ARCoSS

LNCS 12651

Jan Friso Groote
Kim Guldstrand Larsen (Eds.)

Tools and Algorithms for the Construction and Analysis of Systems

27th International Conference, TACAS 2021
Held as Part of the European Joint Conferences
on Theory and Practice of Software, ETAPS 2021
Luxembourg City, Luxembourg, March 27 – April 1, 2021
Proceedings, Part I

1 Part I

ETAPS
EUROPEAN JOINT CONFERENCES ON
THEORY & PRACTICE OF SOFTWARE

Springer

OPEN ACCESS

Improving Neural Network Verification through Spurious Region Guided Refinement

Pengfei Yang[1,2], Renjue Li[1,2], Jianlin Li[1,2], Cheng-Chao Huang[3,4], Jingyi Wang[5], Jun Sun[6], Bai Xue[1,2], and Lijun Zhang[1,2,3](✉)

[1] SKLCS, Institute of Software, Chinese Academy of Sciences, Beijing, China
[2] University of Chinese Academy of Sciences, Beijing, China
[3] Institute of Intelligent Software, Guangzhou, China
[4] CAS Software Testing (Guangzhou) Co., Ltd., Guangzhou, China
[5] Zhejiang University NGICS Platform, Hangzhou, China
[6] Singapore Management University, Singapore, Singapore
zhanglj@ios.ac.cn

Abstract. We propose a spurious region guided refinement approach for robustness verification of deep neural networks. Our method starts with applying the DeepPoly abstract domain to analyze the network. If the robustness property cannot be verified, the result is inconclusive. Due to the over-approximation, the computed region in the abstraction may be *spurious* in the sense that it does not contain any true counterexample. Our goal is to identify such spurious regions and use them to guide the abstraction refinement. The core idea is to make use of the obtained constraints of the abstraction to infer new bounds for the neurons. This is achieved by linear programming techniques. With the new bounds, we iteratively apply DeepPoly, aiming to eliminate spurious regions. We have implemented our approach in a prototypical tool DeepSRGR. Experimental results show that a large amount of regions can be identified as spurious, and as a result, the precision of DeepPoly can be significantly improved. As a side contribution, we show that our approach can be applied to verify quantitative robustness properties.

1 Introduction

In the seminal work [34], deep neural networks (DNN) have been successfully applied in Go to play against expert humans. Afterwards, they have achieved exceptional performance in many other applications such as image, speech and audio recognition, self-driving cars, and malware detection. Despite the success of solving these problems, DNNs have also been shown to be often lack of robustness, and are vulnerable to adversarial samples [39]. Even for a well-trained DNN, a small (and even imperceptible) perturbation may fool the network. This is arguably one of the major obstacles when we deploy DNNs in safety-critical applications like self-driving cars [42], and medical systems [33].

It is thus important to guarantee the robustness of DNNs for safety-critical applications. In this work, we focus on (local) robustness, i.e., given an input and a manipulation region around the input (which is usually specified according to a certain

J. F. Groote and K. G. Larsen (Eds.): TACAS 2021, LNCS 12651, pp. 389–408, 2021.
https://doi.org/10.1007/978-3-030-72016-2_21

norm), we verify that a given DNN never makes any mistake on any input in the region. The first work on DNN verification was published in [30], which focuses on DNNs with sigmoid activation functions with a partition-refinement approach. In 2017, Katz et al. [20] and Ehlers [10] independently implemented Reluplex and Planet, two SMT solvers to verify DNNs with the ReLU activation function on properties expressible with SMT constraints. Since 2018, abstract interpretation has been one of the most popular methods for DNN verification in the lead of AI^2 [13], and subsequent works like [36,37,23,1,35,28,24] have improved AI^2 in terms of efficiency, precision and more activation functions (like sigmoid and $\tanh$) so that abstract interpretation based approach can be applied to DNNs of larger size and more complex structures.

Among the above methods, DeepPoly [37] is a most outstanding one regarding precision and scalability. DeepPoly is an abstract domain specially developed for DNN verification. It sufficiently considers the structures and the operators of a DNN, and it designs a polytope expression which not only fits for these structures and operators to control the loss of precision, but also works with a very small time overhead to achieve scalability. However, as an abstraction interpretation based method, it provides very little insight if it fails to verify the property. In this work, we propose a method to improve DeepPoly by eliminating spurious regions through abstraction refinement. A spurious region is a region computed using abstract semantics, conjuncted with the negation of the property to be verified. This region is spurious in the sense that if the property is satisfied, then this region, although not empty, does not contain any true counterexample which can be realized in the original program. In this case, we propose a refinement strategy to rule out the spurious region, i.e., to prove that this region does not contain any true counterexamples.

Our approach is based on DeepPoly and improves it by refinement of the spurious region through linear programming. The core idea is to intersect the abstraction constructed by abstract interpretation with the negation of the property to generate a spurious region, and perform linear programming on the constraints of the spurious region so that the bounds of the ReLU neurons whose behaviors are uncertain can be tightened. As a result, some of these neurons can be determined to be definitely activated or deactivated, which significantly improves the precision of the abstraction given by abstract interpretation. This procedure can be performed iteratively and the precision of the abstraction are gradually improved, so that we are likely to rule out this spurious region in some iteration. If we successfully rule out all the possible spurious regions through such an iterative refinement, the property is soundly verified. Our method is similar in spirit to counterexample guided abstraction refinement (CEGAR) [6], i.e., we apply abstract interpretation for abstraction and linear programming for refinement. A fundamental difference is that we use the constraints of the spurious region, instead of a concrete counterexample (which is challenging to construct in our setting), as the guidance of refinement.

The same spurious region guided refinement approach is also effective in quantitative robustness verification. Instead of requiring that all inputs in the region should be correctly classified, a certain probability of error in the region is allowed. Quantitative robustness is more realistic and general compared to the ordinary robustness, and a DNN verified against quantitative robustness is useful in practice as well. The spurious region guided refinement approach naturally fits for this setting, since a comparatively precise over-approximation of the spurious region implies a sound robustness confidence. To the best of our knowledge, for DNNs, this is the first work to verify quantitative robustness with strict soundness guarantee, which distinguishes our approach from the previous sampling based methods like [45,46,3].

In summary, our main contributions are as follows:

- We propose spurious region guided refinement to verify robustness properties of deep neural networks. This approach significantly improves the precision of DeepPoly and it can verify more challenging properties than DeepPoly.
- We implement the algorithms as a prototype and run them on networks trained on popular datasets like MNIST and ACAS Xu. The experimental results show that our approach significantly improves the precision of DeepPoly in successfully verifying much stronger robustness properties (larger maximum radius) and determining the behaviors of a great proportion of uncertain ReLU neurons.
- We apply our approach to solve quantitative robustness verification problem with strict soundness guarantee. In the experiments, we observe that, comparing to using only DeepPoly, the bounds by our approach can be up to two orders of magnitudes better in the experiments.

Organisations of the paper. We provide preliminaries in Section 2. DeepPoly is recalled in Section 3. We present our overall verification framework and the algorithm in Section 4, and discuss quantitative robustness verification in Section 5. Section 6 evaluates our algorithms through experiments. Section 7 reviews related works and concludes the paper.

2 Preliminaries

In this section we recall some basic notions on deep neural networks, local robustness verification, and abstract interpretation. Given a vector $x \in \mathbb{R}^m$, we write x_i to denote its i-th entry for $1 \leq i \leq m$.

2.1 Robustness verification of deep neural networks

In this work, we focus on deep feedforward neural networks (DNNs), which can be represented as a function $f : \mathbb{R}^m \to \mathbb{R}^n$, mapping an input $x \in \mathbb{R}^m$ to its output $y = f(x) \in \mathbb{R}^n$. A DNN f often classifies an input x by obtaining the maximum dimension of the output, i.e., $\arg\max_{1 \leq i \leq n} f(x)_i$. We denote such a DNN by $C_f : \mathbb{R}^m \to C$ which is defined by $C_f(x) = \arg\max_{1 \leq i \leq n} f(x)_i$ where $C = \{1, \ldots, n\}$ is the set of classification classes.

