An RNN-Based Framework for the MILP Problem in Robustness Verification of Neural Networks

Hao Xue[1], Xia Zeng[1,2], Wang Lin[3], Zhengfeng Yang[1(✉)], Chao Peng[1], and Zhenbing Zeng[4]

[1] Shanghai Key Lab of Trustworthy Computing, East China Normal University, Shanghai, China
hxue@stu.ecnu.edu.cn, {zfyang,cpeng}@sei.ecnu.edu.cn
[2] School of Computer and Information Science, Southwest University, Chongqing, China
xzeng0712@swu.edu.cn
[3] School of Information Science and Technology, Zhejiang Sci-Tech University, Hangzhou, China
linwang@zstu.edu.cn
[4] Department of Mathematics, Shanghai University, Shanghai, China
zbzeng@shu.edu.cn

Abstract. Robustness verification of deep neural networks is becoming increasingly crucial for their potential use in many safety-critical applications. Essentially, the problem of robustness verification can be encoded as a typical Mixed-Integer Linear Programming (MILP) problem, which can be solved via branch-and-bound strategies. However, these methods can only afford limited scalability and remain challenging for verifying large-scale neural networks. In this paper, we present a novel framework to speed up the solving of the MILP problems generated from the robustness verification of deep neural networks. It employs a semi-planet relaxation to abstract ReLU activation functions, via an RNN-based strategy for selecting the relaxed ReLU neurons to be tightened. We have developed a prototype tool L2T and conducted comparison experiments with state-of-the-art verifiers on a set of large-scale benchmarks. The experiments show that our framework is both efficient and scalable even when applied to verify the robustness of large-scale neural networks.

Keywords: Robustness verification · Learning methods · Semi-planet relaxation · Neural networks.

1 Introduction

While the deep neural network has been widely applied to various tasks and achieved outstanding performance in recent years, it also encounters robustness problems from many adversarial examples [14, 30], which could raise serious consequences in safety-critical systems such as autonomous driving, aircraft flight control, and nuclear systems. For example, a slightly perturbed image could make an autonomous car incorrectly classify a white truck to cloud and cause a crash accident [13]. Consequently, robustness verification of neural networks becomes increasingly crucial, which aims to check that the outputs for all inputs in a given domain have the correct corresponding label.

Several methods [3, 16, 28] have been proposed to use abstraction for achieving the robustness. With the wide use of the ReLU activation function in neural networks, it is more useful in practice to investigate the robustness verification problem [20].

Due to the piece-wise linear structure of ReLU networks, the problem of robustness verification can be rewritten into a typical Mixed-Integer Linear Programming (MILP) problem, which can be solved via branch-and-bound strategies [7]. However, for large-scale ReLU networks, solving the MILP problems stemming from robustness verification still remains a challenge. To this end, several convex relaxation approaches have been studied for approximating ReLU activation functions [5, 9, 22, 24, 26]. Nevertheless, as the error accumulation arising from the over-relaxation is often hard to estimate, these approaches may fail to prove the robustness.

Meanwhile, machine learning methods have already been used in solving MILP problems. In [1] and [15], regression and ranking approaches are used to assign a branching score to each potential branching choice. Bayesian optimization is also applied in [11] to learn verification policies. The scheme in [21] captures the structure of a neural network and makes tightening decisions through a graph neural network. Recently, symbolic interval propagation and linear programming are used to improve the verification performance [17]. However, the above schemes are either limited to specific neural networks or relied heavily on hand-designed features.

To address the issues mentioned above, in this paper, we propose a novel framework to speed up the solving of MILP problems generated from the robustness verification of neural networks. Since the relaxation of ReLU activation functions may result in failure of the original robustness verification, we propose a semi-planet relaxation, which adopts an RNN-based strategy for selecting the relaxed ReLU neurons to be tightened. Our framework entails an iterative process to gradually repair failures caused by over-relaxation, hence it can perform effective verification while ensuring the efficiency of the MILP problem solving.

To summarize, the main contributions of this paper are:

- We present a general scheme, which augments the original neural network by additional layers with one output neuron, and transforms the verification problem into the output positiveness checking of the augmented network. Thus we transform the robustness verification problem into a single MILP problem, making our RNN-based solving method much easier to implement and scale.
- We build an iterative framework that enhances the efficiency of robustness verification through an intelligent neuron selection strategy by an RNN to gradually tighten the relaxed MILP solution.
- Our RNN embraces scalability by utilizing the network's structure information, and the trained RNN is generalizable for robustness verification on different network structures. We also provide an approach to generate the training dataset, which makes RNN much easier to train and implement.
- We conduct comparison experiments with state-of-the-art verifiers on large-scale benchmarks, the results show that our framework has excellent applicability: it outperforms most other verifiers in terms of average solving time and has lower timeout rate in most cases.

2 Preliminaries

In this section, we provide the background on Deep Neural Networks (DNNs) and describe the robustness verification of DNNs.

2.1 Deep Neural Networks

A deep neural network $\mathcal{N}$ is a tuple $\langle \mathcal{X}, \mathcal{H}, \Phi \rangle$, where $\mathcal{X} = \{\mathbf{x}^{[0]}, \ldots, \mathbf{x}^{[n]}\}$ is a set of layers, $\mathcal{H} = \{H_1, \ldots, H_n\}$ consists of affine functions between layers, and $\Phi = \{\phi_1, \ldots, \phi_n\}$ is a set of activation functions. More specifically, $\mathbf{x}^{[0]}$ is the input layer, $\mathbf{x}^{[n]}$ is the output layer, and $\mathbf{x}^{[1]}, \ldots, \mathbf{x}^{[n-1]}$ are called hidden layers. Each layer $\mathbf{x}^{[i]}, 0 \leq i \leq n$ is associated with an s_i-dimensional vector space, in which each dimension corresponds to a neuron. For each $1 \leq i \leq n$, the affine function is written as $H_i(\mathbf{x}^{[i-1]}) = W^{[i]}\,\mathbf{x}^{[i-1]} + \mathbf{b}^{[i]}$, where $W^{[i]}$ and $\mathbf{b}^{[i]}$ are called the weight matrix and the bias vector, respectively. Furthermore, for each layer $\mathbf{x}^{[i]}$, the value of each neuron in the hidden layer is assigned by the affine function H_i for the values of neurons in the previous layer, and then applying the activation function ϕ_i, i.e., $\mathbf{x}^{[i]} = \phi_i(W^{[i]}\,\mathbf{x}^{[i-1]} + \mathbf{b}^{[i]})$, with the activation function ϕ_i being applied element-wise.

