Under consideration for publication in Formal Aspects of Computing

Enhancing Robustness Verification for Deep Neural Networks via Symbolic Propagation

Pengfei Yang[1,2], Jianlin Li[1,2], Jiangchao Liu[3], Cheng-Chao Huang[4],
Renjue Li[1,2], Liqian Chen[3], Xiaowei Huang[5], and Lijun Zhang[1,2,4]
[1] State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China
[2] University of Chinese Academy of Sciences, Beijing, China
[3] National University of Defense Technology, Changsha, China
[4] Institute of Intelligent Software, Guangzhou, China
[5] Department of Computer Science, University of Liverpool, Liverpool, UK

Abstract. Deep neural networks (DNNs) have been shown lack of robustness, as they are vulnerable to small perturbations on the inputs. This has led to safety concerns on applying DNNs to safety-critical domains. Several verification approaches based on constraint solving have been developed to automatically prove or disprove safety properties for DNNs. However, these approaches suffer from the scalability problem, i.e., only small DNNs can be handled. To deal with this, abstraction based approaches have been proposed, but are unfortunately facing the precision problem, i.e., the obtained bounds are often loose. In this paper, we focus on a variety of local robustness properties and a (δ, ε)-global robustness property of DNNs, and investigate novel strategies to combine the constraint solving and abstraction-based approaches to work with these properties:

- We propose a method to verify local robustness, which improves a recent proposal of analyzing DNNs through the classic abstract interpretation technique, by a novel symbolic propagation technique. Specifically, the values of neurons are represented *symbolically* and propagated from the input layer to the output layer, on top of the underlying abstract domains. It achieves significantly higher precision and thus can prove more properties.
- We propose a Lipschitz constant based verification framework. By utilising Lipschitz constants solved by semidefinite programming, we can prove global robustness of DNNs. We show how the Lipschitz constant can be tightened if it is restricted to small regions. A tightened Lipschitz constant can be helpful in proving local robustness properties. Furthermore, a global Lipschitz constant can be used to accelerate batch local robustness verification, and thus support the verification of global robustness.
- We show how the proposed abstract interpretation and Lipschitz constant based approaches can benefit from each other to obtain more precise results. Moreover, they can be also exploited and combined to improve constraints based approach.

We implement our methods in the tool PRODeep, and conduct detailed experimental results on several benchmarks.

Keywords: deep neural network; verification; robustness; abstract interpretation; symbolic propagation; Lipschitz constant

Correspondence and offprint requests to: Lijun Zhang. E-mail: zhanglj@ios.ac.cn

1. Introduction

The past few years have witnessed significant progress of deep neural networks (DNNs) in solving long-standing artificial intelligent tasks, such as nature language processing [HDY+12], image classification [KSH12], and game playing [SHM+16]. The technical progress has led to broad applications of DNNs to many industrial sectors, including automotive, health and social care, and digital finance. The performance of these DNNs, when measured with the prediction precision over a test dataset, is comparable to, or even better than, that of manually crafted software. Not surprisingly, especially for safety-critical applications, the DNNs should be certified with respect to *safety properties*.

The *robustness property* is one of the most important safety properties for DNNs. Intuitively, an input x, whose classification is the target y, is said to be *locally robust*, if all neighboring inputs x' are being classified as y as well. The L_∞-norm is most widely used in characterizing neighbourhood relations, due to its intuitive interpretation of the constraints for the input. Unfortunately, DNNs have been found lack of robustness. Specifically, [SZS+14] discovered that it is possible to add a small, or even imperceptible, perturbation to a correctly classified input and make it misclassified. Such adversarial examples have raised serious concerns on the safety of DNNs. If we consider a self-driving system controlled by a DNN, a failure on the recognization of a traffic light may lead to serious consequences because human lives are at stake.

Algorithms used to find adversarial examples are based on gradient descent (see e.g., [SZS+14, CW17]), saliency maps (see e.g., [PMJ+15]), evolutionary algorithm (see e.g., [NYC15]), etc. Roughly speaking, these are heuristic search algorithms without the guarantees to find the optimal values, that is to say, the bound on the gap between an obtained value and its ground truth is unknown. If an adversarial example is found, it demonstrates that the network is not robust at some input x. However, the certification of a robust input x needs provable guarantees. Thus, techniques based on formal verification have been developed. Up to now, DNN verification includes constraint-solving [PT10, KBD+17b, LM18, Ehl17, NKR+17, WK18, DSG+18], layer-by-layer exhaustive search [HKWW17, WHK18, WZC+18], global optimization [RHK18a], and abstract interpretation [GMDC+18, SGPV19b, SGM+18]. Abstract interpretation is a theory in static analysis which verifies a program by using sound approximation of its semantics [CC77]. Its basic idea is to use an abstract domain to over-approximate the computation on inputs. In [GMDC+18], this idea using abstract interpretation was first employed for verifying DNNs. However, abstract interpretation can be imprecise, due to the non-linearity in DNNs. The paper[SGM+18] implements a faster Zonotope domain for DNN verification and it can deal with more activation functions like sigmoid. In a later work, [SGPV19b] puts forward a new abstract domain specially for DNN verification and it is more efficient and precise than Zonotope.

Another useful way to characterize robustness is exploiting the Lipschitz continuity. From the perspective of functions, the Lipschitz constant is a measure of the sensitivity of a function, which indicates the maximum ratio between variations in the output space and variations in the input space. When viewing the DNN as a function characterizing it, its Lipschitz constant can be extremely useful in a variety of applications. A technique based on semidefinite programming can be used to compute guaranteed upper bounds on the Lipschitz constant of DNNs [FRH+19]. In this paper, we discuss how it can be used to verify robustness of DNNs. We consider first local robustness based on general L_p-norms. We discuss how L_1 and L_2-norms can be encoded as constraints for the inputs. Leveraging a property of Lipschitz continuous function, we show how to handle robustness properties based on L_p-norms. Moreover, we consider global robustness properties and establish their connections.

An overview of the contributions of the paper is given in Figure 1, which is detailed below.

- Firstly, this paper proposes a novel symbolic propagation technique to enhance the precision of abstract interpretation based DNN verification. This part is based on our previous paper [LLY+19]. For every neuron, we *symbolically* represent, with an expression, how its activation value can be determined by the activation values of neurons in previous layers. By both illustrative examples and experimental results, we show that, comparing with using only abstract domains, our new approach can find significantly tighter constraints over the neurons' activation values.
  In Fig. 1, the abstract interpretation is at the center. Because abstract interpretation is a sound approximation, with tighter constraints, we may feed the constraints to SMT based frameworks, which allow us to handle larger networks.
- Secondly, we consider global robustness properties, asserting that when two inputs are close (specified by δ), their values in the output stay also close by (specified by ϵ). By employing upper bound of Lipschitz constant, we give two methods for verifying the norm-based robustness properties of DNNs, both of which are proved to be sound. For local L_p-norm robustness, we present a method to compute the maximum verifiable radius for a certain input. For global robustness, we also present a method to verify the so-called (δ, ε)-global robustness. Comparing with abstract interpretation, Lipschitz based methods have complementary advantages. Lipschitz based methods

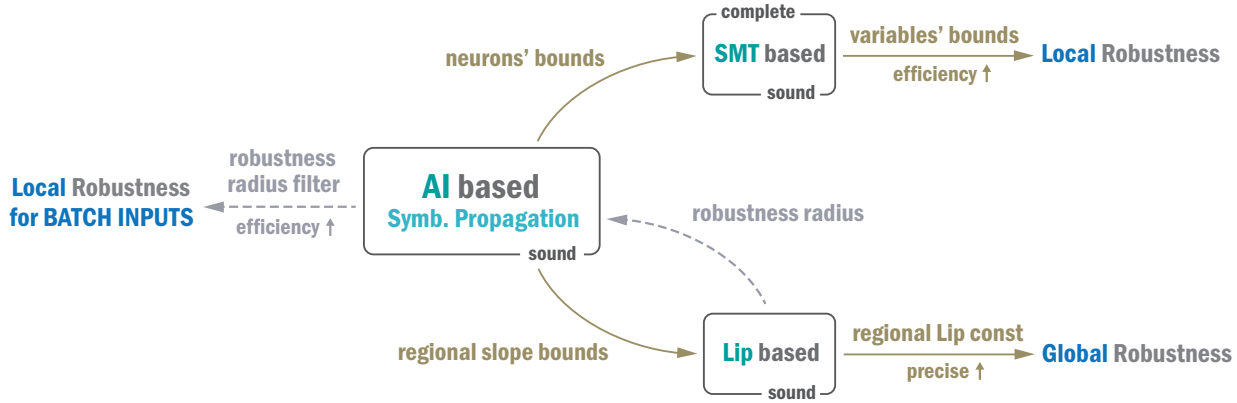


Fig. 1. Cooperation among three methods for robustness verification, in which AI, SMT and Lip are abbreviations for abstract interpretation, satisfiability modulo theories and Lipschtiz respectively.