A DNN has a sequence of layers, including an input layer at the beginning, followed by several hidden layers, and an output layer in the end. The output of a layer is the input of the next layer. Each layer contains multiple neurons, the number of which is known as the dimension of the layer. The DNN f is the composition of the transformations between layers. Typically an affine transformation followed by a non-linear activation function is performed. For an affine transformation $y = Ax + b$, if the matrix A is not sparse, we call such a layer fully connected. A DNN with only fully connected layers and activation functions is a fully connected neural network (FNN). In this work, we focus on the rectified linear unit (ReLU) activation function, defined as $\mathrm{ReLU}(x) = \max(x, 0)$ for $x \in \mathbb{R}$. Typically, a DNN verification problem is defined as follows:

Definition 1. *Given a DNN $f : \mathbb{R}^m \to \mathbb{R}^n$, a set of inputs $X \subseteq \mathbb{R}^m$, and a property $P \subseteq \mathbb{R}^n$, we need to determine whether $f(X) := \{f(x) \mid x \in X\} \subseteq P$ holds.*

Local robustness describes the stability of the behaviour of a normal input under a perturbation. The range of input under this perturbation is the robustness region. For a DNN $C_f(x)$ which performs classification tasks, a robustness property typically states that C_f outputs the same class on the robustness region.

There are various ways to define a robustness region, and one of the most popular ways is to use the L_p norm. For $x \in \mathbb{R}^m$ and $1 \leq p < \infty$, we define the L_p norm of x to be $\|x\|_p = (\sum_{i=1}^{m} |x_i|^p)^{\frac{1}{p}}$, and its L_∞ norm $\|x\|_\infty = \max_{1 \leq i \leq m} |x_i|$. We write $\bar{B}_p(x, r) := \{x' \in \mathbb{R}^m \mid \|x - x'\|_p \leq r\}$ to represent a (closed) L_p ball for $x \in \mathbb{R}^m$ and $r > 0$, which is a neighbourhood of x as its robustness region. If we set $X = \bar{B}_p(x, r)$ and $P = \{y \in \mathbb{R}^n \mid \arg\max_i y_i = C_f(x)\}$ in Def. 1, it is exactly the robustness verification problem. Hereafter, we set $p = \infty$.

2.2 Abstract interpretation for DNN verification

Abstract interpretation [7] is a static analysis method and it is aimed to find an over-approximation of the semantics of programs and other complex systems so as to verify their correctness. Generally we have a function $f : \mathbb{R}^m \to \mathbb{R}^n$ representing the concrete program, a set $X \subseteq \mathbb{R}^m$ representing the property that the input of the program satisfies, and a set $P \subseteq \mathbb{R}^n$ representing the property to verify. The problem is to determine whether $f(X) \subseteq P$ holds. However, in many cases it is difficult to calculate $f(X)$ and to determine whether $f(X) \subseteq P$ holds. Abstract interpretation uses abstract domains and abstract transformations to over-approximate sets and functions so that an over-approximation of the output can be obtained efficiently.

Now we have a concrete domain $\mathcal{C}$, which includes X as one of its elements. To make computation efficient, we need an abstract domain $\mathcal{A}$ to abstract elements in the concrete domain. We assume that there is a partial order $\leq$ on $\mathcal{C}$ and $\mathcal{A}$, which in our settings is the subset relation $\subseteq$. We also have a concretization function $\gamma : \mathcal{A} \to \mathcal{C}$ which assigns an abstract element to its concrete semantics, and $\gamma(a)$ is the least upper bounds of the concrete elements that can be soundly abstracted by $a \in \mathcal{A}$. Naturally $a \in \mathcal{A}$ is a sound abstraction of $c \in \mathcal{C}$ if and only if $c \leq \gamma(a)$.

The design of an abstract domain is one of the most important problems in abstract interpretation because it determines the efficiency and precision. In practice, we use a certain type of constraints to represent the abstract elements in an abstract domain. Classical abstract domains for Euclidean spaces include Box, Zonotope [14,15], and Polyhedra [38].

Not only do we need abstract domains to over-approximate sets, but we are also required to adopt over-approximation to functions. Here we consider the lifting of the function $f : \mathbb{R}^m \to \mathbb{R}^n$ defined as $T_f(X) : \mathcal{P}(\mathbb{R}^m) \to \mathcal{P}(\mathbb{R}^n)$, $T_f(X) := f(X) =$

$\{f(x) \mid x \in X\}$. Now we have an abstract domain $\mathcal{A}_k$ for the k-dimension Euclidean space and the corresponding concretization γ. A function $T_f^{\#} : \mathcal{A}_m \to \mathcal{A}_n$ is a sound abstract transformer of T_f, if $T_f \circ \gamma \subseteq \gamma \circ T_f^{\#}$.

When we have a sound abstraction $X^{\#} \in \mathcal{A}$ of X and a sound abstract transformer $T_f^{\#}$, we can use the concretization of $T_f^{\#}(X^{\#})$ to over-approximate $f(X)$ since we have $f(X) = T_f(X) \subseteq T_f(\gamma(X^{\#})) \subseteq \gamma \circ T_f^{\#}(X^{\#})$. If $\gamma \circ T_f^{\#}(X^{\#}) \subseteq P$, the property P is successfully verified. Obviously, verification through abstract interpretation is sound but not complete. Hereafter, we write $f^{\#}$ to represent $T_f^{\#}$ for simplicity.

AI^2 [13] first adopted abstract interpretation to verify DNNs, and many subsequent works like [36,37,23] focused on improving its efficiency and precision through, e.g., defining new abstract domains. As a deep neural network, the function $f : \mathbb{R}^m \to \mathbb{R}^n$ can be regarded as a composition $f = f_l \circ \cdots \circ f_1$ of its $l+1$ layers, where f_j performs the transformation between the j-th and the $(j+1)$-th layer, i.e., it can be an affine transformation, or a ReLU operation. If we choose Box, Zonotope, or Polyhedra as the abstract domain, then for linear transformations and the ReLU functions, their abstract transformers have been developed in [13]. After we have abstract transformers $f_j^{\#}$ for these f_j, we can conduct abstract interpretation layer by layer as $f_l^{\#} \circ \cdots \circ f_1^{\#}(X^{\#})$.

3 A Brief Introduction to DeepPoly

Our approach relies on the abstract domain DeepPoly [37], which is the state-of-the-art abstract domain for DNN verification. It defines the abstract transformers of multiple activation functions and layers used in DNNs. The core idea of DeepPoly is to give every variable an upper and a lower bound in the form of an affine expression using only variables that appear before it. It can express a polyhedron globally. Moreover, experimentally, it often has better precision than Box and Zonotope domains.

We denote the n-dimensional DeepPoly abstract domain with $\mathcal{A}_n$. Formally an abstract element $a \in \mathcal{A}_n$ is a tuple $(a^{\leq}, a^{\geq}, l, u)$, where $a^{\leq}$ and $a^{\geq}$ give the i-th variable x_i a lower bound and an upper bound, respectively, in the form of a linear combination of variables which appear before it, i.e. $\sum_{k=1}^{i-1} w_k x_k + w_0$, for $i = 1, \ldots, n$, and $l, u \in \mathbb{R}^n$ give the lower bound and upper bound of each variable, respectively. The concretization of a is defined as

$$\gamma(a) = \{x \in \mathbb{R}^n \mid a_i^{\leq} \leq x_i \leq a_i^{\geq},\ \ i = 1, \ldots, n\}. \tag{1}$$

The abstract domain $\mathcal{A}_n$ also requests that its abstract elements a should satisfy the invariant $\gamma(a) \subseteq [l, u]$. This invariant helps construct efficient abstract transformers.