In this paper, we focus on DNNs with Rectified Linear Unit (ReLU) activation functions [12], defined as $\mathrm{ReLU}(x) = \max(0, x)$. From the mapping point of view, the i-th layer can be seen as the mapping image of the $(i-1)$-th layer. For each non-input layer $\mathbf{x}^{[i]}$, $1 \leq i \leq n$, we can define a function $f^{[i]} : \mathbb{R}^{s_{i-1}} \to \mathbb{R}^{s_i}$, with

$$f^{[i]}(\mathbf{x}) = \phi_i(W^{[i]}\mathbf{x} + \mathbf{b}^{[i]}), \quad 1 \leq i \leq n. \tag{1}$$

A DNN $\mathcal{N}$ is expressed as the composition function $f : \mathbb{R}^{s_0} \to \mathbb{R}^{s_n}$, i.e.,

$$f(\mathbf{x}) = f^{[n]}(f^{[n-1]}(\cdots(f^{[1]}(\mathbf{x})))). \tag{2}$$

Given an input $\mathbf{x}^{[0]}$, the DNN $\mathcal{N}$ assigns the label selected with the largest logit of the output layer $\mathbf{x}^{[n]}$, that is, $k = \arg\max(f(\mathbf{x}^{[0]})) = \underset{1 \leq j \leq s_n}{\arg\max}(\mathbf{x}_j^{[n]})$.

2.2 Robustness Verification of DNNs

We begin with formally defining the notion of robustness, and introducing the robustness verification of DNNs.

Given a DNN $\mathcal{N}$ with its associated function f and an input point $\mathbf{x}_c$, we now define ϵ-robustness for a DNN $\mathcal{N}$ on $\mathbf{x}_c$. Let $B(\mathbf{x}_c, \epsilon)$ be the ℓ_∞-ball of radius $\epsilon \in \mathbb{R}_{>0}$ around the point $\mathbf{x}_c$, i.e., $B(\mathbf{x}_c, \epsilon) = \{\mathbf{x}^{[0]} \in \mathbb{R}^{s_0} \mid \|\mathbf{x}^{[0]} - \mathbf{x}_c\|_\infty \leq \epsilon\}$.

We say that an input domain $B(\mathbf{x}_c, \epsilon)$ has the same label if any input $\mathbf{x}^{[0]}$ chosen from $B(\mathbf{x}_c, \epsilon)$ has the same label as $\mathbf{x}_c$ has. This property that the ϵ-ball around the point having the same label is called as ϵ-*robustness*, i.e.,

$$\arg\max(f(\mathbf{x}^{[0]})) = \arg\max(f(\mathbf{x}_c)),\ \forall \mathbf{x}^{[0]} \in B(\mathbf{x}_c, \epsilon). \tag{3}$$

Determining the ϵ-robustness of $\mathbf{x}_c$ with the label k, can be transformed into the following equivalent optimization problem

$$\left.\begin{array}{ll} p^* = \min\limits_{\mathbf{x},\hat{\mathbf{x}}}\{\ \mathbf{x}_k^{[n]} - \mathbf{x}_{k'}^{[n]},\ \forall k' \neq k\ \} & \\ \text{s.t.} \quad \mathbf{x}^{[0]} \in B(\mathbf{x}_c, \epsilon), & \\ \quad\ \hat{\mathbf{x}}^{[i]} = W^{[i]}\mathbf{x}^{[i-1]} + \mathbf{b}^{[i]}, & i = 1, \ldots, n, \\ \quad\ \mathbf{x}^{[i]} = \phi_i(\hat{\mathbf{x}}^{[i]}), & i = 1, \ldots, n. \end{array}\right\} \tag{4}$$

Remark that $\mathcal{N}$ with respect to $B(\mathbf{x}_c, \epsilon)$ is robust if and only if the optimum of (4) is positive, i.e., $p^* > 0$.

The main challenge of optimization solving (4) is to handle ReLU activation functions, which bring non-linearities to the problem of robustness verification. Seeking the optimal solution of (4) is an arduous task, as the optimization problem is generally NP-hard [18]. An effective technique to eliminate the non-linearities is to encode them with the help of binary variables. Let $\mathbf{l}^{[i]}$ and $\mathbf{u}^{[i]}$ be lower bounds and upper bounds on $\hat{\mathbf{x}}^{[i]}$, respectively. The ReLU activations $\mathbf{x}^{[i]} = \phi^{[i]}(\hat{\mathbf{x}}^{[i]})$ can be encoded by the following constraints:

$$\mathcal{I}(\mathbf{x}^{[i]}) \triangleq \begin{cases} \mathbf{x}^{[i]} \succcurlyeq \mathbf{0}, \quad \mathbf{x}^{[i]} \succcurlyeq \hat{\mathbf{x}}^{[i]}, \\ \mathbf{u}^{[i]} \odot \mathbf{z}^{[i]} \succcurlyeq \mathbf{x}^{[i]}, \\ \hat{\mathbf{x}}^{[i]} - \mathbf{l}^{[i]} \odot (\mathbf{1} - \mathbf{z}^{[i]}) \succcurlyeq \mathbf{x}^{[i]}, \\ \mathbf{z}^{[i]} \in \{0,1\}^{s_i}, \end{cases} \tag{5}$$

where s is the binary vector with length s_i, $\odot$ denotes the Hadamard product of matrices, and the generalized inequality $\succcurlyeq$ denotes the componentwise inequality between vectors. Therefore, the robustness verification problem (4) can be transformed into an equivalent Mixed-Integer Linear Programming (MILP) problem [6]:

$$\left.\begin{array}{ll} p^* = \min\limits_{\mathbf{x},\hat{\mathbf{x}},\mathbf{z}}\{\ \mathbf{x}_k^{[n]} - \mathbf{x}_{k'}^{[n]},\ \forall k' \neq k\ \} & \\ \text{s.t.} \quad \mathbf{x}^{[0]} \in B(\mathbf{x}_c, \epsilon), & \\ \quad\ \hat{\mathbf{x}}^{[i]} = W^{[i]}\mathbf{x}^{[i-1]} + \mathbf{b}^{[i]}, & i = 1, \ldots, n, \\ \quad\ \mathcal{I}(\mathbf{x}^{[i]}), & i = 1, \ldots, n. \end{array}\right\} \tag{6}$$

It is easy to verify that $\mathbf{z}_j^{[i]} = 0 \Leftrightarrow \mathbf{x}_j^{[i]} = 0$ and $\mathbf{z}_j^{[i]} = 1 \Leftrightarrow \mathbf{x}_j^{[i]} = \hat{\mathbf{x}}_j^{[i]}$.