can handle robustness related to general L_p-norm, where the regions cannot be directly abstracted by polyhedra (for example the L_2-norm). Also, they are more efficient, especially for a batch task with large number of inputs, because the Lipschitz constant can be reused to compute the maximum verifiable radius through simple arithmetic operations on the output values for different inputs. In other words, when the Lipschitz constant is obtained, the methods verify the robustness in a black-box way, regardless of the inner structure of the DNN.

- Thirdly, we further show how to intertwine the three verification methodologies, namely abstract interpretation, SMT and Lipschitz constant based approaches, to achieve better performance. In Fig. 1, the solid arrows demonstrate the improvements of the other two methods by invoking DeepSymbol [LLY+19].
  - For SMT based methods, DeepSymbol provides the bounds on hidden neurons, which indicate the ranges of the variables of the SMT problem encoded from a verification problem of local robustness. That will significantly accelerate the solving procedure (the SMT based DNN verifier Reluplex [KBD+17b]).
  - For Lipschitz constant based methods, DeepSymbol provides the bounds of slope restrictions of each activation functions in DNN, which can be used to compute a tighter upper approximation of the Lipschitz constant for a given region. The regional Lipschitz constant will be further used to verify the (δ, ε)-global robustness and leads to more precise results.

  The dashed arrows reversely demonstrates the auxiliary role of Lipschitz constant based methods in verifying local robustness properties.

  - When the global Lipschitz constant is obtained, a robustness radius w.r.t. L_p-norm can be computed in an efficient way for a certain input. So for the batch task of verifying local robustness, it can be used as a filter to quickly recognize the robustness cases whose regions are covered by their radii. That will often speed up the verification process.

We have implemented our approaches in the tool PRODeep [LLcH+20], and provided detailed experimental results on benchmark datasets such as MNIST and DNNs trained for the ACAS Xu system.

Organization of the paper. We provide preliminaries in Section 2. Robustness properties for DNNs are presented in Section 3. We present an overview of DNN verification methods in Section 4 and present our symbolic propagation technique in Section 5. In Section 6, we introduce the algorithm which uses semidefinite programming to calculate the Lipschitz constant of a given DNN and put forward a framework for verifying local and global robustness of DNNs with the Lipschitz constant. Experimental evaluation is shown in Section 7. Soundness guarantees and related works are further discussed in Section 8, and Section 9 concludes the paper.

2. Preliminaries

We recall some basic notions on deep neural networks and abstract interpretation. For a vector $\bar{x} \in \mathbb{R}^n$, we use x_i to denote its i-th entry. For a matrix $W \in \mathbb{R}^{m \times n}$, $W_{i,j}$ denotes the entry in its i-th row and j-th column.

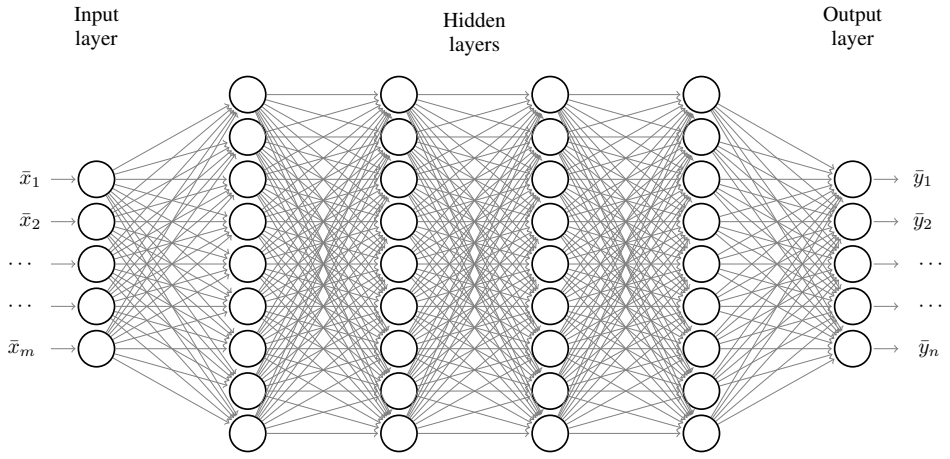


Fig. 2. A fully connected network: Each layer performs the composition of an affine transformation $\mathrm{Affine}(\bar{x}; W, b)$ and the activation function, where the coefficients of the matrix W are recorded on edges between neurons accordingly.

2.1. Deep neural networks

We work with deep feedforward neural networks, or DNNs, which can be represented as a function $f : \mathbb{R}^m \to \mathbb{R}^n$, mapping an input $\bar{x} \in \mathbb{R}^m$ to its corresponding output $\bar{y} = f(\bar{x}) \in \mathbb{R}^n$. In this work we consider DNNs for classification tasks. In this case, the output dimensions correspond to classification labels, and usually the label given by a DNN f is the one with the maximum output, i.e., $\arg\max_{1 \le i \le n} f(\bar{x})_i$. A DNN has in its structure a sequence of layers, including an input layer at the beginning, followed by several hidden layers, and an output layer in the end. Basically the output of a layer is the input of the next layer. To unify the representation, we denote the activation values at each layer as a vector. Thus the transformation between layers can also be seen as a function in $\mathbb{R}^{m'} \to \mathbb{R}^{n'}$. The DNN f is the composition of the transformations between layers, which is typically composed of an affine transformation followed by a non-linear activation function. In this paper we mainly consider one of the most commonly used activation functions – the rectified linear unit (ReLU) activation function, defined as

$$\mathrm{ReLU}(x) = \max(x, 0)$$

for $x \in \mathbb{R}$ and $\mathrm{ReLU}(\bar{x}) = (\mathrm{ReLU}(x_1), \ldots, \mathrm{ReLU}(x_n))$ for $\bar{x} \in \mathbb{R}^n$. Besides ReLU, there are other activation functions like sigmoid and $\tanh$, defined as

$$\sigma(x) = \frac{1}{1 + \mathrm{e}^{-x}}, \ \ \tanh(x) = \frac{\mathrm{e}^{x} - \mathrm{e}^{-x}}{\mathrm{e}^{x} + \mathrm{e}^{-x}}.$$

Typically an affine transformation is of the form $\mathrm{Affine}(\bar{x}; W, b) = W\bar{x} + b : \mathbb{R}^m \to \mathbb{R}^n$, where $W \in \mathbb{R}^{n \times m}$ and $b \in \mathbb{R}^n$. Mostly in DNNs we use a **fully connected layer** to describe the composition of an affine transformation $\mathrm{Affine}(\bar{x}; W, b)$ and the activation function, if the coefficient matrix W is not sparse and does not have shared parameters. We call a DNN with only fully connected layers a fully connected neural network (FNN). Fig. 2 gives an intuitive description of fully connected layers and fully connected networks. Apart from fully connected layers, we also have affine transformations whose coefficient matrices are sparse and have many shared parameters, like **convolutional layers**. Readers can refer to e.g. [GMDC+18] for its formal definition. In our paper, we do not specially deal with convolutional layers, because they can be regarded as common affine transformations. In the architecture of DNNs, a convolutional layer is often followed by a non-linear **max pooling layer**, which takes as an input a three dimensional vector $\bar{x} \in \mathbb{R}^{m \times n \times r}$ with two parameters p and q which divide m and n respectively, defined as

$$\mathrm{MaxPool}_{\mathrm{p,q}}(\bar{x})_{i,j,k} = \max\{x_{i',j',k} \mid i' \in (p \cdot (i-1), p \cdot i] \ \wedge \ j' \in (q \cdot (j-1), q \cdot j]\}.$$

We call a DNN with only fully connected, convolutional, and max pooling layers a convolutional neural network (CNN).