For an affine transformation $x_i = \sum_{k=1}^{i-1} w_k x_k + w_0$, we set

$$a_i^{\leq} = a_i^{\geq} = \sum_{k=1}^{i-1} w_k x_k + w_0.$$

By substituting the variables x_j appearing in $a_i^{\leq}$ with $a_j^{\leq}$ or $a_j^{\geq}$ according to its coefficient at most $i-1$ times, we can obtain a sound lower bound in the form of linear

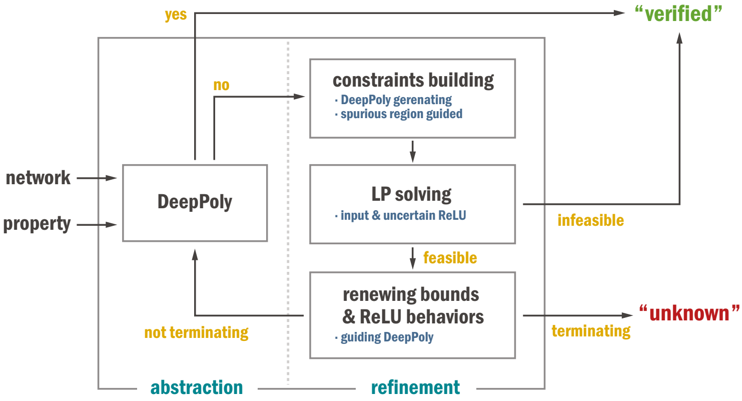

Fig. 1. Framework of spurious region guided refinement

combination on input variables only, and l_i can be computed immediately from the range of input variables. A similar procedure also works for computing u_i.

For a ReLU transformation $x_i = \mathrm{ReLU}(x_j)$, we consider two cases:

- If $l_j \geq 0$ or $u_j \leq 0$, this ReLU neuron is definitely *activated* or *deactivated*, respectively. In this case, this ReLU transformation actually performs an affine transformation, and thus its abstract transformer can be defined as above.
- If $l_j < 0$ and $u_j > 0$, the behavior of this ReLU neuron is *uncertain*, and we need to over-approximate this relation with a linear upper/lower bound. The best upper bound is $a_i^{\geq} = \frac{u_j(x_j - l_j)}{u_j - l_j}$. For the lower bound, there are multiple choices $a_i^{\leq} = \lambda x_j$ where $\lambda \in [0, 1]$. We choose $\lambda \in \{0, 1\}$ which minimizes the area of the constraints. Basically we have two abstraction modes here, corresponding to the two choices of λ.

Note that for a DNN with only ReLU as non-linear operators, over-approximation occurs only when there are uncertain ReLU neurons, which are over-approximated using a triangle. The key of improving the precision is thus to compute the bounds of the uncertain ReLU neurons as precisely as possible, and to determine the behaviors of the most uncertain ReLU neurons.

DeepPoly also supports activation functions which are monotonically increasing, convex on $(-\infty, 0]$ and concave on $[0, +\infty)$, like sigmoid and $\tanh$, and it supports max pooling layers. Readers can refer to [37] for details.

4 Spurious Region Guided Refinement

We explain the main steps of our algorithm, as depicted in Fig. 1. For the input property and network, we first employ DeepPoly as the initial step to compute $f^{\#}(X^{\#})$. The concretization of $f^{\#}(X^{\#})$ is the conjunction of many linear inequities given in Eq. 1, and for the robustness property P, the negation $\neg P$ is the disjunction of several linear inequities $\neg P = \bigvee_{t \neq C_f(x)} (y_{C_f(x)} - y_t \leq 0)$.

1. We check whether $f^{\#}(X^{\#}) \cap^{\#} (y_{C_f(x)} - y_t \leq 0) = \bot$ holds for each t, which follows the same method as DeepPoly, i.e., we compute the lower bound of $y_{C_f(x)} - y_t$ and see whether it is larger than 0. In case of yes, it indicates that the label t cannot be classified, as it is dominated by $C_f(x)$. Otherwise, we have $f^{\#}(X^{\#}) \cap^{\#} \neg P \neq \bot$, we have the conjunction $\gamma(f^{\#}(X^{\#})) \wedge \neg P$ as a potential *spurious region*, which represents the intersection of the abstraction of the real semantics and the negation of the property to verify. We call such a region spurious because if the property is satisfied, then this region does not contain a true counterexample, i.e., a pair of input and output (x^*, y^*) such that $y^* = f(x^*)$ and y^* violates the property P. In this case, this region is spuriously constructed due to the abstraction of the real semantics, where the counterexamples cannot be realized, and thus we aim to rule out the spurious region.
2. If no potential spurious region is found, our algorithm safely returns yes.
3. Assume now that we have a the potential spurious region. The core idea is to use the constraints of the spurious region to refine this spurious region. Here a natural way to refine the spurious region is linear programming, since all the constraints here are linear inequities. If the linear programming is infeasible, it indicates that the region is spurious, and thus we can return an affirmative result. Otherwise, our refinement will tighten the bounds of variables involved in the DNN, especially the input variables and uncertain ReLU neurons, and these tightened bounds help further give a more precise abstraction.
4. As our approach is based on DeepPoly, similarly, we cannot guarantee completeness. We set a threshold N of the number of iterations as a simple termination condition. If the termination condition is not reached, we run DeepPoly again, and return to the first step.

Below we give an example, illustrating how refinement can help in robustness verification.

Example 1. Consider the network $f(x) = \mathrm{ReLU}\left(\begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix} x + \begin{pmatrix} 0 \\ 2.5 \end{pmatrix}\right)$ and the region $\bar{B}_\infty((0,0)^{\mathrm{T}}, 1)$. The robustness property P here is $y_2 - y_1 > 0$. We invoke first DeepPoly: the lower bound of $y_2 - y_1$ given by DeepPoly is -0.5. As a result, the robustness property cannot be verified directly. Fig. 2(a) shows details of the example.

We fail to verify the property in Example 1 because for the uncertain ReLU relation $y_1 = \mathrm{ReLU}(x_3)$, the abstraction is imprecise, and the key to making the abstraction more precise here is to obtain as tight a bound as possible for x_3.

Example 2. We use the constraints in Fig. 2(a) and additionally the constraint $y_2 - y_1 \leq 0$ (i.e., $\neg P$) as the input of linear programming. Our aim is to obtain a tighter bound of the input neurons x_1 and x_2, as well as the uncertain ReLU neuron x_3, so the objective functions of the linear programming are $\min x_i$ and $\min -x_i$ for $i = 1, 2, 3$. All the three neurons have a tighter bound after the linear programming (see the red part in Fig. 2(b)). Fig. 2(b) shows the running of DeepPoly under these new bounds, where the input range and the abstraction of the uncertain ReLU neuron are both refined. Now the lower bound of $y_2 - y_1$ is 0.25, so DeepPoly successfully verifies the property.

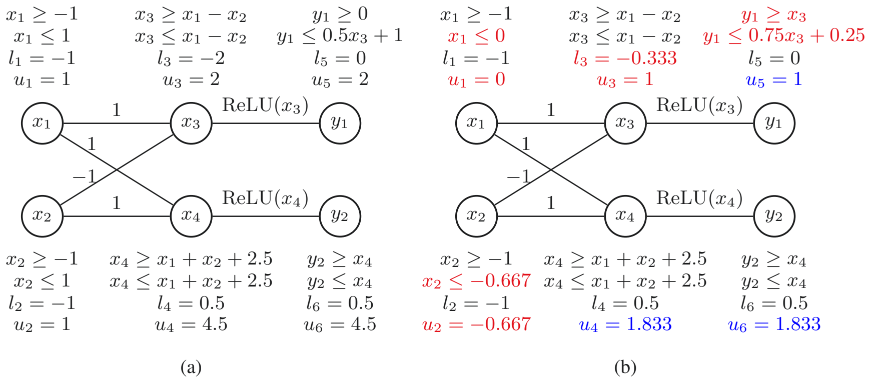

Fig. 2. Example 1 (left) and Example 2 (right): where the red parts are introduced through linear programming based refinement and the blue parts are introduced by a second run of DeepPoly.