3 Semi-Planet Relaxation for MILP Problems

3.1 Formulating the Minimum Function in MILP Model

The objective function of the optimization problem (6) is piecewise-linear [7]. There are some approaches [8, 10, 25] making it amenable to mathematical programming solvers, but either discard the network structure or sacrifice the scalability. In the following, we propose an approach for rewriting the objective function with multiple neurons into an equivalent single-neuron setting by adding a few layers at the end of the network $\mathcal{N}$. This allows us to preserve network structure and ensure the scalability of solving the problem. Note that only linear layers and ReLU activation functions are added for the sake of simplicity.

Before doing this, we first add a new layer to represent the difference between $\mathbf{x}_k^{[n]}$ and all other output neurons:

$$\begin{aligned} \mathbf{x}^{[n+1]} &= \phi_{n+1}\left(W^{[n+1]}\mathbf{x}^{[n]} + \mathbf{b}^{[n+1]}\right), \\ W^{[n+1]} &= \mathbf{1}^{\mathrm{T}}\mathbf{e}_k - I, \quad \mathbf{b}^{[n+1]} = \mathbf{0}, \end{aligned} \tag{7}$$

where $\mathbf{e}_k$ is one-hot encoding vector for label k, I is an identity matrix. Now, the remaining task is to compute the minimum of $\mathbf{x}^{[n+1]}$. For simplicity, we illustrate how to use the neural network structure to represent *Min Function* of two elements,

$$\min(a,b) = \frac{a+b-|a-b|}{2} = \frac{\phi(a+b)-\phi(-a-b)-\phi(a-b)-\phi(b-a)}{2}, \tag{8}$$

that is, $\min(a,b) = W_2 \cdot \phi(W_1 \cdot \begin{bmatrix} a \\ b \end{bmatrix})$, where

$$W_1 = \begin{bmatrix} 1 & -1 & 1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}^{\mathrm{T}}, \quad W_2 = \left[\tfrac{1}{2}, -\tfrac{1}{2}, -\tfrac{1}{2}, -\tfrac{1}{2}\right]. \tag{9}$$

By calling the above process recursively, it is easy to build a neural network to express *Min Function* for greater than 2 elements. Therefore, computing the minimum of s_{n+1} neurons, can be represented as a neural network $\mathcal{N}_g$,

$$g(\mathbf{x}) = W_2 g_{\lceil \log_2 s_{n+1} \rceil}\left(\cdots\left(g_2\left(g_1(\mathbf{x})\right)\right)\right), \tag{10}$$

where $g_j(\mathbf{x}) = \phi_{n+1+j}\left(W^{[n+1+j]}\mathbf{x} + \mathbf{b}^{[n+1+j]}\right)$, $j = 1, \ldots, \lceil \log_2 s_{n+1} \rceil$, which halves the input vector space by finding the minimum in pairs.

Combining $\mathcal{N}$ and $\mathcal{N}_g$ can produce an augmented network, denoted by $\widehat{\mathcal{N}}$, which has $m = n + 1 + \lceil \log_2 s_{n+1} \rceil$ layers. Now, the original robustness verification problem can be transformed into that of determining the positiveness of the output of the augmented network $\widehat{\mathcal{N}}$. Therefore, the problem (6) can be rewritten as an equivalent MILP problem:

$$\left.\begin{array}{lll} J_{\mathcal{I}}^* = \min\limits_{\mathbf{x},\hat{\mathbf{x}},\mathbf{z}} & W_2\mathbf{x}^{[m]} & \\ \text{s.t.} & \mathbf{x}^{[0]} \in B(\mathbf{x}_c, \epsilon), & \\ & \hat{\mathbf{x}}^{[i]} = W^{[i]}\mathbf{x}^{[i-1]} + \mathbf{b}^{[i]}, & i = 1, \ldots, m, \\ & \mathcal{I}(\mathbf{x}^{[i]}), & i = 1, \ldots, m. \end{array}\right\} \tag{11}$$

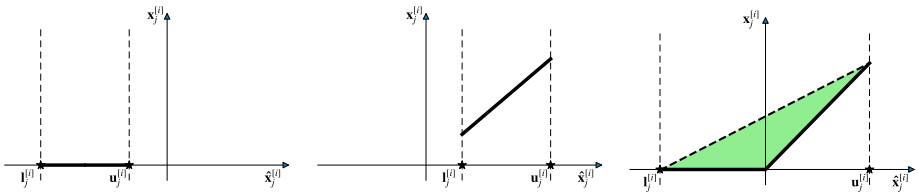
Notably, the optimum $J_{\mathcal{I}}^*$ is positive if and only if the original network $\mathcal{N}$ with respect to $B(\mathbf{x}_c, \epsilon)$ is robust. Hereafter, we abbreviate the optimization problem (11) as following

$$J_{\mathcal{I}}^* = \min\ J\left(\widehat{\mathcal{N}}, \mathcal{I}(X)\right), \tag{12}$$

where $\mathcal{I}(X)$ denotes the binary constraints for the set of ReLU neurons X of $\widehat{\mathcal{N}}$.

3.2 Relaxation and Tightening for MILP Constraints

When encountering practical neural networks, due to binary constraints on overwhelming $\mathcal{I}(\mathbf{x}_j^{[i]})$, how to solve MILP problem (12) is the key challenge. Planet relaxation [9]



(a) $\mathbf{u}_j^{[i]} \leq 0$ (b) $\mathbf{l}_j^{[i]} \geq 0$ (c) Otherwise

Fig. 1: The form of $\mathcal{P}(\mathbf{x}_j^{[i]})$ is contingent on the value of $\mathbf{u}_j^{[i]}$ and $\mathbf{l}_j^{[i]}$.

has been widely used for ReLU neurons relaxation [5, 6, 21] due to its tightness of formulation. Applying planet relaxation to ReLU activation functions $\mathbf{x}_j^{[i]} = \phi(\hat{\mathbf{x}}_j^{[i]})$ may yield the following over-approximation with linear inequalities,

$$\mathcal{P}(\mathbf{x}_j^{[i]}) \triangleq \begin{cases} \mathbf{x}_j^{[i]} = 0, & \text{if } \mathbf{u}_j^{[i]} \leq 0 \\ \mathbf{x}_j^{[i]} = \hat{\mathbf{x}}_j^{[i]}, & \text{if } \mathbf{l}_j^{[i]} \geq 0 \\ \mathbf{x}_j^{[i]} \geq 0, \mathbf{x}_j^{[i]} \geq \hat{\mathbf{x}}_j^{[i]}, \mathbf{x}_j^{[i]} \leq \dfrac{\mathbf{u}_j^{[i]}(\hat{\mathbf{x}}_j^{[i]} - \mathbf{l}_j^{[i]})}{\mathbf{u}_j^{[i]} - \mathbf{l}_j^{[i]}}. & \text{otherwise} \end{cases} \tag{13}$$

Fig. 1 is depicted to show the planet relaxation of a ReLU unit.