In the rest of the paper, we let the DNN f have N layers, each of which has m_k neurons, for $0 \le k < N$. Therefore, $m_0 = m$ and $m_{N-1} = n$.

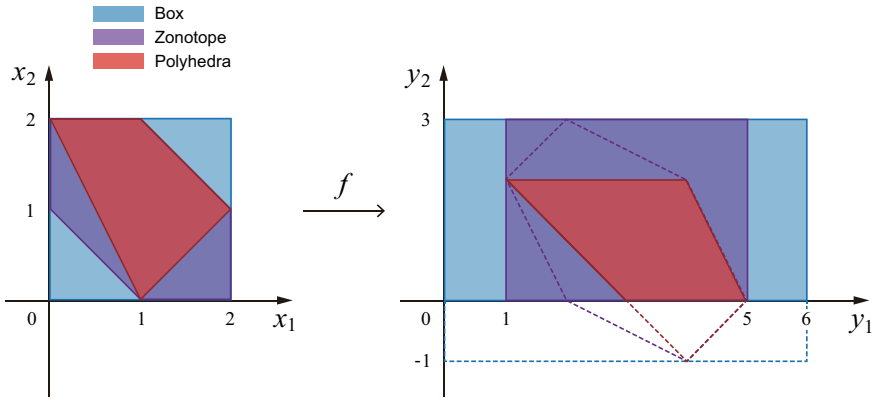


Fig. 3. An illustration of Example 2.1 and Example 4.2, where on the right the dashed lines give the abstraction region before the ReLU operation and the full lines give the final abstraction $f^\sharp(X^\sharp)$.

2.2. Abstract interpretation

Abstract interpretation is a theory in static analysis which verifies a program by using sound approximation of its semantics [CC77]. Its basic idea is to use an abstract domain to over-approximate the computation on inputs and propagate it through the program. In the following, we describe its adaptation to work with DNNs.

Generally, on the input layer, we have a concrete domain $\mathcal{C}$, which includes a set of inputs X as one of its elements. To enable an efficient computation, we choose an abstract domain $\mathcal{A}$ to infer the relation of variables in $\mathcal{C}$. We assume that there is a partial order $\sqsubseteq$ on $\mathcal{C}$ as well as $\mathcal{A}$, which in our settings is the subset relation $\subseteq$. We have a concretization function $\gamma : \mathcal{A} \to \mathcal{C}$ which assigns each abstract element $a \in \mathcal{A}$ to its concrete element $\gamma(a) \in C$, which is the concrete semantics of the abstract element a. Note that, $a \in \mathcal{A}$ is a sound abstraction of $c \in \mathcal{C}$ if and only if $c \sqsubseteq \gamma(a)$. Intuitively, here soundness guarantees that a sound abstraction is an over-approximation of the concrete element.

In abstract interpretation, it is important to choose a suitable abstract domain because it determines the efficiency and precision of the abstract interpretation. In practice, we use specific types of constraints to represent the abstract elements. Geometrically, a certain type of constraints corresponds to a special shape. E.g., the conjunction of a set of arbitrary linear constraints corresponds to a polyhedron. Abstract domains that are suitable for for verifying DNN include Box, Zonotope [GGP09, GGP10], and Polyhedra, etc. We briefly recall them and give an example showing intuitively how these three abstract domains work in the following.

Box. A box B contains bound constraints of the form of $a \leq x_i \leq b$. The conjunction of bound constraints expresses a box in the Euclidean space. The form of the constraint for each dimension is an interval, and thus it is also named the Interval abstract domain.

Zonotope. A zonotope Z consists of constraints of the form of $z_i = a_i + \sum_{j=1}^{m} b_{ij}\epsilon_j$, where a_i, b_{ij} are real constants and ϵ_j is bounded by a constant interval $[l_j, u_j]$. The conjunction of these constraints express a center-symmetric polyhedra in the Euclidean space.

Polyhedra. A Polyhedron P has constraints of the form of linear inequalities, i.e., $\sum_{i=1}^{n} a_i x_i + b \leq 0$ and it gives a closed convex polyhedron in the Euclidean space.

Example 2.1. Let $\bar{x} \in \mathbb{R}^2$, and the possible values of $\bar{x}$ be $X = \{(1,0),(0,2),(1,2),(2,1)\}$. With Box, we can abstract the inputs X as $[0,2] \times [0,2]$, and with Zonotope, X can be abstracted as

$$\left\{ x_1 = 1 - \frac{1}{2}\epsilon_1 - \frac{1}{2}\epsilon_3,\ \ x_2 = 1 + \frac{1}{2}\epsilon_1 + \frac{1}{2}\epsilon_2 \right\}.$$

where $\epsilon_1, \epsilon_2, \epsilon_3 \in [-1, 1]$. With Polyhedra, X can be abstracted as

$$\{x_2 \leq 2,\ \ x_2 \leq -x_1 + 3,\ \ x_2 \geq x_1 - 1,\ \ x_2 \geq -2x_1 + 2\}.$$

Fig. 3 (the left part) gives an intuitive description for the three abstractions.

3. Robustness Properties

The problem of verifying DNNs with respect to a robustness property can be stated formally as follows.

Definition 3.1. ([GMDC+18]) Given a function $f : \mathbb{R}^m \to \mathbb{R}^n$ which expresses a DNN, a set of inputs $X_0 \subseteq \mathbb{R}^m$, and a property $C \subseteq \mathbb{R}^n$, verifying the property is to determine whether $T_f(X_0) \subseteq C$ holds, where $T_f(X_0) := \{f(\bar{x}) \mid \bar{x} \in X_0\}$. We write such a property (f, X_0, C).

Verification of robustness properties over DNNs is contained in this problem. In this section we formally introduce robustness properties in different settings, which are standard in DNN verification.

3.1. Local robustness

For a DNN, local robustness mainly focuses on the consistency of output labels in a neighborhood of a certain input. Following the Def. 3.1, we obtain the local robustness property by letting X_0 be a neighborhood of an input $\bar{x}$ with the output label l, and defining C to be the set $\{\bar{y} \in \mathbb{R}^n \mid \arg\max_{1\leq i\leq n} y_i = l\}$. Different ways to define the neighborhood X_0 lead to the following definitions of local robustness.

3.1.1. Box-Based Robustness

Using the intervals of each variables is a basic way to define a robustness region, which is a box (or hyperrectangle). As its name suggests, we focus on the robustness in the region which is the Cartesian product of constant intervals containing the given input. Formally, for $\bar{\alpha}, \bar{\beta} \in \mathbb{R}^m_{\geq 0}$,

$$\text{Neighborhood}_{\text{Box}}(\bar{x}, \bar{\alpha}, \bar{\beta}) = \{\bar{x}' \in \mathbb{R}^m \mid \bar{x}_i - \bar{\alpha}_i \leq \bar{x}'_i \leq \bar{x}_i + \bar{\beta}_i, 1 \leq i \leq m\}.$$

In addition to giving the bounds of each variables in an explicit way, many other classes of neighborhoods of an input can be reduced to a box.

L_∞-norm By bounding the L_∞-norm, the region named L_∞ ball can be defined as $B_\infty(\bar{x}, r) = \{\bar{x}' \in \mathbb{R}^n \mid \|\bar{x}' - \bar{x}\|_\infty \leq r\}$. The L_∞ norm is most widely used to characterise norm-based robustness for the following reasons. First, an L_∞ ball is a box region intuitively, which can be precisely represented by the box domain, so all the tools mentioned in this paper can deal with such input constraints. Also, the L_∞ based robustness usually has an explicit meaning in DNN models. For example, in image recognition, the L_∞ based robustness gives a upper bound of the disturbance on all the pixels, but other L_p-norms, like the L_2-norm, are not so intuitive in this setting. Last but not least, the L_∞ based robustness is stronger than L_p based robustness, since it is a standard result that $\|\bar{x}\|_\infty \leq \|\bar{x}\|_p$ for any $\bar{x}$.