4.1 Main algorithm

Alg. 1 presents our algorithm. First we run abstract interpretation to find the uncertain neurons and the spurious regions (Line 2–5). For each possible spurious region, we have a **while** loop which iteratively refines the abstraction. In each iteration we perform linear programming to renew the bounds of the input neurons and uncertain ReLU neurons; when we find that the bound of an uncertain ReLU neuron becomes definitely non-negative or non-positive, then the ReLU behavior of this neuron is renewed (Line 14–20). We use them to guide abstract interpretation in the next step (Line 21–22). Here in Line 22, we make sure that during the abstract interpretation, the abstraction of previous uncertain neurons (namely the uncertain neurons before the linear programming step in the same iteration) compulsorily follows the new bounds and new ReLU behaviors given by the current $C_{\geq 0}$, $C_{\leq 0}$, l, and u, where these bounds will not be renewed by abstract interpretation, and the concretization of Y is defined as

$$\gamma(Y) = \{x \mid \forall i.\ Y_i^{\leq} \leq x_i \leq Y_i^{\geq}\} \cap [l, u]. \tag{2}$$

The **while** loop ends when (i) either we find that the spurious region is infeasible (Line 11, 24) and we proceed to refine the next spurious region, with a label Verified True, (ii) or we reach the terminating condition and fail to rule out this spurious region, in which case we return UNKNOWN. If every **while** loop ends with the label Verified True, we successfully rule out all the spurious regions and return YES. An observation is that, if some spurious regions have been ruled out, we can add the constraints of their negation to make the current spurious region smaller so as to improve the precision (Line 9).

Here we discuss the soundness of Alg. 1. We focus on the **while** loop and claim that it has the following loop invariant:

Invariant 1 *The abstract element Y over-approximates the intersection of the semantics of f on $\bar{B}_\infty(x, r)$ and the spurious region, i.e., $f(\bar{B}_\infty(x, r)) \cap \mathrm{Spu} \subseteq \gamma(Y)$.*

Algorithm 1 Spurious region guided robustness verification

```
Input:
    DNN $f$, input $x$, radius $r$.
Output:
    Return “YES” if verified, or “UNKNOWN” otherwise.
 1: function VERIFY($f, x, r$)
 2:     $Y_0 \leftarrow f^{\#}(\bar{B}_\infty(x, r))$    ▷ abstract interpretation with DeepPoly
 3:     $V_\text{u} \leftarrow \{v \mid v$ was marked as uncertain in Line 2$\}$
 4:     $A = \{t \mid Y_0 \cap^{\#} (y_{C_f(x)} - y_t \leq 0) \neq \bot\}$
 5:     if $A = \emptyset$ then return YES    ▷ otherwise $A = \{t_1, \ldots, t_l\}$
 6:     for $i \leftarrow 1$ to $l$ do
 7:         Verified $\leftarrow$ False, $V \leftarrow V_\text{u}$, $Y \leftarrow Y_0$    ▷ denote $Y = (Y^{\leq}, Y^{\geq}, l, u)$
 8:         $C_{\geq 0} \leftarrow \emptyset$, $C_{\leq 0} \leftarrow \emptyset$    ▷ set of new activated/deactivated neurons
 9:         Spu $\leftarrow (y_{C_f(x)} - y_{t_i} \leq 0) \wedge \bigwedge_{j=1}^{i-1}(y_{C_f(x)} - y_{t_j} \geq 0)$    ▷ spurious region
10:         while terminating condition not satisfied do
11:             if $Y \wedge$ Spu is infeasible then
12:                 Verified $\leftarrow$ True
13:                 break
14:             for $v \in V \cup V_0$ do    ▷ $V_0$: set of input neurons
15:                 $(l_v, u_v) \leftarrow$ LP($Y \wedge$ Spu$, v$)
16:             for $v \in V$ do
17:                 if $l_v \geq 0$ then
18:                     $C_{\geq 0} \leftarrow C_{\geq 0} \cup \{v\}$, $V \leftarrow V \setminus \{v\}$
19:                 else if $u_v \leq 0$ then
20:                     $C_{\leq 0} \leftarrow C_{\leq 0} \cup \{v\}$, $V \leftarrow V \setminus \{v\}$
21:             $X \leftarrow \bigcap_{v \in V_0} \{l_v \leq v \leq u_v\}$
22:             $Y \leftarrow f^{\#}(X)$ according to $C_{\geq 0}$, $C_{\leq 0}$, $l$, and $u$
23:             $V \leftarrow \{v \mid v$ was marked as uncertain in Line 22$\} \setminus (C_{\geq 0} \cup C_{\leq 0})$
24:             if $Y \cap^{\#} (y_{C_f(x)} - y_{t_i} \leq 0) = \bot$ then
25:                 Verified $\leftarrow$ True
26:                 break
27:         if Verified = False then return UNKNOWN
28:     return YES
```

The initialization of Y is $f^{\#}(\bar{B}_\infty(x, r))$ and it is naturally an over-approximation. The box X is obtained by linear programming on $Y \wedge$ Spu, and $f^{\#}(X)$ is calculated through abstract interpretation and the bounds given by linear programming on $Y \wedge$ Spu, and thus it remains an over-approximation. It is worth mentioning that, when we run DeepPoly in Line 22, we are using the bounds obtained by linear programming to guide DeepPoly, and this may violate the invariant $\gamma(a) \subseteq [l, u]$ mentioned in Sect. 3. Nonotheless, soundness still holds since the concretization of Y is newly defined in Eq. 2, where both items in the intersection over-approximate $f(\bar{B}_\infty(x, r)) \cap$ Spu. With Invarient 1, Alg. 1 returns YES if for any possible spurious region Spu, the over-approximation of $f(\bar{B}_\infty(x, r)) \cap$ Spu is infeasible, which implies the soundness of Alg. 1.

4.2 Iterative refinement of the spurious region

Here we present more theoretical insight on the iterative refinement of the spurious region. An iteration of the **while** loop in Alg. 1 can be represented as a function $\mathcal{L} : \mathcal{A} \to \mathcal{A}$, where $\mathcal{A}$ is the DeepPoly domain. An interesting observation is that, the abstract transformer $f^{\#}$ in the DeepPoly domain is not necessarily increasing, because different input ranges, even if they have inclusion relation, may lead to different choices of the abstraction mode of some uncertain ReLU neurons, which may violate the inclusion relation of abstraction. We have found such examples during our experiment, which is illustrated in the following example.

Example 3. Let $f(x) = \mathrm{ReLU}(x)$ with input ranges $I_1 = [-2, 1]$ and $I_2 = [-2, 3]$. We have $f^{\#}(I_1) = \{(x_1, x_2)^{\mathrm{T}} \in \mathbb{R}^2 \mid -2 \le x_1 \le 1,\ \ x_2 \ge 0,\ \ x_2 \le \frac{1}{3}x_1 + \frac{2}{3}\}$ and $f^{\#}(I_2) = \{(x_1, x_2)^{\mathrm{T}} \in \mathbb{R}^2 \mid -2 \le x_1 \le 3,\ \ x_2 \ge x_1,\ \ x_2 \le \frac{3}{5}x_1 + \frac{6}{5}\}$. We observe $(1, 0)^{\mathrm{T}} \in f^{\#}(I_1)$ but $(1, 0)^{\mathrm{T}} \notin f^{\#}(I_2)$, which implies that the transformer $f^{\#}$ is not increasing.