By introducing $\mathcal{P}(\mathbf{x}_j^{[i]})$ to relax $\mathcal{I}(\mathbf{x}_j^{[i]})$, the MILP problem (12) is transformed into the following LP problem:

$$J_{\mathcal{P}}^* \;=\; \min\; J\left(\widehat{\mathcal{N}}, \mathcal{P}(X)\right), \tag{14}$$

where $\mathcal{P}(X)$ denotes the planet relaxation for the set of ReLU neurons X of $\widehat{\mathcal{N}}$.

Remark 1. In comparison with the MILP problem (12), the feasible set of the LP problem (14) is a superset of that of (12). Consequently, the optimum of (14) is a lower bound of the one of (12), i.e., $J_{\mathcal{P}}^* \leq J_{\mathcal{I}}^*$. Therefore, the relaxation transformation between the optimization problems derives sufficient conditions for robustness verification of the neural networks.

The gap between $J_{\mathcal{P}}^*$ and $J_{\mathcal{I}}^*$ depends on the tightness of planet relaxation. Unfortunately, the relaxation may yield too conservative optimum $J_{\mathcal{P}}^*$, such that $J_{\mathcal{P}}^* < 0 < J_{\mathcal{I}}^*$ holds, which fails to prove the robustness property for the given neural network. Therefore, how to provide adequate precision and scalability to satisfy the requirement of robustness verification is crucial. In this work, we suggest a semi-planet relaxation scheme where relaxation and tightening processes cooperate to approximate the ReLU neurons. Concretely, when the optimum $J_{\mathcal{P}}^* < 0$, a tightening procedure is performed to tighten the relaxations of some ReLU neurons and recover their original binary constraints. Let $T \subseteq X$ be a subset of ReLU neurons that are chosen to tighten the relaxation, based on the above semi-planet relaxation, (14) can be transformed into

$$J_T^* \;=\; \min\; J\left(\widehat{\mathcal{N}}, \mathcal{I}(T) \wedge \mathcal{P}(X - T)\right). \tag{15}$$

Obviously, the optimum J_T^* of (15) satisfies the inequalities, i.e., $J_{\mathcal{P}}^* \leq J_T^* \leq J_{\mathcal{I}}^*$, which derives that the augmented network $\widehat{\mathcal{N}}$ with respect to $B(\mathbf{x}_c, \epsilon)$ is robust when

$J_T^* > 0$. As the number of tightened neurons in T increases, J_T^* will be closer to $J_{\mathcal{I}}^*$. As a result, the complexity of solving the problem (15) will also grow exponentially. So the challenge is how to balance the relaxation and tightening, that is, how to choose as few neurons as possible to tighten for proving the robustness of $\widehat{\mathcal{N}}$.

4 Learning to Tighten

The MILP problem (12), derived from robustness verification of neural networks, transfers a more tractable linear programming (LP) problem when all ReLU neurons are relaxed by the planet relaxation technique. However, this relaxation of non-linearity may yield too conservative constraints so that the optimum of the resulted LP is negative, which fails to prove the original robustness verification problem.

A natural idea to avoid this situation would be to tight only a part of neurons that have more influence on the verification problem. The key is how to determine those key neurons. In this work, we will propose a framework for solving the MILP problem which incorporates an RNN-based strategy for selecting neurons to be tightened.

4.1 RNN-Based Framework

In this section, we describe an RNN-based framework for solving the MILP problem in robustness verification of neural networks. To build the framework we adopt an iterative process to gradually repair failures caused by over-relaxation, and pursue effective verification while ensuring the efficiency of the MILP problem solving. Specifically, we begin the iteration with a complete relaxed problem in (14) from the original problem, by applying the planet relaxation (12) for all ReLU neurons. If the solution satisfies $J_{\mathcal{P}}^* > 0$, then the robustness of the original problem has been verified. Otherwise, we apply the one-neuron-tighten-test (15) one by one for the set X consisting of all current relaxed neurons, and further solve each updated semi-relaxed problem.

Once the optimal solution satisfies $J_T^* > 0$ after testing on certain set T of the relaxed neurons, we derive that the neural network has been verified to be robust. Otherwise one more neuron is chosen to be tightened. To guarantee the completeness, suppose $\mathbf{x}^*$ is the minimizer of J_T^*, then assign $\mathbf{x}^*$ to (12) can arrive at the upper bound J_T', and the constraint $J_T' > 0$ is incorporated to ensure that our tightening scheme is always available for robustness verification. Meanwhile, if it occurs that $J_T' < 0$ at one stage, it is easy to show that the neural network $\widehat{\mathcal{N}}$ is unrobust.

As shown in Fig. 2, the framework consists of two main functional modules, the MILP-solving module and the neuron-selection module. Given a feed-forward neural network $\mathcal{N}$ with label k, we first augment $\mathcal{N}$ into $\widehat{\mathcal{N}}$ for generating a generic MILP problem (Sec. 3.1), and then relax all or some of the neurons (Sec. 3.2). Next, we check whether $J_T^* > 0$ or $J_T' < 0$ by the MILP-solving module, and terminate the iteration if the condition is satisfied at certain steps. Otherwise, the process enters the neuron-selection module, i.e., the Recurrent Tightening Module (RTM), as shown inside the dashed-round-box in Fig. 2. The RTM module is designed to implement the neuron selection strategy for solving the MILP problem. The trained RTM works by grading

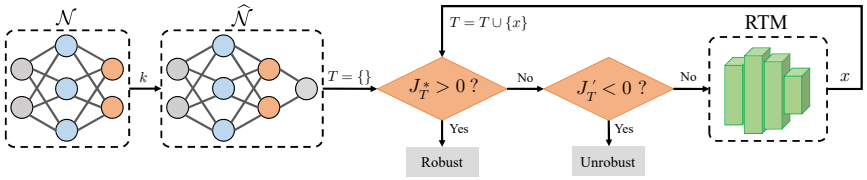


Fig. 2: The Architecture of RNN-based framework.

