}, \cdots, u_{j-1}, l_j, u_{j+1}, \cdots, u_n) \\
=&\frac{u_i - u_k}{u_i - l_i}(u_i - l_i) + \frac{u_j - u_k}{u_j - l_j}(l_j - l_j) + u_k \\
=&u_i - u_k + u_k \\
=&f(u_1, \cdots, u_{i-1}, u_i, u_{i+1}, \cdots, u_{j-1}, l_j, u_{j+1}, \cdots, u_n)
\end{aligned}
$$

and

$$
\begin{aligned}
&u(l_1, \cdots, l_{i-1}, l_i, l_{i+1}, \cdots, l_{j-1}, u_j, l_{j+1}, \cdots, l_n) \\
=&\frac{u_i - u_k}{u_i - l_i}(l_i - l_i) + \frac{u_j - u_k}{u_j - l_j}(u_j - l_j) + u_k \\
=&u_j - u_k + u_k \\
=&f(l_1, \cdots, l_{i-1}, l_i, l_{i+1}, \cdots, l_{j-1}, u_j, l_{j+1}, \cdots, l_n)
\end{aligned}
$$

As

$$
\begin{aligned}
\boldsymbol{m} =&\frac{1}{2}(u_1, \cdots, u_{i-1}, u_i, u_{i+1}, \cdots, u_{j-1}, l_j, u_{j+1}, \cdots, u_n) \\
&+\frac{1}{2}(l_1, \cdots, l_{i-1}, l_i, l_{i+1}, \cdots, l_{j-1}, u_j, l_{j+1}, \cdots, l_n)
\end{aligned}
$$

We notice that

$\boldsymbol{a} := (l_1, \cdots, l_{j-1}, u_j, l_{j+1}, \cdots, l_n)$ and $\boldsymbol{b} := (u_1, \cdots, u_{j-1}, l_j, u_{j+1}, \cdots, u_n))$ are the space diagonal of $\times_{i=1}^{n}[l_i, u_i]$, and $\boldsymbol{m} = \frac{1}{2}(\boldsymbol{a} + \boldsymbol{b})$. Similar to the proof in case 2, we can prove that the plane is the neuron-wise tightest linear upper bounding plane.

Lower linear bound:

$l(x_1, \cdots, x_n) = x_j = argmax_i m_i$, and $\forall (x_1, \cdots, x_n) \in \times_{i=1}^{n}[l_i, u_i]$,

$l(\boldsymbol{m}) = f(m_1, \cdots, m_n) \geq l'(\boldsymbol{m}), \forall l \in \mathcal{L}$, hence, $l(x_1, \cdots, x_n)$ is the neuron-wise tightest lower bounding plane.

This completes the proof. □

Table 4. The additional experimental setup and source of neural networks used in experiments.

Dataset	Network	#Nodes	Accuracy	#Properties	ϵ	Source
MNIST	Conv_MaxPool	14592	81.9	81	2/255	ERAN
	Convnet_MaxPool	25274	98.8	96	10/255	Verivital
	CNN, 4 layers	36584	99.0	94	3/255	CNN-Cert
	CNN, 5 layers	52872	99.1	99	2/255	
	CNN, 6 layers	56392	99.1	99	2/255	
	CNN, 7 layers	56592	99.1	91	2/255	
	CNN, 8 layers	56912	99.3	99	2/255	
CIFAR-10	Conv_MaxPool	57020	44.6	48	0.001	ERAN
	CNN, 4 layers	49320	71.3	27	0.001	CNN-Cert
	CNN, 5 layers	71880	71.1	16	0.001	
	CNN, 6 layers	77576	73.8	14	0.001	
	CNN, 7 layers	77776	75.6	29	0.001	
	CNN, 8 layers	78416	68.1	12	0.001	
ModelNet40	16p_Natural	26200	76.8	59	0.005	3DCertify
	32p_Natural	49800	83.2	62	0.005	
	64p_Natural	97000	85.7	64	0.005	
	64p_FGSM	97000	86.0	66	0.01	
	64p_IBP	97000	78.1	60	0.01	

Table 5. The performance of Ti-Lin on CNN-Cert (backsubstitution-based).