This fact also implies that $\mathcal{L}$ is not necessarily increasing, which violates the condition of Kleene's Theorem on fixed point [4].

Now we turn to the analysis of the sequence $\{Y_k = \mathcal{L}^k(f^{\#}(\bar{B}_\infty(x, r)))\}_{k=1}^{\infty}$, where $\mathcal{L}^1 := \mathcal{L}$ and $\mathcal{L}^k := \mathcal{L} \circ \mathcal{L}^{k-1}$ for $k \ge 2$. First we have the following lemma showing that in our settings every decreasing chain S in the DeepPoly domain $\mathcal{A}$ has a meet $\bigcap^{\#} S \in \mathcal{A}$.

Lemma 1. *Let $\mathcal{A}_n$ be the n-dimensional DeepPoly domain and $\{a^{(k)}\} \subseteq \mathcal{A}_n$ a decreasing bounded sequence of non-empty abstract elements. If the coefficients in $a_i^{(k),\le}$ and $a_i^{(k),\ge}$ are uniformly bounded, then there exists an abstract element $a^* \in \mathcal{A}_n$ s.t. $\gamma(a^*) = \bigcap_{k=1}^{\infty} \gamma(a^{(k)})$.*

Remark: The condition that the coefficients in $a_i^{(k),\le}$ and $a_i^{(k),\ge}$ are uniformly bounded are naturally satisfied in our setting, since in a DNN the coefficients and bounds involved have only finitely many values. Readers can refer to [50] for a formal proof.

Lemma 1 implies that if our sequence $\{Y_k\}$ is decreasing, then the iterative refinement converges to an abstract element in DeepPoly, which is the greatest fixed point of $\mathcal{L}$ that is smaller than $f^{\#}(\bar{B}_\infty(x, r))$. A sufficient condition for $\{Y_k\}$ being decreasing is that during the abstract interpretation in every Y_k, every initial uncertain neuron maintains its abstraction mode, i.e. its corresponding λ does not change, before its ReLU behavior is determined. A weaker sufficient condition for convergence is that change in abstraction mode of uncertain neurons never happens after finitely many iterations.

If the abstraction mode of uncertain neurons changes infinitely often, generally the sequence $\{Y_k\}$ does not converge. In this case, we can consider its subsequence in which every Y_k is obtained with the same abstraction mode. It is easy to see that such a subsequence must be decreasing and thus have a meet, as it is an accumulative point of the sequence $\{Y_k\}$. Since there are only finitely many choices of abstraction modes, such a accumulative points exists in $\{Y_k\}$, and there are only finitely many accumulative points. We conclude these results in the following theorem which describes the convergence behavior of our iterative refinement of the spurious region:

Theorem 2. *There exists a subsequence $\{Y_{n_k}\}$ of $\{Y_k\}$ s.t. $\{Y_{n_k}\}$ is decreasing and thus has a meet $\bigcap^{\#}\{Y_{n_k}\}$. Moreover, the set*

$$\left\{\bigcap{}^{\#}\{Y_{n_k}\} \mid \{Y_{n_k}\} \textit{ is a decreasing subsequence of } \{Y_k\}\right\}$$

is finite, and it is a singleton if exact one abstraction mode of uncertain ReLU *neurons happens infinitely often.*

Proof. Since the abstraction modes of uncertain ReLU neurons have only finitely many choices, there must be one which happens infinitely often in the computation of the sequence $\{Y_k\}$, and we choose the subsequence $\{Y_{n_k}\}$ in which every item is computed through this abstraction mode. Obviously $\{Y_{n_k}\}$ is decreasing and thus has a meet.

For a decreasing subsequence $\{Y_{n_k}\}$, we can find its subsequnce in which the abstraction mode of uncertain ReLU neurons does not change, and they have the same meet. Since there are only finitely many choices of abstraction modes of uncertain ReLU neurons, such accumulative points of $\{Y_k\}$ also have finitely many values. If exact one abstraction mode of uncertain ReLU neurons happens infinitely often, obviously there is only one accumulative point in $\{Y_k\}$. □

4.3 Optimizations

In the implementation of our main algorithm, we propose the following optimizations to improve the precision of refinement.

Optimization 1: More precise constraints in linear programming. In Line 15 of Alg. 1, it is not the best choice to take the linear constraints in the abstract element Y into linear programming, because the abstraction of uncertain ReLU neurons in DeepPoly is not the best. Planet [10] has a component which gives a more precise linear approximation for uncertain ReLU relations, where it uses the linear constraints $y \leq \frac{u(x-l)}{u-l},\ y \geq x,\ y \geq 0$ to over-approximate the relation $y = \mathrm{ReLU}(x)$ with $x \in [l, u]$.

Optimization 2: Priority to work on small spurious regions. In Line 6 of Alg. 1,we determine the order of refining the spurious regions based on their sizes, i.e., a smaller region is chosen earlier. This is based on the intuition that Alg. 1 works effectively if the spurious region is small. After the small spurious regions are ruled out, the constraints of large spurious regions can be tightened with the conjunction $\bigwedge_{j=1}^{i-1}(y_{C_f(x)} - y_{t_j} \geq 0)$. It is difficult to strictly determine which spurious region is the smallest, and thus we refer to the lower bound of $y_{C_f(x)} - y_{t_i}$ given by DeepPoly, i.e., the larger this lower bound is, the smaller the spurious region is likely to be, and we perform the **for** loop in Line 6 of Alg. 1 in this order.

5 Quantitative Robustness Verification

In this section we recall the notion of quantitative robustness and show how to verify a quantitative robustness property of a DNN with spurious region guided refinement.

In practice, we may not need a strict condition of robustness to ensure that an input x is not an adversarial example. A notion of mutation testing is proposed in [44,43], which requires that an input x is normal if it has a low *label change rate* on its neighbourhood. They follow a statistical way to estimate the label change rate of an input, which motivates us to give a formal definition of the property showing a low label change rate, and to consider the verification problem for such a property. Below we recall the definition of *quantitative robustness* [27], where we have a parameter $0 < \eta \leq 1$ representing the confidence of robustness.

Definition 2. *Given a DNN $C_f : \mathbb{R}^m \to C$, an input $x \in \mathbb{R}^m$, $r > 0$, $0 < \eta \leq 1$, and a probability measure μ on $\bar{B}_\infty(x, r)$, f is η-robust at x, if*

$$\mu(\{x' \in \bar{B}_\infty(x, r) \mid C_f(x') = C_f(x)\}) \geq \eta.$$

Def. 2 has a tight association with label change rate, i.e., if x is η-robust, then the label change rate should be smaller than, or close to $1 - \eta$. Hereafter, we set μ to be the uniform distribution on $\bar{B}_\infty(x, r)$.

It is natural to adapt spurious region guided refinement to quantitative robustness verification. In Alg. 1, we do not return UNKNOWN when we cannot rule out a spurious region, but record the volume of the box X as an over-approximation of the Lebesgue measure of the spurious region. After we work on all the spurious regions, we calculate the sum of these volume, and obtain a sound robustness confidence. Here we do not calculate the volume of the spurious region because precise calculation of volume of a high-dimensional polytope remains open, and we do not choose to use randomized algorithms because it may not be sound.

We further improve the algorithm through the powerset technique [13]. Powerset technique is a classical and effective way to enhance the precision of abstract interpretation. We split the input region into several subsets, and run abstract interpretation on these subsets, In our quantitative robustness verification setting, powerset technique not only improves the precision, but also accelerates the algorithm in some situations: If the subsets have the same volume, and the percentage of the subsets on which we may fail to verify robustness is already smaller than $1 - \eta$, then we have successfully verified the η-robustness property.