Brightness robustness In image recognition, brightness attack is a common way of attacking DNNs: It allows pixels with brightness greater than $1 - \delta$ to become brighter. Formally, the robustness region of this brightness attack is

$$\text{Neighborhood}_{\text{Brightness}}(\bar{x}, \delta) = \{\bar{x}' \in \mathbb{R}^m \mid \forall i, 1 - \delta \leq \bar{x}_i \leq \bar{x}'_i \leq 1 \vee \bar{x}_i = \bar{x}'_i\}.$$

Brightness attack also describes a box region of the input layer (but not an L_∞ ball), so SMT based and abstract interpretation based methods can both deal with it precisely.

3.1.2. Norm-Based Robustness

A typical way to define neighbourhood of an input is to use norm distance, especially the L_p-norm. Many attacking approaches make perturbation based on some L_p-norm to generate adversarial examples [MMS+18, SRBB19, TB19], and [NWL19] proposes a method to defend from attacks based on L_p-norms.

Formally the L_p-norm on $\mathbb{R}^n$ is a function $\|\cdot\|_p : \mathbb{R}^n \to [0, \infty)$ which assigns a point $\bar{x} \in \mathbb{R}^n$ to a non-negative value, and we can use $\|\bar{x} - \bar{x}'\|_p$ to characterize the L_p distance between $\bar{x}$ and $\bar{x}'$. For $1 \leq p < \infty$, the L_p-norm is defined as

$$\|\bar{x}\|_p = \left(\sum_{i=1}^{n} |x_i|^p\right)^{\frac{1}{p}},$$

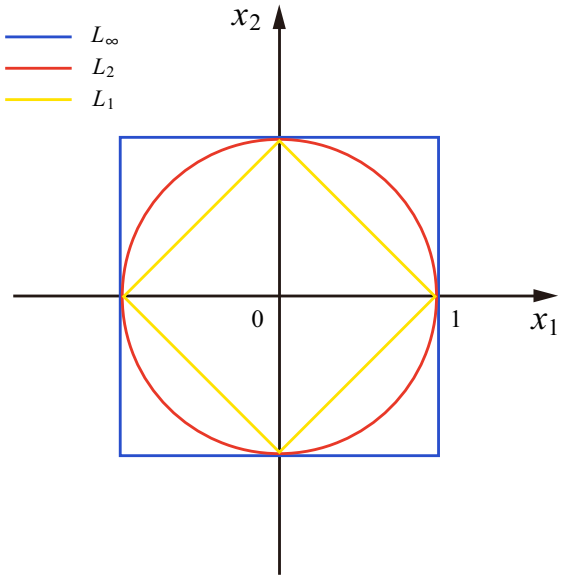


Fig. 4. The boundaries of unit L_1, L_2 and L_∞ balls in $\mathbb{R}^2$.

and the L_∞ norm as mentioned above can be regarded as the L_p-norm with p tending to infinity, and it can be explicitly expressed as

$$\|\bar{x}\|_\infty = \max_{1\le i\le n} |x_i|.$$

As the case of the L_∞ ball, the neighbourhood of an input $\bar{x}$ bounded by the L_p-norm can be described as an L_p ball: The L_p (closed) ball with the center $\bar{x} \in \mathbb{R}^n$ and the radius $r > 0$ is defined as

$$B_p(\bar{x}, r) = \{\bar{x}' \in \mathbb{R}^n \mid \|\bar{x}' - \bar{x}\|_p \le r\}.$$

Fig. 4 gives the boundaries of the unit L_1, L_2 and L_∞ balls in $\mathbb{R}^2$. As is shown in Fig. 4, except for L_∞ norm, the robustness region defined by L_p-norm cannot be abstracted to a box precisely in general.

General L_p-norm For $1 < p < \infty$, the constraints of an L_p ball is no longer linear, and they can not be encoded precisely by the tools mentioned in this paper, like SMT-based tools (Reluplex, Planet, etc.), and abstract interpretation based tools (ERAN, DeepSymbol, etc.). Although the L_∞ based robustness implies L_p based robustness, it may result in a big loss of precision: For instance, the Lebesgue measure of $B_\infty(0, 1)$ is $n!$ times that of $B_1(0, 1)$ in $\mathbb{R}^n$, so $B_\infty(0, 1)$ may not be a good abstraction of $B_1(0, 1)$.

Up to now, there have been a few approaches to dealing with general L_p based robustness, and they are based on the Lipschitz continuity of the network. A function $f : \mathbb{R}^m \to \mathbb{R}^n$ is Lipschitz continuous, if there exists $\mathcal{L} > 0$, s.t. for any $\bar{x}, \bar{x}' \in \mathbb{R}^m$,

$$\|f(\bar{x}) - f(\bar{x}')\|_2 \le \mathcal{L} \cdot \|\bar{x} - \bar{x}'\|_2.$$

Here we can get an over-approximation of the output range of a Lipschitz continuous function on an L_2 ball if we know its Lipschitz constant $\mathcal{L}$. Also, by using the inequality $\|\bar{x}\|_p \le n^{\frac{1}{p}-\frac{1}{q}}\|\bar{x}\|_q$, where $\bar{x} \in \mathbb{R}^n$, we can obtain the output range with any L_p-norm on the input and the output layers as

$$m^{-(\frac{1}{p}-\frac{1}{2})}\|f(y) - f(x)\|_p \le \|f(y) - f(x)\|_2 \le \mathcal{L}\|y - x\|_2 \le n^{\frac{1}{2}-\frac{1}{q}}\mathcal{L}\|y - x\|_q, \tag{1}$$

where n and m are the dimension of the input layer and the output layer, respectively. In [RHK18b], it is proved that deep neural networks are Lipschitz continuous, and the authors provide an algorithm to calculate a Lipschitz constant of a given DNN. A more efficient and accurate algorithm for calculating a Lipschitz constant of a DNN is proposed in [FRH+19], where the authors pose the Lipschitz estimation problem as semidefinite programming (SDP). Finally, Fast-Lip [WZC+18] is an algorithm to over-approximate the output range of a given DNN on an L_p ball using its Lipschitz constant. These Lipschitz continuity based methods can help verify general L_p based robustness, and they

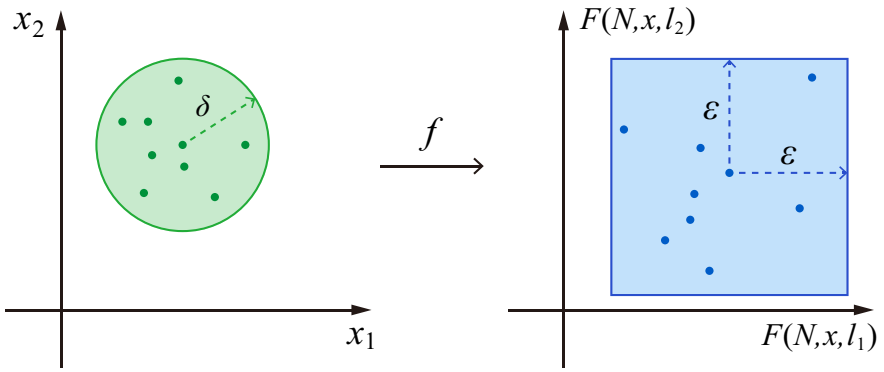


Fig. 5. (δ, ε)-global robustness: the absolute differences of outputs are bounded by ε when the distances between the inputs are bounded by δ through the L_2-norm. Here $F(N, \bar{x}, l)$ is the value of the output node indicating the label $l \in L$ for a given DNN N with an input $\bar{x}$.

can work on DNNs with most activation functions including ReLU. The disadvantage of such methods is the low precision since the use of the Lipschitz constant is even further from the precise reachability; it characterises the behaviour of a DNN by an upper bound of the local change rate.

3.2. Global robustness

For a DNN, global robustness focuses on the global behavior of all inputs in a certain region. Specifically, the (δ, ε)-global robustness constrains the absolute differences between each coordinates of outputs by the norm distances of pairs inputs (see Fig. 5). In [KBD$^+$17a], the (δ, ε)-global robustness was defined as below.