Dataset	Network	l_p	Certified Bounds(10^{-5})		Bound Impr.(%)	Average Runtime(min)	
			CNN-Cert	Ti-Lin, on CNN-Cert	vs. CNN-Cert	CNN-Cert	Ti-Lin, on CNN-Cert
MNIST	CNN	l_∞	1318	**1837**	39.4	1.8	1.7
	4 layers	l_2	4427	**6478**	46.3	1.4	1.4
	36584 nodes	l_1	8544	**12642**	48.0	1.4	1.4
	CNN	l_∞	1288	**1817**	41.0	8.4	8.8
	5 layers	l_2	5164	**7359**	42.5	11.9	9.2
	52872 nodes	l_1	10147	**14292**	40.9	10.8	9.5
	CNN	l_∞	1025	**1382**	34.8	20.5	20.9
	6 layers	l_2	3954	**5409**	36.8	20.6	20.4
	56392 nodes	l_1	7708	**10455**	35.6	20.6	20.0
	CNN	l_∞	647	**930**	43.7	24.7	24.6
	7 layers	l_2	2733	**4022**	47.2	25.1	23.8
	56592 nodes	l_1	5443	**8002**	47.0	22.9	22.9
	CNN	l_∞	847	**1221**	44.2	26.5	26.7
	8 layers	l_2	3751	**5320**	41.8	25.0	24.9
	56912 nodes	l_1	7515	**10655**	41.8	23.7	24.2
	LeNet_ReLU	l_∞	1204	**1864**	54.8	0.2	0.2
	3 layers	l_2	6534	**10862**	66.2	0.2	0.2
	8080 nodes	l_1	17937	**30305**	69.0	0.2	0.2
	LeNet_Sigmoid	l_∞	1684	**2042**	21.3	0.3	0.3
	3 layers	l_2	9926	**12369**	24.6	0.3	0.3
	8080 nodes	l_1	26937	**33384**	23.9	0.3	0.3
	LeNet_Tanh	l_∞	613	**817**	33.3	0.3	0.3
	3 layers	l_2	3462	**4916**	42.0	0.3	0.3
	8080 nodes	l_1	9566	**13672**	42.9	0.3	0.3
	LeNet_Atan	l_∞	617	**836**	35.5	0.3	0.3
	3 layers	l_2	3514	**5010**	42.6	0.3	0.3
	8080 nodes	l_1	9330	**13345**	43.0	0.3	0.3
CIFAR-10	CNN	l_∞	108	**129**	19.4	3.1	2.9
	4 layers	l_2	751	**1038**	38.2	2.5	2.5
	49320 nodes	l_1	2127	**3029**	42.4	2.5	2.5
	CNN	l_∞	115	**146**	27.0	13.1	13.0
	5 layers	l_2	953	**1342**	40.8	12.4	12.7
	71880 nodes	l_1	2850	**4087**	43.4	12.3	12.6
	CNN	l_∞	99	**120**	21.2	28.6	28.6
	6 layers	l_2	830	**1078**	29.9	27.6	27.9
	77576 nodes	l_1	2387	**3174**	33.0	27.7	27.4
	CNN	l_∞	66	**83**	25.8	33.4	33.3
	7 layers	l_2	573	**773**	34.9	32.5	32.8
	77776 nodes	l_1	1673	**2303**	37.7	33.6	32.6
	CNN	l_∞	56	**70**	25.0	36.9	37.5
	8 layers	l_2	536	**705**	31.5	37.5	36.6
	78416 nodes	l_1	1609	**2160**	34.2	36.9	37.0
Tiny ImageNet	CNN	l_∞	77	**123**	59.7	184.9	184.0
	7 layers	l_2	580	**939**	61.9	184.4	183.3
	703512 nodes	l_1	1747	**2875**	64.6	193.6	183.9

7.4. Experiment setup

In this subsection, we present some experiment setups of Section 5 in detail. Concretely, we list the number of nodes, the sources of networks, the number of Properties to be verified, and the perturbation range ϵ in Table 4. Following the setting of ERAN, we generate properties for all networks by the correctly classified inputs in the first 100 inputs. We evaluate our method on four datasets, including MNIST, a dataset of 28×28 handwritten digital images in 10 classes, CIFAR-10, a dataset

Table 6. The performance of Ti-Lin and MaxPool2ReLU on α, β-CROWN verifier.