6 Experimental Evaluation

We implement our approach as a prototype called DeepSRGR. The implementation is based on a re-implementation of the ReLU and the affine abstract transformers of DeepPoly in Python 3.7 and we amend it accordingly to implement Alg. 1. We use CVXPY [8] as our modeling language for convex optimization problems and CBC [18] as the LP solver. It is worth mentioning that we ignore the floating point error in our re-implementation of DeepPoly because sound linear programming currently does not scale in our experiments. In the terminating condition, we set $N = 5$. The two optimizations in Sect. 4.3 are adopted in all the experiments. All the experiments are conducted on a CentOS 7.7 server with 16 Intel Xeon Platinum 8153 @2.00GHz (16 cores) and 512G RAM, and they use 96 sub-processes concurrently at most. Readers can find all the source code and other experimental materials in https://iscasmc.ios.ac.cn/ToolDownload/?Tool=DeepSRGR.

Datasets. We use MNIST [22] and ACAS Xu [12,17] as the datasets in our experiments. MNIST contains 60 000 grayscale handwritten digits of the size 28×28. We can train DNNs to classify the images by the written digits on them. The ACAS Xu system is aimed to avoid airborne collisions for unmanned aircrafts and it uses an observation table to make decisions for the aircraft. In [19], the observation table is realized by training DNNs instead of storing it.

Networks. On MNIST, we trained seven fully connected networks of the size 6×20, 3×50, 3×100, 6×100, 6×200, 9×200, and 6×500, where $m \times n$ refers m hidden layers and n neurons in each hidden layer, and we name them from FNN2 to FNN8, respectively (we also have a small network FNN1 for testing). On ACAS Xu, we randomly choose three networks used in [20], all of the size 6×50.

6.1 Improvement in precision

First we compare DeepPoly and DeepSRGR in terms of their precision of robustness verification. We consider the following two indices: (i) the maximum radius that the two tools can verify, and (ii) the number of uncertain ReLU neurons whose behaviors can be further determined by DeepSRGR. For each network, we randomly choose three images from the MNIST dataset, and calculate their maximum radius that the two tools can verify through a binary search on the seven FNNs. In column *"# uncertin ReLU"* we record the number of the uncertain ReLU neurons when first applying DeepPoly, and also count how many of them are *renewed*, namely become definitely activated/deactivated in later iterations when applying DeepSRGR.

Table 1 shows the results. We can see from Table 1 that DeepSRGR can verify much stronger (i.e., larger maximum radius) robustness properties than DeepPoly. The average number of iterations for ruling out a spurious region is 2.875, and about half of the spurious regions can be ruled out within 2 iterations. DeepSRGR sometimes determines behaviors of a large proportion of uncertain ReLU neurons on large networks: Considering the last picture of the most challenging network FNN8, more than ninety percent ($92.6\% \approx \frac{1269}{1371}$) of the uncertain neurons are renewed. Improvement in precision evaluated in this experiment works for verification of both robustness and quantitative robustness, and this is why our method is effective in both tasks.

6.2 Robustness verification performance

In this setting, we randomly choose 50 samples from the MNIST dataset. We fix four radii, 0.037, 0.026, 0.021, and 0.015 for the four networks FNN4 – FNN7 respectively, and verify the robustness property with the corresponding radius on the 50 inputs. The radius chosen here is very challenging for the corresponding network.

Table 2 presents the results. As we can see, DeepSRGR can verify significantly more properties than DeepPoly. Linear programming in DeepSRGR takes a large amount of time in the experiment, and thus DeepSRGR is less efficient (a DeepPoly run takes no

	Maximum radius		# spurious	# uncertain ReLU		% renewed		# iterations	
	DeepPoly	DeepSRGR	regions	Original	Renewed	MAX	AVG	MAX	GT
FNN2	0.034	0.047	6	51	38	74.5%	48.4%	5	17
	0.017	0.023	3	47	37	78.7%	51.8%	4	9
	0.017	0.023	1	34	25	73.5%	73.5%	4	4
FNN3	0.049	0.066	6	88	69	78.4%	60.9%	5	15
	0.025	0.033	7	94	85	90.4%	46.0%	5	18
	0.045	0.058	3	98	45	45.1%	27.2%	5	9
FNN4	0.045	0.060	6	180	102	56.7%	35.2%	5	19
	0.024	0.030	6	199	144	72.4%	36.5%	4	15
	0.035	0.046	2	155	103	66.5%	42.9%	5	7
FNN5	0.034	0.042	7	305	245	80.3%	37.8%	5	20
	0.016	0.019	5	315	204	64.8%	34.0%	4	14
	0.021	0.027	7	337	256	76.0%	34.9%	5	18
FNN6	0.022	0.026	7	683	271	39.7%	19.8%	4	18
	0.011	0.013	6	657	483	73.5%	36.7%	3	14
	0.021	0.025	8	723	169	23.4%	12.2%	5	21
FNN7	0.021	0.023	9	987	297	30.1%	10.0%	5	29
	0.010	0.011	5	877	648	73.9%	26.8%	3	11
	0.017	0.019	7	913	352	38.6%	24.3%	3	16
FNN8	0.037	0.044	9	1 504	976	64.9%	45.9%	5	36
	0.020	0.022	9	1 213	818	67.4%	33.3%	3	21
	0.033	0.040	9	1 371	1 269	92.6%	51.1%	5	37

Table 1. Maximum radius which can be verified by DeepPoly and DeepSRGR, and details of DeepSRGR running on its maximum radius, where in the number of renewed uncertain nuerons, we show the largest one among the spurious regions. MAX, AVG, and GT means the maximum, the average, and the grant total among the spurious regions, respectively. The indices of the three images are 414, 481, and 65 in the MNIST dataset.

more than 100 seconds on FNN7). Furthermore, we again run the 15 running examples which are not verified by DeepSRGR on FNN4, by resetting the maximum number of iterations to 20 and 50. We have the following observations:

- Two more properties (out of 15) are successfully verified when we change N to 20. No more properties can be verified even if we change N from 20 to 50.
- In this experiment, 13 more spurious regions are ruled out, six of which takes 6 iterations, one takes 7, two takes 8, and the other four takes 13, 22, 27, and 32 iterations, respectively. In these running examples, the average number of renewed ReLU behaviors is 102.8, and a large proportion are renewed in the last iteration (47.4% on average). Fig. 3 shows the detailed results.
- As for the 13 spurious regions which cannot be ruled out within 50 iterations, the average number of renewed ReLU behaviors is only 8.54, which is significantly lower than the average of the 13 spurious regions which are newly ruled out. In these running examples, changes in ReLU behaviors and ReLU abstraction modes do not happen after the 9th iteration, and the average number is 4.4.

Model	Size	Radius	# verified		Time (s)	
			DeepPoly	DeepSRGR	MAX	AVG
FNN4	3 × 100	0.037	14	35	3 384	781
FNN5	6 × 100	0.026	19	31	7 508	1 689
FNN6	6 × 200	0.021	14	25	23 157	6 178
FNN7	9 × 200	0.015	25	36	61 760	8 960

Table 2. The number that DeepPoly and DeepSRGR verifies among the 50 inputs, and the maximum/average running time of DeepSRGR.

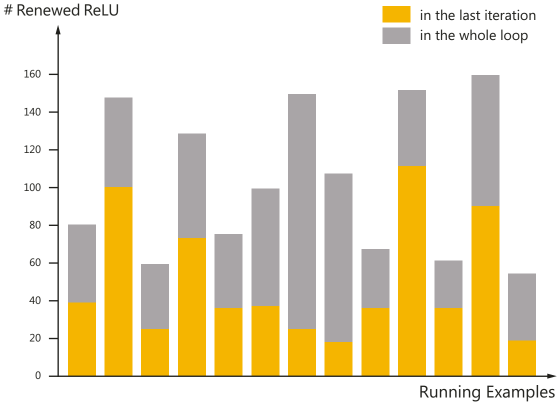

Fig. 3. Number of renewed ReLU behaviors in the spurious regions newly ruled out.