Definition 3.2 (Def. 2 of [KBD$^+$17a]). A DNN f is (δ, ε)-globally robust in the input region D if

$$\forall \bar{x}, \bar{x}' \in D,\ ||\bar{x} - \bar{x}'||_2 \leq \delta \ \Rightarrow\ \|f(x) - f(x')\|_\infty < \epsilon.$$

Fig. 5 gives an intuitive explanation of global robustness.

Although global robustness does not necessarily imply a corresponding local robustness property, yet they still have a close relationship in that global robustness actually gives a valid Lipschitz constant, which is helpful in local robustness verification. Compared with local robustness, a global robustness property is generally more difficult to verify, because it is often difficult to consider the whole high dimensional input space.

4. Methods for Verifying Local Robustness

In this section, we review two main classes of techniques for verifying local robustness properties. First, we recall constraint based DNN verification algorithms based on SMT solvers. Then, we describe how to use abstract interpretation to verify DNNs.

4.1. SMT based methods

In [KBD$^+$17b, Ehl17], two SMT solvers Reluplex and Planet were presented to verify DNNs. Typically an SMT solver is the combination of a SAT solver with the specialized decision procedures for other theories. The verification of DNNs uses linear arithmetic over real numbers, in which an atom may have the form of $\sum_{i=1}^{n} w_i x_i \leq b$, where w_i and b are real numbers. Both Reluplex and Planet use the DPLL algorithm to split cases and rule out conflict clauses. They are different in dealing with the intersection. For Reluplex, it inherits rules from the Simplex algorithm and adds a few rules dedicated to ReLU operation. Through the classical pivot operation, it searches for a solution to the linear constraints, and then applies the rules for ReLU to ensure the ReLU relation for every node. Differently, Planet uses linear approximation to over-approximate the DNN, and manages the conditions of ReLU and max pooling nodes with logical formulas.

The following example shows that how we encode a DNN verification problem into an SMT problem.

Example 4.1. Consider the toy network $f(\bar{x}) = \mathrm{ReLU}\left(\begin{pmatrix}1 & 2\\ 1 & -1\end{pmatrix}\bar{x} + \begin{pmatrix}0\\ 1\end{pmatrix}\right)$. Basically we write all the transformations in the networks, namely $y_1 = x_1 + 2x_2$, $y_2 = x_1 - x_2 + 1$, $z_1 = \mathrm{ReLU}(y_1)$, and $z_2 = \mathrm{ReLU}(y_2)$. In order to verify a property (f, X, C), we only need to put the constraints above, along with the constraints of X and the negation of C as the input to the SMT solver. If the SMT solver returns SAT, then we have found a counterexample to violate the property, or otherwise the property is successfully verified.

Up to now, the state-of-art SMT based tools include Reluplex, Planet, and Marabou, and they support only piecewise linear activation functions, and linear constraints for input and output. SMT-based methods are theoretically sound and complete, but their time complexity is proved to be NP-complete, so they do not scale on large networks.

4.2. Abstract interpretation based methods

Under the framework of abstract interpretation, to conduct verification of DNNs, we first need to choose an abstract domain $\mathcal{A}$. Then we represent the set of inputs of a DNN as an abstract element (value) $X_0^\sharp$ in $\mathcal{A}$. After that, we pass it through the DNN layers by applying abstract transformers of the abstract domain. Recall that N is the number of layers in a DNN and m_k is the number of nodes in the k-th layer. Let f_k (where $1 \leq k < N$) be the layer function mapping from $\mathbb{R}^{m_{k-1}}$ to $\mathbb{R}^{m_k}$. We can lift f_k to $T_{f_k} : \mathcal{P}(\mathbb{R}^{m_{k-1}}) \to \mathcal{P}(\mathbb{R}^{m_k})$ such that $T_{f_k}(X) = \{f_k(\bar{x}) \mid \bar{x} \in X\}$.

Definition 4.1. An abstract transformer $T_{f_k}^\sharp$ is a function mapping an abstract element $X_{k-1}^\sharp$ in the abstract domain $\mathcal{A}$ to another abstract element ${X^\sharp}_k$. Moreover, $T_{f_k}^\sharp$ is sound if $T_{f_k} \circ \gamma \subseteq \gamma \circ T_{f_k}^\sharp$.

Intuitively, a sound abstract transformer $T_{f_k}^\sharp$ maintains a sound relation between the abstract post-state and the abstract pre-state of a transformer in DNN (such as linear transformation, ReLU operation, etc.).

Let $X_k = f_k(...(f_1(X_0)))$ be the exact set of resulting vectors in $\mathbb{R}^{m_k}$ (i.e., the k-th layer) computed over the concrete inputs X_0, and ${X^\sharp}_k = {T_{f_k}}^\sharp(...({T_{f_1}}^\sharp({X^\sharp}_0)))$ be the corresponding abstract value of the k-th layer when using an abstract domain $\mathcal{A}$. Note that $X_0 \subseteq \gamma({X^\sharp}_0)$. We have the following conclusion.

Proposition 4.1. If $X_{k-1} \subseteq \gamma({X^\sharp}_{k-1})$, then we have $X_k \subseteq \gamma({X^\sharp}_k) = \gamma \circ T_{f_k}^\sharp({X^\sharp}_{k-1})$.

Proof. Because $T_{f_k}^\sharp$ is a sound abstract transformer, we have

$$X_k = T_f(X_{k-1}) \subseteq T_f(\gamma({X^\sharp}_{k-1})) \subseteq \gamma \circ T_{f_k}^\sharp(\gamma({X^\sharp}_{k-1})) = \gamma({X^\sharp}_k).$$

□

Therefore, when performing abstract interpretation over the transformations in a DNN, the abstract pre-state ${X^\sharp}_{k-1}$ is transformed into abstract post-state ${X^\sharp}_k$ by applying the abstract transformer $T_{f_k}^\sharp$ which is built on top of an abstract domain. This procedure starts from $k = 1$ and continues until reaching the output layer (and getting ${X^\sharp}_{N-1}$). Finally, we use ${X^\sharp}_{N-1}$ to check the property C as follows:

$$\gamma({X^\sharp}_{N-1}) \subseteq C. \tag{2}$$

The following theorem states that this verification procedure based on abstract interpretation is sound for the DNN verification problem.

Theorem 4.1. If Equation (2) holds, then $T_f(X_0) \subseteq C$.

Proof. By induction on N, it is easy to see that $T_f(X_0) \subseteq \gamma({X^\sharp}_{N-1})$, so Equation (2) implies $T_f(X_0) \subseteq C$. □

It's not hard to see that the other direction does not necessarily hold due to the potential incompleteness caused by the over-approximation made in both the abstract elements and the abstract transformers $T_{f_k}^\sharp$ in an abstract domain.

Example 4.2. Suppose that $\bar{x}$ takes the value in X given in Example 2.1, and we consider the transformation $f(\bar{x}) = \mathrm{ReLU}\left(\begin{pmatrix}1 & 2\\ 1 & -1\end{pmatrix}\bar{x} + \begin{pmatrix}0\\ 1\end{pmatrix}\right)$. Now we use the three abstract domains to calculate the resulting abstraction.

- Box. The abstraction after the affine transformation is $[0, 6] \times [-1, 3]$, and thus the final result is $[0, 6] \times [0, 3]$.

- Zonotope. After the affine transformation, the zonotope abstraction can be obtained straightforward:

$$\left\{ y_1 = 3 + \frac{1}{2}\epsilon_1 + \epsilon_2 - \frac{1}{2}\epsilon_3,\ \ y_2 = 1 - \epsilon_1 - \frac{1}{2}\epsilon_2 - \frac{1}{2}\epsilon_3 \mid \epsilon_1, \epsilon_2, \epsilon_3 \in [-1, 1] \right\}.$$

  The first dimension y_1 is definitely positive, so it remains the same after the ReLU operation. The second dimension y_2 can be either negative or non-negative, so its abstraction after ReLU will become a box which only preserves the range in the non-negative part, i.e. $[0, 3]$, so the final abstraction is

$$\left\{ y_1 = 3 + \frac{1}{2}\epsilon_1 + \epsilon_2 - \frac{1}{2}\epsilon_3,\ \ y_2 = \frac{3}{2} + \frac{3}{2}\eta_1 \mid \epsilon_1, \epsilon_2, \epsilon_3, \eta_1 \in [-1, 1] \right\},$$

  whose concretization is $[1, 5] \times [0, 3]$.
- Polyhedra. It is easy to obtain the polyhedron before ReLU: $P_1 = \{y_2 \leq 2, y_2 \geq -y_1 + 3, y_2 \geq y_1 - 5, y_2 \leq -2y_1 + 10\}$. Similarly, the first dimension is definitely positive, and the second dimension can be either negative or non-negative, so the resulting abstraction is $(P_1 \wedge (y_2 \geq 0)) \vee (P_1 \wedge (y_2 = 0))$, i.e. $\{y_2 \leq 2, y_2 \geq -y_1 + 3, y_2 \geq 0, y_2 \leq -2y_1 + 10\}$.