		Certified accuracy(%)		Avg. time of safe instances(s)		Speedup
Network	Radius	MaxPool2ReLU	Ti-Lin	MaxPool2ReLU	Ti-Lin	vs. MaxPool2ReLU
MNIST_Conv_MaxPool	0.008	56.8	**65.4**	32.5	**17.3**	1.9
	0.009	49.4	**60.5**	46.3	**25.2**	1.8
	0.010	40.7	**54.3**	65.9	**32.5**	2.0
ConvNet_MaxPool	0.020	60.0	**80.0**	191.0	**0.6**	318.3
	0.030	15.0	**50.0**	377.5	**37.5**	10.1
	0.040	0.0	**20.0**	-	**44.2**	-
		Timeout rate(%)		Avg. time of all instances(s)		Speedup
Network	Radius	MaxPool2ReLU	Ti-Lin	MaxPool2ReLU	Ti-Lin	vs. MaxPool2ReLU
MNIST_Conv_MaxPool	0.008	37.0	**28.4**	494.9	**65.5**	7.6
	0.009	43.2	**32.1**	667.8	**77.4**	8.6
	0.010	50.6	**37.0**	770.1	**88.7**	8.7
ConvNet_MaxPool	0.020	20.0	**0.0**	294.6	**0.7**	420.9
	0.030	60.0	**20.0**	549.5	**104.7**	5.2
	0.040	45.0	**5.0**	566.1	**47.5**	11.9

Table 7. The performance of Ti-Lin and MaxPool2ReLU on ERAN framework.

		Certified accuracy(%)		Averaged Time(s)		Speedup
Network	Radius	MaxPool2ReLU	Ti-Lin	MaxPool2ReLU	Ti-Lin	vs. MaxPool2ReLU
MNIST_Conv_MaxPool	0.006	24.7	**69.1**	645.3	**18.5**	34.9
	0.007	11.1	**64.2**	776.2	**25.7**	30.2
	0.008	7.4	**45.7**	882.3	**36.9**	23.9
ConvNet_MaxPool	0.020	0.0	**65.0**	984.3	**86.5**	11.4
	0.030	0.0	**50.0**	1022.5	**153.5**	6.7
	0.040	0.0	**35.0**	973.6	**149.6**	6.5

of 60,000 $32 \times 32 \times 3$ images in 10 classes, e.g., airplane, bird, and ship, Tiny ImageNet [13], a dataset of 100,000 $64 \times 64 \times 3$ images in 200 classes, and ModelNet40, a dataset of 12,311 pre-aligned shapes from 40 categories.

7.5. Additional experiments

In this subsection, we conduct some additional experiments to further illustrate (I) the performance of Ti-Lin on CNN-Cert, a back-substitution-based verifier. (II) We compare Ti-Lin to OSIP to illustrate the superiority of tight linear approximation for MaxPool over the MaxPool2ReLU transformation.

7.5.1. Results (I): performance on CNN-Cert

To compare Ti-Lin to CNN-Cert fairly, we implement Ti-Lin on CNN-Cert. We follow the metrics used in CNN-Cert. We use the certified robustness bound introduced in Section 3 as the tightness metric and the average computation time as the efficiency metric. As for the improvement of tightness, we use $\frac{100(\epsilon_l' - \epsilon_l)}{\epsilon_l}\%$ to quantify the percentage of improvement, where ϵ_l' and ϵ_l represent the average certified lower bounds certified by Ti-Lin and CNN-Cert, respectively. We evaluate Ti-Lin and CNN0Cert on 10 inputs for all CNNs in Table 5. The initial perturbation range is 0.005. Testing on 10 inputs can sufficiently evaluate the performance of verification methods, as it is shown that the average certified results of 1000 inputs are similar to 10 images [5]. The results are shown in Table 5. The results indicates that, under different networks, datasets, perturbation norm(l_1, l_2, l_∞), Ti-Lin's certified bounds achieved better performance than CNN-Cert without incurring additional time overhead. Specifically, Ti-Lin exhibits larger robustness bounds compared to CNN-Cert with improvements of up to 69.0, 43.3, 64.6% on the MNIST, CIFAR-10, and Tiny ImageNet datasets, respectively.