Algorithm 1 The RNN-based Framework for Robustness Verification

Require: Network $\mathcal{N}$, ℓ_∞-ball $B(\mathbf{x}_c, \epsilon)$, label k

1: $\widehat{\mathcal{N}}, X \leftarrow$ Augment$(\mathcal{N}, B(\mathbf{x}_c, \epsilon), k)$ ▷ Augment network $\mathcal{N}$ and initialize ReLU neurons
2: $T \leftarrow \{\}$ ▷ Initialize an empty tighten set T
3: **while** $\neg(J_T^* > 0) \wedge \neg(J_T' < 0)$ **do**
4: $\quad \Theta \leftarrow \left\{\widehat{\mathcal{N}},\ \mathcal{I}(T),\ \mathcal{P}(X - T)\right\}$ ▷ Gather information of current MILP problem
5: $\quad x \leftarrow$ RTM(Θ) ▷ Select a relaxed neuron to tighten
6: $\quad T \leftarrow T \cup \{x\}$
7: $\quad$ Update J_T^* and J_T' with $J\left(\widehat{\mathcal{N}}, \mathcal{I}(T) \wedge \mathcal{P}(X - T)\right)$
8: **if** $J_T^* > 0$ **then return** Robust
9: **else if** $J_T' < 0$ **then return** Unrobust

each neuron from the currently relaxed collection X for optimal tightening choice. A more detailed explanation on how to train such RTM will be given in Sec. 4.2.

The main data flow of the RNN-based framework is given in Alg. 1. The algorithm takes a neural network $\mathcal{N}$ with its input domain $B(\mathbf{x}_c, \epsilon)$ and labels it by k. In line 1, the algorithm augments $\mathcal{N}$ to $\widehat{\mathcal{N}}$ and extracts all the ReLU neurons X. In line 2, the set T for collecting neurons to tighten is initialized to the empty set. The main loop from Line 3 to Line 8 is to check the robustness condition and do the neuron selection, where line 5 is implemented by the neuron-selection module (cf. RTM in Fig. 2), and line 7 carries out semi-relaxed MILP problem by the MILP-solving module. The algorithm either returns 'Robust' when $J_T^* > 0$ is satisfied in line 8, or reaches 'Unrobust' if $J_T' < 0$ (Line 9).

4.2 Recurrent Tightening Module

The Recurrent Tightening Module (RTM) implements a learned tightening strategy, the key module of the RNN-based framework introduced in Sec. 4.1. It takes the network presented by a numerical tensor as input. The final output of the RTM is a selected neuron which has the highest score based on the result of the RNN whose training process has been illustrated in Sec. 4.3.

The neuron selection process of RTM is shown in Fig. 3. Actually, the input of the module at the beginning is a semi-relaxed MILP problem to be verified, for simplicity we shall express it in an associate network form. Without causing ambiguity,

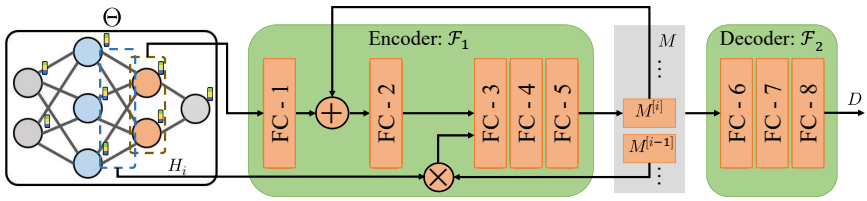


Fig. 3: The flow-chart of RTM.

we denote the input as Θ which carries the parameter information of the current MILP problem, including the structure parameters of the augmented network $\widehat{\mathcal{N}}$ to be verified, collection of all ReLU neurons X, and the tightened ReLU neuron collection T. The green rectangle submodule in Fig. 3, an RNN consists of the Encoder and the Decoder, is designed to score the tighten. The current neurons in X of the network Θ will be graded and recorded layer-by-layer through this RNN-based scorer. The scoring results of all neurons will be recorded in a list denoted by D whose dimension is the same as the number of neurons. In the final stage, the RTM selects the neuron with the highest score as the output.

In RTM, the neuron selecting is processing in the following way. In the initial, a trained RNN equipped with parameter $\boldsymbol{\theta} = \{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2\}$ encodes $\widehat{\mathcal{N}}$ in layer-by-layer wise according to the structure of the input network, and produces a list of feature matrices $M = \{M^{[1]}, \ldots, M^{[m]}\}$ which has the same layer with $\widehat{\mathcal{N}}$, where each $M^{[i]}$ denotes the extracted feature matrix on the i-th layer, and each row of $M^{[i]}$ is the feature vector for a neuron in the layer. To improve the efficiency of the overall algorithm, we have also stored M in order to reuse it in the subsequent encoding process. In addition, we denote the fully-connected Encoder and Decoder by functions $\mathcal{F}_1$ and $\mathcal{F}_2$, respectively. Mathematically, the RTM can be expressed as follows,

$$\begin{aligned} M^{[i]} &= \mathcal{F}_1(\Theta, H_i(M^{[i-1]}), M^{[i]}; \boldsymbol{\theta}_1), \quad i = 1, \ldots, m, \\ D &= \mathcal{F}_2(M; \boldsymbol{\theta}_2), \quad d = \arg\max(D), \quad x = \text{Index}(X, d), \end{aligned} \tag{16}$$

where x is the selected neuron to be added into T for the next tightening.

Remark 2. In addition to update $M^{[i]}$ by Θ, we also allow the updating process reusing $M^{[i]}$ to collect previous information and capturing the local information in $M^{[i-1]}$ by utilizing the feed-forward function H_i of the network $\widehat{\mathcal{N}}$. Therefore, (16) is a recurrent function for tightening the neuron successively.

Remark 3. To increase the scalability of the method for different network structures, the RTM module also encode the structure of network and decode all the neurons in the network layer-by-layer processing.

4.3 Training the RNN in RTM

In this subsection, we explain the two important parts of the RNN training process that is based on supervised learning for making the neuron selection policy.

In the part for generating the training dataset, we consider to characterize the impact of each neuron after being tightened on the current MILP problem to obtain the ground truth of scoring list, which reflects the performance of every neuron to be selected and tightened in the current semi-relaxed problem presented by an associate network $\widehat{\mathcal{N}}$. Namely, given a semi-relaxed problem Θ, we design the following function to measure the improvement of tightening for each tightening decision candidate neuron x,

$$y_T(x) \;=\; \big(J_T^* - \min(J_{T\cup\{x\}}^*,\, 0)\big) \Big/ J_T^*. \tag{17}$$

Here notice that for each neuron $x \in X$, if $x \in T$ then $y_T(x) = 0$, else if $x \notin T$ and $J_{T\cup\{x\}}^* > 0$ then $y_T(x) = 1$, which means the current MILP has been verified after tightening the neuron x and x is the one we prefer to select. Otherwise $0 < y_T(x) < 1$ for all $x \notin T$ and $J_{T\cup\{x\}}^* < 0$, and the value $y_T(x)$ measures the relative improvement for the result of the current MILP after tightening the neuron x; briefly, $y_T(x)$ is the target score of selecting and tightening x. Hereafter, we use Y_T to denote the ground truth of score list of all neurons ordered layer-by-layer.