Fig. 3 (the right part) gives an illustration for the abstract interpretation with the three abstract domains in this example.

The abstract value computed via abstract interpretation can be directly used to verify properties. Take the local robustness property, which expresses an invariance on the classification of f over a region $B(\bar{x}_0, \delta)$, as an example. Let $l_i(\bar{x})$ be the confidence of $\bar{x}$ being labeled as i, and $l(\bar{x}) = \mathrm{argmax}_i l_i(\bar{x})$ be the label. It has been shown in [SZS+14, RHK18a] that DNNs are Lipschitz continuous. Therefore, when δ is small, we have that $|l_i(\bar{x}) - l_i(\bar{x}_0)|$ is also small for all labels i. That is, if $l_i(\bar{x}_0)$ is significantly greater than $l_j(\bar{x}_0)$ for $j \neq i$, it is highly likely that $l_i(\bar{x})$ is also significantly greater than $l_j(\bar{x})$. It is not hard to see that the more precise the relations among $l_i(\bar{x}_0), l_i(\bar{x}), l_j(\bar{x}_0), l_j(\bar{x})$ computed via abstract interpretation, the more likely we can prove the robustness. Based on this rational, this paper aims to derive techniques to enhance the precision of abstract interpretation such that it can prove properties that cannot be proven by the original abstract interpretation approach.

5. Optimisations by Symbolic Propagation

To improve the effectiveness and the efficiency of robustness verification by abstract interpretation, we introduce symbolic propagation to take advantage of the linearity in most part of the DNNs. Comparing with traditional abstract interpretation based methods, this method will significantly improve in terms of the precision and memory usage. Futhermore, by providing neurons' bounds computed by our method, it is possible to accelerate the verification of SMT-based methods.

5.1. DeepSymbol: Symbolic propagation for DNN abstract interpretation

Symbolic propagation can ensure soundness while providing more precise results. In [WPW+18], a technique called symbolic interval propagation is present and we extend it to our abstract interpretation framework so that it works on all abstract domains. First, we use the following example to show that using only abstract transformations in an abstract domain may lead to precision loss, while using symbolic propagation could enhance the precision.

Example 5.1. Assume that we have a two-dimensional input $(x_1, x_2) \in [0, 1] \times [0, 1]$ and a few transformations $y_1 := x_1 + x_2$, $y_2 := x_1 - x_2$, and $z := y_1 + y_2$. Suppose we use the Box abstract domain to analyze the transformations.

- When using only the Box abstract domain, we have $y_1 \in [0, 2]$, $y_2 \in [-1, 1]$, and thus $z \in [-1, 3]$ (i.e., $[0, 2] + [-1, 1]$).
- By symbolic propagation, we record $y_1 = x_1 + x_2$ and $y_2 = x_1 - x_2$ on the neurons y_1 and y_2 respectively, and then get $z = 2x_1 \in [0, 2]$. This result is more precise than that given by using only the Box abstract domain.

Non-relational (e.g., intervals) and weakly-relational abstract domains (e.g., zones, octagons, zonotopes, etc.)[Min17] may lose precision on the application of the transformations from DNNs. The transformations include affine transformations, ReLU, and max pooling operations. Moreover, it is often the case for weakly-relational abstract domains that the composition of the optimal abstract transformers of individual statements in a sequence does not result in the

optimal abstract transformer for the whole sequence, which has been shown in Example 3 when using only the Box abstract domain. An option to precisely handle general linear transformations is to use the Polyhedra abstract domain which uses a conjunction of linear constraints as domain representation. However, the Polyhedra domain has worst-case exponential space and time complexity when handling the ReLU operation (via the join operation in the abstract domain). As a consequence, DNN verification with the Polyhedra domain is impractical for large scale DNNs, which has been also confirmed in [GMDC$^+$18].

In this paper, we leverage symbolic propagation technique to enhance the precision for abstract interpretation based DNN verification. The insight behind is that affine transformations account for a large portion of the transformations in a DNN. Furthermore, when we verify properties such as robustness, the activation of a neuron can often be deterministic for inputs around an input with small perturbation. Hence, there should be a large number of linear equality relations that can be derived from the composition of a sequence of linear transformations via symbolic propagation. And we can use such linear equality relations to improve the precision of the results given by the abstract domains. In Section 7, our experimental results confirm that, when the perturbation tolerance δ is small, there is a significant proportion of neurons whose ReLU activations are consistent, i.e., they are always activated or deactivated.

First, given X_0, a ReLU neuron $y := \mathrm{ReLU}(\sum_{i=1}^{n} w_i x_i + b)$ can be classified into one of the following 3 categories (according to its range information): (1) definitely-activated, if the range of $\sum_{i=1}^{n} w_i x_i + b$ is a subset of $[0, \infty)$, (2) definitely-deactivated, if the range of $\sum_{i=1}^{n} w_i x_i + b$ is a subset of $(-\infty, 0]$, and (3) uncertain, otherwise.

Now we detail our symbolic propagation technique. We first introduce a symbolic variable s_i for each node i in the input layer, to denote the initial value of that node. For a ReLU neuron $d := \mathrm{ReLU}(\sum_{i=1}^{n} w_i c_i + b)$ where c_i is a symbolic variable, we make use of the resulting abstract value of abstract domain at this node to determine whether the value of this node is definitely greater than 0 or definitely less than 0. If it is a definitely-activated neuron, we record for this neuron the linear combination $\sum_{i=1}^{n} w_i c_i + b$ as its symbolic representation (i.e., the value of symbolic propagation). If it is a definitely-deactivated neuron, we record for this neuron the value 0 as its symbolic representation. Otherwise, we cannot have a linear combination as the symbolic representation and thus a fresh symbolic variable s_d is introduced to denote the output of this ReLU neuron. We also record the bounds for s_d, such that the lower bound for s_d is set to 0 (since the output of a ReLU neuron is always non-negative) and the upper bound keeps the one obtained by abstract interpretation.

To formalize the algorithm for ReLU node, we first define the abstract states in the analysis and three transfer functions for linear assignments, condition tests and joins respectively. An abstract state in our analysis is composed of:

- an abstract element for a numeric domain (e.g., Box) $\mathbf{n}^\sharp \in \mathbf{N}^\sharp$,
- a set of free symbolic variables $\mathbf{C}$ (those not equal to any linear expressions),
- a set of constrained symbolic variables $\mathbf{S}$ (those equal to a certain linear expression), and
- a map from constrained symbolic variables to linear expressions $\xi ::= \mathbf{S} \to \{\sum_{i=1}^{n} a_i x_i + b \mid x_i \in \mathbf{C}\}$. Note that we only allow free variables in the linear expressions in ξ.

In Algorithm 1, we show how to compute the transfer functions for linear assignments $[\![y := \sum_{i=1}^{n} w_i x_i + b]\!]^\sharp$ which over-approximates the behaviors of $y := \sum_{i=1}^{n} w_i x_i + b$. In the beginning, all input variables are taken as free symbolic variables. If $n > 0$ (i.e., the right value expression is not a constant), the variable y is added to the constrained variable set $\mathbf{S}$. All constrained variables in $\sum_{i=1}^{n} w_i x_i + b$ are replaced by their corresponding expressions in ξ, with the resulting expression denoted by $\mathbf{expr}$. Then, the map from y to the new $\mathbf{expr}$ is recorded in ξ. Abstract numeric element $\mathbf{n}^\sharp$ is updated by the transfer function for assignments in the numeric domain $[\![y := \mathbf{expr}]\!]^\sharp_{\mathbf{N}^\sharp}$ (note that we use $[\![\cdot]\!]^\sharp_{\mathbf{N}^\sharp}$ to denote the transfer function in the numeric domain $\mathbf{N}^\sharp$). If $n \leq 0$, the right-value expression is a constant, then y is added to $\mathbf{C}$, and is removed from $\mathbf{S}$ and ξ.