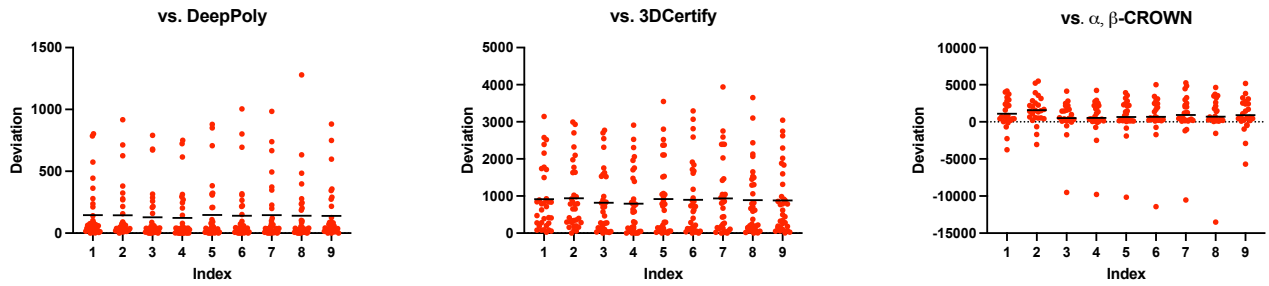


Figure 3. Visualization of the global lower bounds verified by DeepPoly, 3DCertify, α, β-CROWN, and Ti-Lin. Red dots represent the deviation of the global bounds $\boldsymbol{L} - \boldsymbol{L'}$, where $\boldsymbol{L}$ and $\boldsymbol{L'}$ represent the global bounds of Ti-Lin and other methods testing on 100 inputs, respectively. Black lines represent the mean of the deviations.

7.5.2. Results (II): comparison to MaxPool2ReLU transformation method

Notably, as OSIP [26] is not open-sourced, we use another alternative method, called MaxPool2ReLU for evaluation. Concretely, we follow OSIP to transform the MaxPool-based network into a ReLU-based network. ERAN, employing advanced multi-ReLU relaxation for ReLU, and α, β-CROWN, leveraging the branch-and-bound technique for ReLU neurons, are two cutting-edge verifiers designed for handling the ReLU layer. Therefore, we use the results of the transformed networks tested by ERAN and α, β-CROWN as the alternative results of OSIP, denoted as MaxPool2ReLU. The results are shown in Table 6 and Table 7. The results show that MaxPool2ReLU transformation would lead to coarse certified results and much more time consumption. Concretely, Ti-Lin computes higher certified accuracy with up to 35% and 65% improvement than MaxPool2ReLU on α, β-CROWN and ERAN, respectively. Further, Ti-Lin can accelerate the verification process with up to 420.9$\times$ and 34.9$\times$ speedup regarding the average time of all instances on α, β-CROWN and ERAN, respectively. This is because the transformation would make the network deeper, leading to the cumulative overapproximation region and much time consumption to verify. For example, when the pool size is 2×2, one MaxPool layer can be transformed into three ReLU layers and three affine layers. Thus, Ti-Lin is much faster and tighter than MaxPool2ReLU. Especially when verifying ConvNet_MaxPool, the verified accuracy of MaxPool2ReLU (ERAN) is zero across all perturbation radii, while Ti-Lin can at least verify 35% inputs when the perturbation radius is 0.040, respectively.

7.5.3. Result (III): Evaluation on global lower bounds

According to Equation 2, we decide whether the perturbed region is safe based on $l_t^K - u_j^K \geq 0, j \neq t, j \in [n_K]$. Therefore, the global lower bounds $\boldsymbol{L} := (l_t^K - u_j^K), j \neq t, j \in [n_K]$ is the raw criterion to evaluate the tightness. To further illustrate the advantages of the neuron-wise tightness over other linear bounds, we analyze the global lower bounds of the last layer computed by Ti-Lin and other methods, including DeepPoly, 3DCertify, and optimized linear bounds, used in α, β-CROWN. In Figure 3, we show the deviation of the global lower bounds of Ti-Lin and the baseline on CIFAR_Conv_MaxPool. The x-axis represents the index of the global lower bound (labels without the true label), and the y-axis represents the deviations between the global lower bounds. As shown in Figure 3, Ti-Lin has larger global bounds on all inputs than DeepPoly and 3DCertify and on most inputs than α, β-CROWN. Further, the mean of the deviations $\boldsymbol{L} - \boldsymbol{L'}$ are all larger than zero. It reveals that the neuron-wise tightest linear bounds can bring tighter output bounds than other methods. Consequently, Ti-Lin can certify much larger certified accuracy than these methods in Table 1 and 3.