The second important part for constructing the neuron selection policy is to design a loss function that is best fitting the relax-and-tighten of the MILP related to robustness verification. For a given RNN parameterized by $\boldsymbol{\theta}$, The following two principles are adopted in the design of the loss function.

- For the current MILP which is presented by Θ, the score list D_T as the output of the training RNN should fit the ground truth of score list Y_T well, i.e.,

$$\mathcal{L}_1(\boldsymbol{\theta}) = \|Y_T \odot (D_T - Y_T)\|_1, \tag{18}$$

  where D_T denotes the predicted tightening scores at Θ, i.e. $D_T = \text{RNN}(\Theta; \boldsymbol{\theta})$. And the Hadamard product is introduced in $\mathcal{L}_1$ for assigning higher weights to those with higher scores according to Y_T.
- It is worth noting that tightening the majority of neurons may give similar improvements, which leads to the inconvenience of ranking the predicted scores by RNN. So we introduce a small constant $s > 0$ and construct the following pairwise ranking loss function $\mathcal{L}_2$ to regularize the scores are at least s-apart for each pair of neurons, i.e.,

$$\mathcal{L}_2(\boldsymbol{\theta}) \;=\; \frac{1}{r^2} \sum_{i=1}^{r-1} \sum_{j=i+1}^{r} \max(s - d_T^{[i]} + d_T^{[j]}, 0), \tag{19}$$

  where $d_T^{[i]}$ and $d_T^{[j]}$ are i-th and j-th highest scores predicted in D_T respectively. For the learning efficiency, the pairwise ranking loss $\mathcal{L}_2$ only involves the top r scores in D_T.

In summary, the total loss function is constructed as follows

$$\mathcal{L}(\boldsymbol{\theta}) \;=\; \lambda\mathcal{L}_1(\boldsymbol{\theta}) + (1-\lambda)\mathcal{L}_2(\boldsymbol{\theta}), \tag{20}$$

where $0 < \lambda < 1$ is the parameter to control the weights between $\mathcal{L}_1$ and $\mathcal{L}_2$. In fact, the experiment performance in Sec. 5 reflects that the design of the loss function is reasonable and helps to improve training efficiency, which validates the above analysis.

Algorithm 2 Generating Training Dataset

Require: $\mathcal{N}, \{B(\mathbf{x}_i, \epsilon_i), k_i\}_{i=1,\ldots,P}$
1: $S \leftarrow \{\}$ ▷ Initialize training dataset S
2: **for** $i = 1, \ldots, P$ **do** ▷ Generate dataset using P images
3: $\widehat{\mathcal{N}}, X \leftarrow$ Augment$(\mathcal{N}, B(\mathbf{x}_i, \epsilon_i), k_i)$; $T \leftarrow \{\}$
4: **for** $j = 1, \ldots, t$ **do** ▷ Tighten neurons at most t times
5: $\Theta \leftarrow \left\{\widehat{\mathcal{N}},\ \mathcal{I}(T),\ \mathcal{P}(X - T)\right\}$
6: Construct ground truth score list Y_T with current Θ
7: $S \leftarrow S \cup \{(\Theta, Y_T)\}$ ▷ Add sample (Θ, Y_T) to dataset S
8: Index the optimal neuron x with highest tightening score in Y_T
9: $T \leftarrow T \cup \{x\}$
10: **if** Robustness is verified **then Break**
11: **return** S

Training dataset. We build a subset of CIFAR-10 images to generate dataset S by the following steps. At first, we randomly pick $P = 400$ images and the corresponding perturbation ϵ with adversarial labels determined from [21], which can cover most robustness verification problems on CIFAR-10. An adversarially trained network called Base model is also chosen according to [21]. In the second step, we initialize an empty set T for each image and measure the improvements for all the relaxed ReLU neurons to construct the ground truth of score list Y_T. At last, we refer to the score list Y_T to select the optimal neuron with the highest improvement and put it into T, and then update the current MILP problem to repeat the above process until the recursion depth reaches the threshold t or the robustness is verified. The procedure is given in Alg. 2.

Training details. In the training setting, we have employed `Adam Optimizer` [19] with $\beta_1 = 0.9$ and $\beta_2 = 0.999$ for backward propagation of the RNN. The RNN is trained for 100 epochs with a learning rate of 0.001 and a weight decay of 0.0001. We have also set $r = 35$, $s = 0.04$, and $\lambda = 0.5$. For generating datasets, we have set $t = 20$ and divided the training set and validation set randomly according to the ratio of 7:3. In terms of training accuracy, we rank tightening scores predicted by RNN and consider the top 5 scores that are also in the top 5 of ground truth scores as correct choices since we are mainly interested in tightening neurons with great improvement using the metric defined by (17). Finally, our RNN reaches 71.6% accuracy on training dataset and 63.5% accuracy on validation dataset. We also run an ablation study by removing $\mathcal{L}_2$, which reduced the accuracy of the validation dataset from 63.5% to 59.7%.

5 Experiments

We have implemented a tool in Pytorch [23] called L2T for the neural network robustness verification, which is based on the policy of learning to tighten (L2T for short) in our proposed framework. The code is available at https://github.com/Vampire689/L2T. For demonstrating the advantages of L2T, we first make a comparison with different tighten-based strategies on the dataset provided in [21] (c.f. Sec. 5.1). And L2T has been compared with the current mainstream robustness verification tools on two large-scale robustness verification benchmarks, i.e., the OVAL benchmark [22] and COLT

Table 1: The result of strategies on Easy, Medium and Hard dataset.

Dataset	Average number of neurons tightened			
	Rand	BaBSR	GNN	L2T
CIFAR - Easy	31.65	24.10	22.82	22.53
CIFAR - Medium	60.71	38.67	27.24	24.90
CIFAR - Hard	122.90	48.74	35.74	31.98

benchmark [4] to show the performance of two aspects: (1) the efficiency of L2T on OVAL benchmark for verifying adversarial properties (c.f. Sec. 5.2); (2) the good performance on scalability to verify larger models on the COLT benchmark (c.f. Sec. 5.3).

5.1 Effectiveness of RNN-Based Tightening Strategy

We analyse L2T by comparing with three different tightening strategies: 1) Random selection (Rand), 2) hand-designed heuristic (BaBSR) [6], and 3) GNN learning-based strategy (GNN) [21]. We use the dataset of three different difficulty levels (Easy, Medium and Hard) with the network of 3172 neurons provided in [21] and compute the average number of neurons tightened on solved properties of all strategies. Results of different tightening strategies are listed in Tab. 1. L2T achieves the least average tightening number of neurons for solving the properties compared to other strategies. In particular, L2T outperforms GNN in all levels, showing that our RNN-based framework has learned the empirical information from continuous tightening strategy.