The abstract transfer function for conditional test is defined as

$$[\![\mathbf{expr} \leq 0]\!]^\sharp(\mathbf{n}^\sharp, \mathbf{C}, \mathbf{S}, \xi) ::= [\![\mathbf{expr} \leq 0]\!]^\sharp_{\mathbf{N}^\sharp}(\mathbf{n}^\sharp, \mathbf{C}, \mathbf{S}, \xi),$$

which only updates the abstract element $\mathbf{n}^\sharp$ by the transfer function in the numeric domain $\mathbf{N}^\sharp$.

The join algorithm in our analysis is defined in Algorithm 2. Only the constrained variables arising in both input $\mathbf{S}_0$ and $\mathbf{S}_1$ with the same corresponding linear expressions are taken as constrained variables. The abstract element in the result is obtained by applying the join operator in the numeric domain $\sqcup_{\mathbf{N}^\sharp}$.

Algorithm 1: Transfer function for linear assignments $[\![y := \sum_{i=1}^{n} w_i x_i + b]\!]^\sharp$

Input: abstract numeric element $\mathbf{n}^\sharp \in \mathbf{N}^\sharp$, free variables $\mathbf{C}$, constrained variables $\mathbf{S}$, symbolic map ξ
$\mathbf{expr} \leftarrow \sum_{i=1}^{n} w_i x_i + b$
When the right value expression is not a constant
if $n > 0$ **then**
  for $i \in [1, n]$ **do**
    if $x_i \in \mathbf{S}$ **then**
      $\mathbf{expr} = \mathbf{expr}_{|x_i \leftarrow \xi(x_i)}$
    end
  end
  $\xi = \xi \cup \{y \mapsto \mathbf{expr}\} \quad \mathbf{S} = \mathbf{S} \cup \{y\} \quad \mathbf{C} = \mathbf{C} \setminus \{y\} \quad \mathbf{n}^\sharp = [\![y := \mathbf{expr}]\!]^\sharp_{\mathbf{N}^\sharp}$
else
  $\xi = \xi \setminus (y \mapsto *) \quad \mathbf{C} = \mathbf{C} \cup \{y\} \quad \mathbf{S} = \mathbf{S} \setminus \{y\} \quad \mathbf{n}^\sharp = [\![y := \mathbf{expr}]\!]^\sharp_{\mathbf{N}^\sharp}$
end
return $(\mathbf{n}^\sharp, \mathbf{C}, \mathbf{S}, \xi)$

The transfer function for a ReLU node is defined as

$$[\![y := \mathrm{ReLU}(\sum_{i=1}^{n} w_i x_i + b)]\!]^\sharp(\mathbf{n}^\sharp, \mathbf{C}, \mathbf{S}, \xi) ::= \mathbf{join}([\![y \geq 0]\!]^\sharp(\psi), [\![y := 0]\!]^\sharp([\![y < 0]\!]^\sharp)(\psi)),$$

where $\psi = [\![y := \sum_{i=1}^{n} w_i x_i + b]\!]^\sharp(\mathbf{n}^\sharp, \mathbf{C}, \mathbf{S}, \xi)$. For $y \geq 0$, the output of a ReLU node is the same as the input, and for $y < 0$, it outputs 0, so the transformer outputs the join of these two cases.

Algorithm 2: Join algorithm **join**

Input: $(\mathbf{n}_0^\sharp, \mathbf{C}_0, \mathbf{S}_0, \xi_0)$ and $(\mathbf{n}_1^\sharp, \mathbf{C}_1, \mathbf{S}_1, \xi_1)$
$\mathbf{n}^\sharp = \mathbf{n}_0^\sharp \sqcup_{\mathbf{N}^\sharp} \mathbf{n}_1^\sharp$
$\xi = \xi_0 \cap \xi_1$
$\mathbf{S} = \{x \mid \exists \mathbf{expr}, x \to \mathbf{expr} \in \xi\}$
$\mathbf{C} = \mathbf{C}_0 \cup (\mathbf{S}_0 \setminus \mathbf{S})$
return $(\mathbf{n}^\sharp, \mathbf{C}, \mathbf{S}, \xi)$

For a max pooling node $d := \max_{1 \leq i \leq k} c_i$, if there exists some c_j whose lower bound is larger than the upper bound of c_i for all $i \neq j$, we set c_j as the symbolic representation for d. Otherwise, we introduce a fresh symbolic variable s_d for d and record its bounds wherein its lower (upper) bound is the maximum of the lower (upper) bounds of c_i's. Note that the lower (upper) bound of each c_i can be derived from the abstract value for this neuron given by the abstract domain.

The algorithm for max-pooling layers can be defined with the three aforementioned transfer functions as follows:

$$\begin{aligned} &\mathbf{join}(\phi_1, \mathbf{join}(\phi_2, \ldots, \mathbf{join}(\phi_{k-1}, \phi_k))), \\ \text{where} \quad &\phi_i = [\![d := c_i]\!]^\sharp [\![c_i \geq c_1]\!]^\sharp \ldots [\![c_i \geq c_k]\!]^\sharp(\mathbf{n}^\sharp, \mathbf{C}, \mathbf{S}, \xi). \end{aligned}$$

Here ϕ_i represents the case that the variable c_i is the maximum, and the abstract transformer outputs the join of all possible cases ϕ_i.

Example 5.2. For the DNN shown in Figure 6 (a), there are two input nodes denoted by symbolic variables x and y, two hidden nodes, and one output node. The initial ranges of the input symbolic variables x and y are given, i.e., $[4, 6]$ and $[3, 4]$ respectively. The weights are labeled on the edges. It is not hard to see that, when using the Interval abstract domain, (the inputs of) the two hidden nodes have bounds $[17, 24]$ and $[0, 3]$ respectively. For the hidden node with $[17, 24]$, we know that this ReLU node is definitely activated, and thus we use symbolic propagation to get a symbolic expression $2x + 3y$ to symbolically represent the output value of this node. Similarly, for the hidden node with $[0, 3]$, we get a symbolic expression $x - y$. Then for the output node, symbolic propagation results in $x + 4y$,

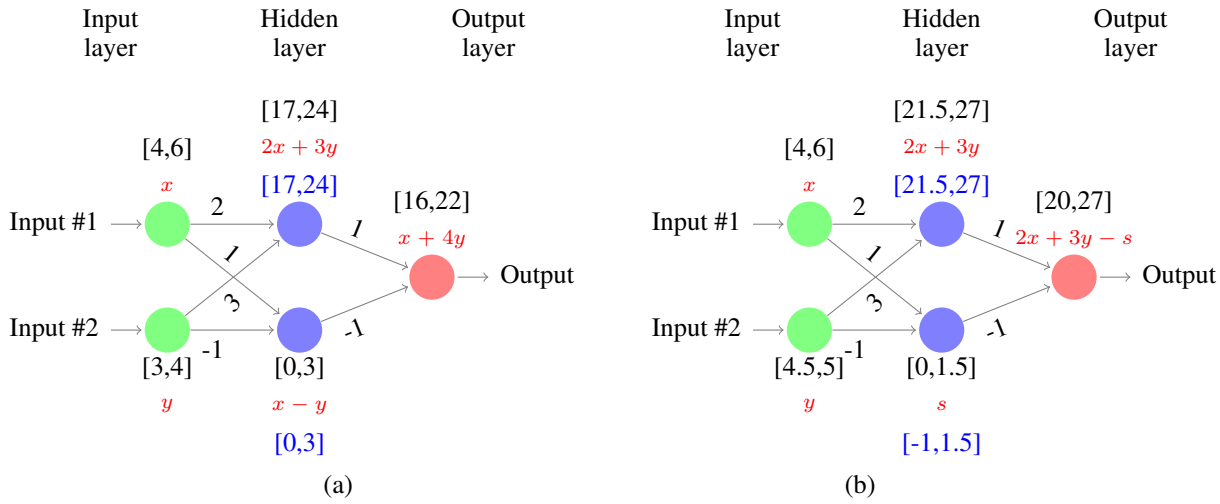


Fig. 6. An illustrative example of symbolic propagation

which implies that the output range of the whole DNN is $[16, 22]$. If we use only the Interval abstract domain without symbolic propagation, we will get the output range $[14, 24]$, which is less precise than $[16, 22]$.