According to the BaB design of α, β-CROWN, we compare the global lower bound for verifying the property 0(true label agaist label 0) on ConvNet_MaxPool, with the results illustrated in Figure 4. Initially, at BaB round = 0, MaxLin achieves a higher global lower bound compared to Ti-Lin. This is attributed to MaxLin's block-wise tightest approach, which is specifically designed for ReLU + MaxPool blocks, giving it an early advantage in representing tighter bounds before the BaB analysis progresses. However, as the BaB process progresses, the α, β-CROWN verification framework employs plane-cutting techniques for ReLU neurons. This enables Ti-Lin's neuron-wise tightest method to leverage its finer granularity, which significantly improves the global lower bound in later BaB rounds. Consequently, Ti-Lin surpasses MaxLin, achieving a much higher global lower bound in the later stages of analysis. This comparison demonstrates that while MaxLin provides better bounds in the initial stages, Ti-Lin's more precise neuron-wise tightness proves to be advantageous for verifying robustness after iterative BaB analysis.

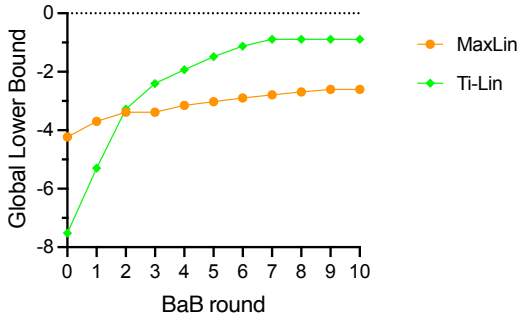


Figure 4. Visualization of the global lower bound verified by MaxLin and Ti-Lin, both of which are built upon the framework of α, β-CROWN.

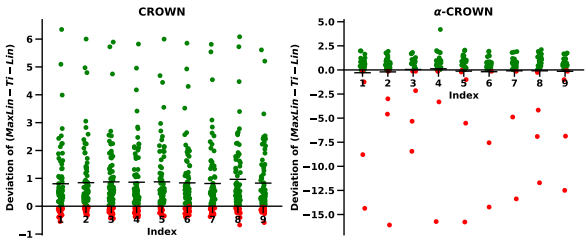


Figure 5. The deviation in global bounds between MaxLin and Ti-Lin when using the CROWN and α-CROWN frameworks.

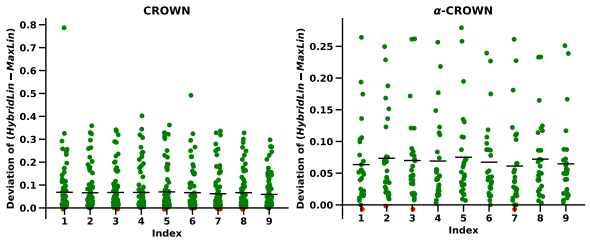


Figure 6. The deviation in global bounds between Hybrid-Lin and MaxLin when using the CROWN and α-CROWN frameworks.

7.5.4. Result (IV): Hybrid-Lin: a combination of Ti-Lin and MaxLin

MaxLin and Ti-Lin make distinct but non-contradictory claims. MaxLin provides the **block-wise tightest** upper linear bound when the ReLU's upper linear bound is $u(x) = \frac{u}{u-l}(x-l)$, while Ti-Lin, being **neuron-wise tightest**, also achieves block-wise tightest bounds when the ReLU's upper linear bound is $u(x) = 0$ or $u(x) = x$. Thus, Ti-Lin outperforms MaxLin when ReLU's upper bound incurs no precision loss. Rather than competing, they complement each other in achieving optimal bounding precision. To illustrate this, we introduce HybridLin, which uses MaxLin when one of the ReLU' upper linear bound is $u(x) = \frac{u}{u-l}(x-l)$ while Ti-Lin when all the ReLU's upper linear bound is $u(x) = 0$ or $u(x) = x$. In Figures 5 and 6, green points indicate positive deviation, red indicate negative. Figure 5 shows that when ReLU's upper linear bound incurring precision loss, MaxLin's performance varies, underscoring the importance of Ti-Lin in achieving tighter results. Figure 6 shows that Hybrid-Lin, using Ti-Lin when ReLU's upper linear bound is precise, consistently achieves tighter bounds than MaxLin.