5.2 Performance on the OVAL Benchmark

We evaluate the performance of L2T and five baselines on OVAL benchmark [22] used in [31]. The benchmark consists of sets of adversarial robustness properties on three adversarially trained CIFAR-10 CNNs, which are Base, Wide and Deep models with 3172, 6244 and 6756 ReLUs respectively [22]. All three models are robustly trained using the method from [33]. There are 100 properties for each of the three models, and each property is assigned a non-correct class and associated with a specifically designed perturbation radius $\epsilon \in [0, 16/255]$ on correctly classified CIFAR-10 images. The benchmark is to verify that for the given ϵ, the trained network will not misclassify by labelling the image as the non-correct class. A timeout of one hour per property is suggested. We conduct experimental comparisons with five state of the art verifiers: 1) OVAL [5], a strong verification framework based on Lagrangian decomposition on GPUs. 2) GNN [21], a state-of-the-art tightening-based verifier using a learned graph neural network to imitate optimal ReLU selecting strategy. 3) ERAN [26–29], a scalable verification toolbox based on abstract interpretation. 4) A.set [22], a latest dual-space verifier with a tighter linear relaxation than Planet relaxations. 5) $\alpha\beta$-crown [32, 34, 35], a most recent fast and scalable verifier with efficient bound propagation.

We report the average solving time and the percentage of timeout over all properties in Tab. 2. It can be shown that L2T ranks the top two in average time efficiency among the current popular tools, and L2T performs the best on the Deep model with no

Table 2: The performance of verifiers on the Base, Wide and Deep model.

Network	Metric	OVAL	GNN	ERAN	A.set	$\alpha\beta$-crown	L2T
Base-3172-[0,16/255]	time(s)	835.2	662.7	805.9	377.3	711.5	442.8
	timeout(%)	20	15	5	7	15	9
Wide-6244-[0,16/255]	time(s)	539.4	268.5	607.1	162.7	354.7	228.8
	timeout(%)	12	6	7	3	5	5
Deep-6756-[0,16/255]	time(s)	258.6	80.8	574.6	190.8	55.9	26.1
	timeout(%)	4	1	1	2	1	0

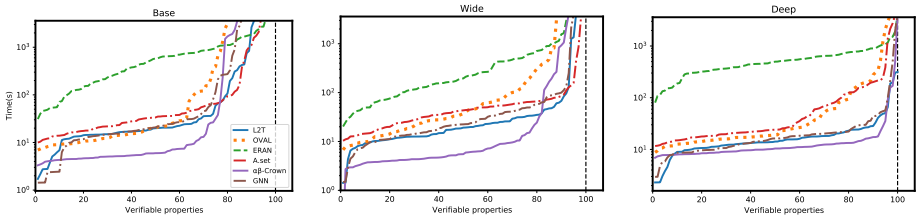


Fig. 4: Cactus plots for Base, Wide and Deep model on OVAL benchmark.

timeout. L2T provides an additional average time saving by 37.8%, 35.5% and 53.3% on three models respectively over the very recent $\alpha\beta$-crown. Although A.set achieves a slightly lower time of verifying properties on the Base and Wide model, L2T leads to 7x faster than A.set on Deep model. Taken together, L2T costs the least total average time on all three models. The cactus plots in Fig. 4 show the increasing solving time with the accumulative verified properties for each verifier. L2T consistently provides great performance on the majority of properties.

5.3 Performance on the COLT Benchmark

Furthermore, we evaluate L2T and six baselines on COLT benchmark [4], which is quite challenging that includes the two largest CIFAR-10 CNNs with tens of thousands ReLUs in VNN-COMP 2021 [2]. The goal is to verify that the classification is robust within an adversarial region defined by l_∞-ball of radius ϵ around an image. The COLT benchmark considers the first 100 images of the test datasets and discards those that are incorrectly classified. The values of ϵ for two CNNs are 2/255 and 8/255 respectively, and the verification time is limited to five minutes per property. To evaluate the performance on COLT benchmark, we introduce another two well-performed verifiers instead the underperforming GNN: 1) Nnenum [3], a recent tool using multi-level abstraction to achieve high-performance verification of ReLU networks. 2) VeriNet [16], a sound and complete symbolic interval propagation-based toolkit for robustness verification.

As shown in Tab. 3, L2T significantly outperforms the majority of verifiers in terms of average solving time and achieves lower timeout rate for two large-scale CIFAR-10 networks. In comparison with A.set, which performed excellent in previous experiments, the average time of L2T on these two networks was reduced by 53.5% and 47.0%, respectively. And the percentage of timeout also drops quite a lot. Although L2T

Table 3: The performance of verifiers on two large-scale CIFAR-10 networks.

Network	Metric	OVAL	Nnenum	VeriNet	ERAN	A.set	$\alpha\beta$-crown	L2T
CIFAR-49402 -2/255	time(s)	85.3	115.2	80.9	87.8	104.5	35.0	48.6
	timeout(%)	26.3	30	23.8	26.3	32.5	3.7	11.2
CIFAR-16634 -8/255	time(s)	124.8	135.0	105.6	111.5	166.1	43.8	88.0
	timeout(%)	27.5	27.5	23.8	16.3	35	6.3	13.8

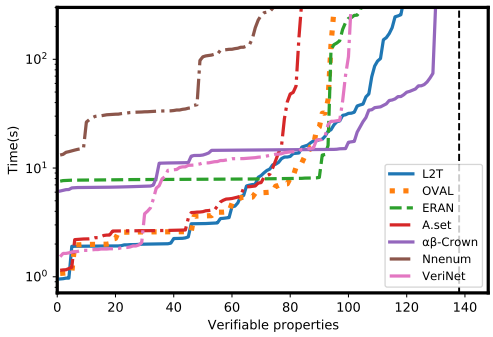


Fig. 5: COLT benchmark cactus plot for L2T and six baselines.

is the second only to $\alpha\beta$-crown in terms of time efficiency, the comprehensive performance of our tool is still competitive as shown in the cactus plot in Fig. 5, which shows the increasing solving time required with accumulative number of verified properties of two networks for each verifier. Within the limited solving time of 10s, L2T verified around 35 more properties than $\alpha\beta$-crown. As verifying robustness properties becomes increasingly complex, $\alpha\beta$-crown verified more properties than L2T, while L2T is more efficient than other verifiers.

6 Conclusion

We have proposed an RNN-based framework for solving MILP problems generated from the robustness verification of deep neural networks. This framework adopts an iterative process to gradually repair failures caused by over-relaxation. Boosted by the intelligent neuron selection strategy through a well-trained RNN in its key recurrent tightening module, our framework can achieve effective verification while ensuring the efficiency of MILP solving. A prototype tool named L2T has been implemented and compared with state-of-the-art verifiers on some large-scale benchmarks. The experimental results demonstrate the efficiency and scalability of our approach for robustness verification on large-scale neural networks.