We observe that, by increasing the termination threshold N from 5 to 50, only two more properties out of 15 can be verified additionally. This suggests that our method can effectively identify these spurious regions which are relevant to verification of the property, in a small number of iterations.

6.3 Quantitative robustness verification on ACAS Xu networks

We evaluate DeepSRGR for quantitative robustness verification on ACAS Xu networks. We randomly choose five inputs, and compute the maximum robustness radius for each input on the three networks with DeepPoly through a binary search. In our experiment, the radius for a running example is the maximum robustness radius plus 0.02, 0.03, 0.04, 0.05, and 0.06. We use the powerset technique and the number of splits is 32. For DeepPoly, the robustness confidence it gives is the proportion of the splits on which DeepPoly verifies the property.

Fig. 4 shows the results. We can see that DeepSRGR gives significantly better over-approximation of $1-\eta$ than DeepPoly. That is, in more than 90% running examples, our over-approximation is no more than one half of that given by DeepPoly, and in more than 75% of the cases, our over-approximation is even smaller than one tenth of that given by DeepPoly.

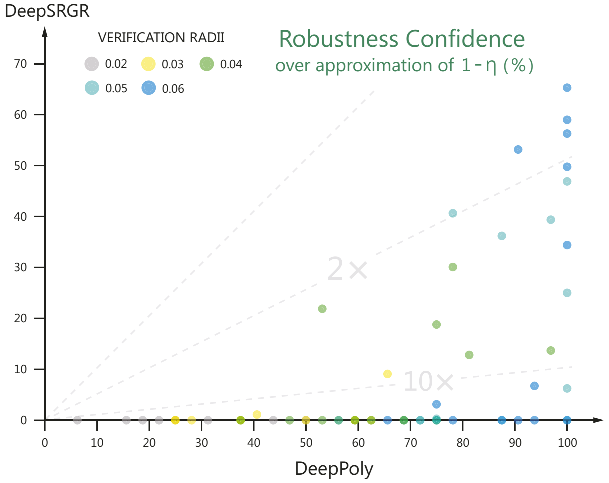

Fig. 4. Quantitative robustness verification using DeepPoly and DeepSRGR

7 Related Works and Conclusion

We have already discussed papers mostly related to our paper. Here we add some more new results. Marabou [21] has been developed as the next generation of Reluplex. Recently, verification approach based on abstraction of DNN models has been proposed in [11,2]. In addition, alternative approaches based on constraint-solving [26,29,5,25], layer-by-layer exhaustive search [16], global optimization [31,9,32], functional approximation [47], reduction to two-player games [48,49], and star set abstraction [41,40] have been proposed as well.

In this work, we propose a spurious region guided refinement approach for robustness and quantitative robustness verification of deep neural networks, where abstract interpretation calculates an abstraction, and linear programming performs refinement with the guidance of the spurious region. Our experimental results show that our tool can significantly improve the precision of DeepPoly, verify more robustness properties, and often provide a quantitative robustness with strict soundness guarantee.

Abstraction interpretation based framework is quite extensive to different DNN models, different properties, and incorporate different verification methods. As future work, we will investigate how to increase the precision further by using more precise linear over-approximation like [35].

Acknowledgement

This work has been partially supported by Key-Area Research and Development Program of Guangdong Province (Grant No. 2018B010107004), National Natural Science Foundation of China (Grant No. 61761136011, 61836005), Natural Science Foundation of Guangdong Province, China (Grant No. 2019A1515011689), and the Fundamental Research Funds for the Zhejiang University NGICS Platform.

References

1. Anderson, G., Pailoor, S., Dillig, I., Chaudhuri, S.: Optimization and abstraction: a synergistic approach for analyzing neural network robustness. In: McKinley, K.S., Fisher, K. (eds.) Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2019, Phoenix, AZ, USA, June 22-26, 2019. pp. 731–744. ACM (2019)
2. Ashok, P., Hashemi, V., Kretínský, J., Mohr, S.: Deepabstract: Neural network abstraction for accelerating verification. In: Hung, D.V., Sokolsky, O. (eds.) Automated Technology for Verification and Analysis - 18th International Symposium, ATVA 2020, Hanoi, Vietnam, October 19-23, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12302, pp. 92–107. Springer (2020)
3. Baluta, T., Chua, Z.L., Meel, K.S., Saxena, P.: Scalable quantitative verification for deep neural networks. CoRR **abs/2002.06864** (2020), https://arxiv.org/abs/2002.06864
4. Baranga, A.: The contraction principle as a particular case of kleene's fixed point theorem. Discret. Math. **98**(1), 75–79 (1991)
5. Bunel, R., Lu, J., Turkaslan, I., Torr, P.H.S., Kohli, P., Kumar, M.P.: Branch and bound for piecewise linear neural network verification. J. Mach. Learn. Res. **21**, 42:1–42:39 (2020)
6. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement. In: Emerson, E.A., Sistla, A.P. (eds.) Computer Aided Verification, 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000, Proceedings. Lecture Notes in Computer Science, vol. 1855, pp. 154–169. Springer (2000)
7. Cousot, P., Cousot, R.: Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Fourth ACM Symposium on Principles of Programming Languages (POPL). pp. 238–252 (1977)
8. Diamond, S., Boyd, S.: CVXPY: A Python-embedded modeling language for convex optimization. Journal of Machine Learning Research **17**(83), 1–5 (2016)
9. Dutta, S., Jha, S., Sankaranarayanan, S., Tiwari, A.: Output range analysis for deep feedforward neural networks. In: Dutle, A., Muñoz, C.A., Narkawicz, A. (eds.) NASA Formal Methods - 10th International Symposium, NFM 2018, Newport News, VA, USA, April 17-19, 2018, Proceedings. Lecture Notes in Computer Science, vol. 10811, pp. 121–138. Springer (2018)
10. Ehlers, R.: Formal verification of piece-wise linear feed-forward neural networks. In: 15th International Symposium on Automated Technology for Verification and Analysis (ATVA2017). pp. 269–286 (2017)
11. Elboher, Y.Y., Gottschlich, J., Katz, G.: An abstraction-based framework for neural network verification. In: Lahiri, S.K., Wang, C. (eds.) Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part I. Lecture Notes in Computer Science, vol. 12224, pp. 43–65. Springer (2020)
12. von Essen, C., Giannakopoulou, D.: Analyzing the next generation airborne collision avoidance system. In: Ábrahám, E., Havelund, K. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 20th International Conference, TACAS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8413, pp. 620–635. Springer (2014)
13. Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., Vechev, M.: AI^2: Safety and robustness certification of neural networks with abstract interpretation. In: 2018 IEEE Symposium on Security and Privacy (S&P 2018). pp. 948–963 (2018)
14. Ghorbal, K., Goubault, E., Putot, S.: The zonotope abstract domain taylor1+. In: International Conference on Computer Aided Verification. pp. 627–633. Springer (2009)

15. Ghorbal, K., Goubault, E., Putot, S.: A logical product approach to zonotope intersection. In: Touili, T., Cook, B., Jackson, P.B. (eds.) Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6174, pp. 212–226. Springer (2010)
16. Huang, X., Kwiatkowska, M., Wang, S., Wu, M.: Safety verification of deep neural networks. In: 29th International Conference on Computer Aided Verification (CAV2017). pp. 3–29 (2017)
17. Jeannin, J., Ghorbal, K., Kouskoulas, Y., Gardner, R., Schmidt, A., Zawadzki, E., Platzer, A.: Formal verification of ACAS x, an industrial airborne collision avoidance system. In: Girault, A., Guan, N. (eds.) 2015 International Conference on Embedded Software, EMSOFT 2015, Amsterdam, Netherlands, October 4-9, 2015. pp. 127–136. IEEE (2015)
18. johnjforrest, Vigerske, S., Santos, H.G., Ralphs, T., Hafer, L., Kristjansson, B., jpfasano, EdwinStraver, Lubin, M., rlougee, jpgoncal1, h-i gassmann, Saltzman, M.: coin-or/cbc: Version 2.10.5 (Mar 2020)
19. Julian, K.D., Kochenderfer, M.J., Owen, M.P.: Deep neural network compression for aircraft collision avoidance systems. CoRR **abs/1810.04240** (2018), http://arxiv.org/abs/1810.04240
20. Katz, G., Barrett, C.W., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: An efficient SMT solver for verifying deep neural networks. In: 29th International Conference on Computer Aided Verification (CAV2017). pp. 97–117 (2017)
21. Katz, G., Huang, D.A., Ibeling, D., Julian, K., Lazarus, C., Lim, R., Shah, P., Thakoor, S., Wu, H., Zeljic, A., Dill, D.L., Kochenderfer, M.J., Barrett, C.W.: The marabou framework for verification and analysis of deep neural networks. In: Dillig, I., Tasiran, S. (eds.) Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I. Lecture Notes in Computer Science, vol. 11561, pp. 443–452. Springer (2019)
22. Lécun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11), 2278–2324 (1998)
23. Li, J., Liu, J., Yang, P., Chen, L., Huang, X., Zhang, L.: Analyzing deep neural networks with symbolic propagation: Towards higher precision and faster verification. In: Chang, B.E. (ed.) Static Analysis - 26th International Symposium, SAS 2019, Porto, Portugal, October 8-11, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11822, pp. 296–319. Springer (2019)
24. Li, R., Li, J., Huang, C., Yang, P., Huang, X., Zhang, L., Xue, B., Hermanns, H.: Prodeep: a platform for robustness verification of deep neural networks. In: Devanbu, P., Cohen, M.B., Zimmermann, T. (eds.) ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020. pp. 1630–1634. ACM (2020)
25. 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: IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019. pp. 11418–11427. Computer Vision Foundation / IEEE (2019)
26. Lomuscio, A., Maganti, L.: An approach to reachability analysis for feed-forward ReLU neural networks. In: KR2018 (2018)
27. Mangal, R., Nori, A.V., Orso, A.: Robustness of neural networks: A probabilistic and practical approach. CoRR **abs/1902.05983** (2019), http://arxiv.org/abs/1902.05983
28. Müller, C., Singh, G., Püschel, M., Vechev, M.T.: Neural network robustness verification on gpus. CoRR **abs/2007.10868** (2020), https://arxiv.org/abs/2007.10868
29. Narodytska, N., Kasiviswanathan, S.P., Ryzhyk, L., Sagiv, M., Walsh, T.: Verifying properties of binarized deep neural networks. In: McIlraith, S.A., Weinberger, K.Q. (eds.) Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the

30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018. pp. 6615–6624. AAAI Press (2018)

30. Pulina, L., Tacchella, A.: An abstraction-refinement approach to verification of artificial neural networks. In: Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings. pp. 243–257 (2010)
31. Ruan, W., Huang, X., Kwiatkowska, M.: Reachability analysis of deep neural networks with provable guarantees. In: IJCAI2018. pp. 2651–2659 (2018)
32. Ruan, W., Wu, M., Sun, Y., Huang, X., Kroening, D., Kwiatkowska, M.: Global robustness evaluation of deep neural networks with provable guarantees for the hamming distance. In: Kraus, S. (ed.) Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019. pp. 5944–5952. ijcai.org (2019)
33. Sheikhtaheri, A., Sadoughi, F., Dehaghi, Z.H.: Developing and using expert systems and neural networks in medicine: A review on benefits and challenges. J. Medical Syst. **38**(9), 110 (2014)
34. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T.P., Leach, M., Kavukcuoglu, K., Graepel, T., Hassabis, D.: Mastering the game of go with deep neural networks and tree search. Nature **529**(7587), 484–489 (2016)
35. Singh, G., Ganvir, R., Püschel, M., Vechev, M.T.: Beyond the single neuron convex barrier for neural network certification. In: Wallach, H.M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E.B., Garnett, R. (eds.) Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada. pp. 15072–15083 (2019)
36. Singh, G., Gehr, T., Mirman, M., Püschel, M., Vechev, M.T.: Fast and effective robustness certification. In: Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada. pp. 10825–10836 (2018)
37. Singh, G., Gehr, T., Püschel, M., Vechev, M.T.: An abstract domain for certifying neural networks. PACMPL **3**(POPL), 41:1–41:30 (2019)
38. Singh, G., Püschel, M., Vechev, M.T.: Fast polyhedra abstract domain. In: Castagna, G., Gordon, A.D. (eds.) Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017. pp. 46–59. ACM (2017)
39. 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 (ICLR2014) (2014)
40. Tran, H., Bak, S., Xiang, W., Johnson, T.T.: Verification of deep convolutional neural networks using imagestars. In: Lahiri, S.K., Wang, C. (eds.) Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part I. Lecture Notes in Computer Science, vol. 12224, pp. 18–42. Springer (2020)
41. Tran, H., Lopez, D.M., Musau, P., Yang, X., Nguyen, L.V., Xiang, W., Johnson, T.T.: Star-based reachability analysis of deep neural networks. In: ter Beek, M.H., McIver, A., Oliveira, J.N. (eds.) Formal Methods - The Next 30 Years - Third World Congress, FM 2019, Porto, Portugal, October 7-11, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11800, pp. 670–686. Springer (2019)
42. Urmson, C., Whittaker, W.: Self-driving cars and the urban challenge. IEEE Intell. Syst. **23**(2), 66–68 (2008)

43. Wang, J., Dong, G., Sun, J., Wang, X., Zhang, P.: Adversarial sample detection for deep neural network through model mutation testing. In: 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE). pp. 1245–1256. IEEE (2019)
44. Wang, J., Sun, J., Zhang, P., Wang, X.: Detecting adversarial samples for deep neural networks through mutation testing. CoRR **abs/1805.05010** (2018), http://arxiv.org/abs/1805.05010
45. Webb, S., Rainforth, T., Teh, Y.W., Kumar, M.P.: A statistical approach to assessing neural network robustness. In: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019. OpenReview.net (2019)
46. Weng, L., Chen, P., Nguyen, L.M., Squillante, M.S., Boopathy, A., Oseledets, I.V., Daniel, L.: PROVEN: verifying robustness of neural networks with a probabilistic approach. In: Chaudhuri, K., Salakhutdinov, R. (eds.) Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA. Proceedings of Machine Learning Research, vol. 97, pp. 6727–6736. PMLR (2019)
47. Weng, T.W., Zhang, H., Chen, H., Song, Z., Hsieh, C.J., Boning, D., Dhillon, I.S., Daniel, L.: Towards Fast Computation of Certified Robustness for ReLU Networks. In: ICML 2018 (Apr 2018)
48. Wicker, M., Huang, X., Kwiatkowska, M.: Feature-guided black-box safety testing of deep neural networks. In: Beyer, D., Huisman, M. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 24th International Conference, TACAS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10805, pp. 408–426. Springer (2018)
49. Wu, M., Wicker, M., Ruan, W., Huang, X., Kwiatkowska, M.: A game-based approximate verification of deep neural networks with provable guarantees. Theor. Comput. Sci. **807**, 298–329 (2020)
50. Yang, P., Li, R., Li, J., Huang, C., Wang, J., Sun, J., Xue, B., Zhang, L.: Improving neural network verification through spurious region guided refinement. CoRR **abs/2010.07722** (2020), https://arxiv.org/abs/2010.07722