Acknowledgements This work is supported by the National Natural Science Foundation of China under Grant 12171159, 61902325, and Shanghai Trusted Industry Internet Software Collaborative Innovation Center.

References

1. Alvarez, A.M., Louveaux, Q., Wehenkel, L.: A machine learning-based approximation of strong branching. INFORMS Journal on Computing pp. 185–195 (2017)
2. Bak, S., Liu, C., Johnson, T.: The second international verification of neural networks competition (vnn-comp 2021): Summary and results. ArXiv preprint arXiv:2109.00498 (2021)
3. Bak, S., Tran, H.D., Hobbs, K., Johnson, T.T.: Improved geometric path enumeration for verifying relu neural networks. In: International Conference on Computer Aided Verification. pp. 66–96 (2020)
4. Balunovic, M., Vechev, M.: Adversarial training and provable defenses: Bridging the gap. In: International Conference on Learning Representations (2020)
5. Bunel, R., De Palma, A., Desmaison, A., Dvijotham, K., Kohli, P., Torr, P., Kumar, M.P.: Lagrangian decomposition for neural network verification. In: Conference on Uncertainty in Artificial Intelligence. pp. 370–379 (2020)
6. Bunel, R., Mudigonda, P., Turkaslan, I., Torr, P., Lu, J., Kohli, P.: Branch and bound for piecewise linear neural network verification. Journal of Machine Learning Research pp. 42:1–42:39 (2020)
7. Bunel, R., Turkaslan, I., Torr, P.H.S., Kohli, P., Mudigonda, P.K.: A unified view of piecewise linear neural network verification. In: Neural Information Processing Systems (2018)
8. Dvijotham, K., Stanforth, R., Gowal, S., Mann, T.A., Kohli, P.: A dual approach to scalable verification of deep networks. In: Conference on Uncertainty in Artificial Intelligence. p. 3 (2018)
9. Ehlers, R.: Formal verification of piece-wise linear feed-forward neural networks. In: International Symposium on Automated Technology for Verification and Analysis. pp. 269–286 (2017)
10. Elboher, Y.Y., Gottschlich, J., Katz, G.: An abstraction-based framework for neural network verification. International Conference on Computer Aided Verification pp. 43–65 (2020)
11. Gasse, M., Chetelat, D., Ferroni, N., Charlin, L., Lodi, A.: Exact combinatorial optimization with graph convolutional neural networks. In: Advances in Neural Information Processing Systems. Curran Associates, Inc. (2019)
12. Glorot, X., Bordes, A., Bengio, Y.: Deep sparse rectifier neural networks. In: Proceedings of the fourteenth international conference on artificial intelligence and statistics. pp. 315–323 (2011)
13. Golson, J.: Tesla driver killed in crash with autopilot active, nhtsa investigating. The Verge (2016)
14. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. ArXiv preprint arXiv:1412.6572 (2014)
15. Hansknecht, C., Joormann, I., Stiller, S.: Cuts, primal heuristics, and learning to branch for the time-dependent traveling salesman problem. ArXiv preprint arXiv:1805.01415 (2018)
16. Henriksen, P., Lomuscio, A.: Efficient neural network verification via adaptive refinement and adversarial search. In: European Conference on Artificial Intelligence. pp. 2513–2520 (2020)
17. Henriksen, P., Lomuscio, A.: Deepsplit: An efficient splitting method for neural network verification via indirect effect analysis. In: International Joint Conference on Artificial Intelligence (2021)
18. Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: An efficient smt solver for verifying deep neural networks. In: International Conference on Computer Aided Verification. pp. 97–117 (2017)
19. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. ArXiv preprint arXiv:1412.6980 (2014)

20. Lin, W., Yang, Z., Chen, X., Zhao, Q., Li, X., Liu, Z., He, J.: Robustness verification of classification deep neural networks via linear programming. In: Conference on Computer Vision and Pattern Recognition. pp. 11418–11427 (2019)
21. Lu, J., Kumar, M.P.: Neural network branching for neural network verification. In: International Conference on Learning Representations (2020)
22. Palma, A.D., Behl, H., Bunel, R.R., Torr, P., Kumar, M.P.: Scaling the convex barrier with active sets. In: International Conference on Learning Representations (2021)
23. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: Pytorch: An imperative style, high-performance deep learning library. In: Conference on Neural Information Processing Systems, pp. 8024–8035 (2019)
24. Raghunathan, A., Steinhardt, J., Liang, P.: Semidefinite relaxations for certifying robustness to adversarial examples. In: Conference on Neural Information Processing Systems. p. 10900–10910 (2018)
25. Ruan, W., Huang, X., Kwiatkowska, M.: Reachability analysis of deep neural networks with provable guarantees. In: International Joint Conference on Artificial Intelligence (2018)
26. Singh, G., Ganvir, R., Puschel, M., Vechev, M.: Beyond the single neuron convex barrier for neural network certification. In: Conference on Neural Information Processing Systems (2019)
27. Singh, G., Gehr, T., Mirman, M., Püschel, M., Vechev, M.T.: Fast and effective robustness certification. In: Conference on Neural Information Processing Systems (2018)
28. Singh, G., Gehr, T., Püschel, M., Vechev, M.: An abstract domain for certifying neural networks. Proceedings of the ACM on Programming Languages pp. 1–30 (2019)
29. Singh, G., Gehr, T., Püschel, M., Vechev, M.: Robustness certification with refinement. In: International Conference on Learning Representations (2019)
30. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R.: Intriguing properties of neural networks. In: International Conference on Learning Representations (2014)
31. VNN-COMP: International verification of neural networks competition (vnn-comp). International Conference on Computer Aided Verification (2020)
32. Wang, S., Zhang, H., Xu, K., Lin, X., Jana, S., Hsieh, C.J., Kolter, J.Z.: Beta-CROWN: Efficient bound propagation with per-neuron split constraints for complete and incomplete neural network verification. Advances in Neural Information Processing Systems (2021)
33. Wong, E., Kolter, Z.: Provable defenses against adversarial examples via the convex outer adversarial polytope. In: International Conference on Machine Learning. pp. 5286–5295 (2018)
34. Xu, K., Zhang, H., Wang, S., Wang, Y., Jana, S., Lin, X., Hsieh, C.J.: Fast and Complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. In: International Conference on Learning Representations (2021)
35. Zhang, H., Weng, T.W., Chen, P.Y., Hsieh, C.J., Daniel, L.: Efficient neural network robustness certification with general activation functions. Advances in neural information processing systems (2018)