For the DNN shown in Figure 6 (b), we change the initial range of the input variable y to be $[4.5, 5]$. For the hidden ReLU node with $[-1, 1.5]$, it is neither definitely activated nor definitely deactivated, and thus we introduce a fresh symbolic variable s to denote the output of this node, and set its bound to $[0, 1.5]$. For the output node, symbolic propagation results in $2x + 3y - s$, which implies that the output range of the whole DNN is $[20, 27]$.

For a definitely-activated neuron, we utilize its symbolic representation to enhance the precision of abstract domains. We add the linear constraint $d == \sum_{i=1}^{n} w_i c_i + b$ into the abstract value at (the input of) this node, via the meet operation (which is used to deal with conditional test in a program) in the abstract domain [CC77]. If the precision of the abstract value for the current neuron is improved, we may find more definitely-activated neurons in the subsequent layers. In other words, the analysis based on abstract domain and our symbolic propagation mutually improves the precision of each other on-the-fly.

After obtaining symbolic representation for all the neurons in a layer k, the computation proceeds to layer $k + 1$. The computation terminates after completing the computation for the output layer. All symbolic representations in the output layer are evaluated to obtain value bounds.

The following theorem shows some results on precision of our symbolic propagation technique.

Theorem 5.1. In the following, Zonotope all refers to [GGP09, GGP10], whose implementation is included in Apron.

(1) For an FNN $f : \mathbb{R}^m \to \mathbb{R}^n$ and a box region $X \subseteq \mathbb{R}^m$, the Box abstract domain with symbolic propagation gives a more precise abstraction for $f(X)$ than the Zonotope abstract domain without symbolic propagation.

(2) For an FNN $f : \mathbb{R}^m \to \mathbb{R}^n$ and a box region $X \subseteq \mathbb{R}^m$, the Box abstract domain with symbolic propagation and the Zonotope abstract domain with symbolic propagation give the same abstraction for $f(X)$.

(3) There exists a CNN $g : \mathbb{R}^m \to \mathbb{R}^n$ and a box region $X \subseteq \mathbb{R}^m$ s.t. the Zonotope abstract domain with symbolic propagation give a more precise abstraction for $g(X)$ than the Box abstract domain with symbolic propagation.

Proof. (1) Since an FNN only contains fully connected layers, we just need to prove that, Box with symbolic propagation (i.e., BoxSymb) is always more precise than Zonotope in the transformations on each ReLU neuron $y := \mathrm{ReLU}(\sum_{i=1}^{n} w_i x_i + b)$. Assume that before the transformation, BoxSymb is more precise or as precise as Zonotope. Since the input is a Box region, the assumption is valid in the beginning. Then we consider three cases: (a) in BoxSymb, the sign of $\sum_{i=1}^{n} w_i x_i + b$ is uncertain, then it must also be uncertain in Zonotope. In both domains, a constant interval with upper bound computed by $\sum_{i=1}^{n} w_i x_i + b$ and lower bound as 0 is assigned to y (this can be inferred from our aforementioned algorithms and [GGP09]). With our assumption, the upper bound computed by BoxSymb is more precise than that in Zonotope; (b) in BoxSymb, the sign of $\sum_{i=1}^{n} w_i x_i + b$ is always positive, then it must be always positive or uncertain in Zonotope. In the former condition, BoxSymb is more precise because it loses no precision, while Zonotope can lose precision because of its limited expressiveness. In the latter condition, BoxSymb is more

precise obviously; (c) in BoxSymb, the sign of $\sum_{i=1}^{n} w_i x_i + b$ is always negative, then it must be always negative or uncertain in Zonotope. Similar to case (b), BoxSymb is also more precise in this case.

(2) Assume that before each transformation on a ReLU neuron $y := \mathrm{ReLU}(\sum_{i=1}^{n} w_i x_i + b)$, BoxSymb and ZonoSymb (Zonotope with symbolic propagation) have the same precision. This assumption is valid when the input is a Box region. Then the evaluation of $\sum_{i=1}^{n} w_i x_i + b$ is the same in BoxSymb and ZonoSymb, thus in the three cases:(a) the sign of $\sum_{i=1}^{n} w_i x_i + b$ is uncertain, and they both compute the same constant interval for y; (b) and (c) $\sum_{i=1}^{n} w_i x_i + b$ is always positive or negative, and they both lose no precision.

(3) It is easy to know that ZonoSymb is more precise or as precise as BoxSymb in all transformations. In CNN, with Max-Pooling layer, we just need to give an example that ZonoSymb can be more precise. Let the Zonotope $X' = \{x_1 = 2+\epsilon_1+\epsilon_2,\ \ x_2 = 2+\epsilon_1-\epsilon_2 \mid \epsilon_1, \epsilon_2 \in [-1, 1]\}$ and the max pooling node $y = \max\{x_1, x_2\}$. Obviously X' can be obtained through a linear transformation on some box region X. With Box with symbolic propagation, the abstraction of y is $[0, 4]$, while Zonotope with symbolic propagation gives the abstraction is $[1, 4]$. □

Thm 5.1 gives us some insights: The symbolic propagation technique is strong (even stronger than Zonotope) in dealing with ReLU nodes, while Zonotope gives a more precise abstraction on max pooling nodes. It also provides useful advice: When we work with FNNs with the input range being a box, we should use Box with symbolic propagation rather than Zonotope with symbolic propagation since it does not improve the precision but takes more time. Results related to Thm 5.1 are also illustrated in our experiments.

5.2. Accelerating SMT-based verification

Now we describe how to utilize the results of abstract interpretation with our symbolic propagation to improve the performance of SMT-based DNN verification approaches.

Generally speaking, there is a huge bottleneck in efficiency of SMT-based DNN verification approaches, e.g., relying on case splitting for ReLU operation. In the worst case, case splitting is needed for each ReLU operation in a DNN, which leads to an exponential blow-up. In particular, when analyzing large-scale DNNs, SMT-based DNN verification approaches may suffer from the scalability problem and account time out, which is also confirmed experimentally in [GMDC$^+$18].

In this paper, we utilize the results of abstract interpretation (with symbolic propagation) to accelerate SMT-based DNN verification approaches. More specifically, we use the bound information of each ReLU node (obtained by abstract interpretation) to reduce the number of case-splitting, and thus accelerate SMT-based DNN verification. For example, on a neuron $d := \mathrm{ReLU}(\sum_{i=1}^{n} w_i c_i + b)$, if we know that this node is a definitely-activated node according to the bounds given by abstract interpretation, we only consider the case $d := \sum_{i=1}^{n} w_i c_i + b$ and thus no split is applied. We remark that this does not compromise the precision of SMT-based DNN verification while improving their efficiency.

6. Lipschitz Constant Based Method

The Lipschitz constant of a DNN measures the maximum ratio between variations in the output space and variations in the input space, which can be useful in DNN verification, especially when the properties are related to general L_p-norms with $1 < p < \infty$, whose regions cannot be expressed by linear constraints.

In this section we give a Lipschitz constant based method (Lip. method for short) for robustness verification. We first need to obtain the Lipschitz constant of the DNN. The problem of computing an upper approximation of the Lipschitz constant can be encoded into an SDP problem, which can be solved effectively by optimization.

For verifying the local robustness properties related to L_p-norm, we present a method to compute a lower approximation of the maximum robustness radius. In what follows, we also call it maximum verifiable radius. We also present a method to verify the (δ, ε)-global robustness of a DNN in a given region.

6.1. Approximating from above the Lipschitz constant by SDP

In this section we briefly review an algorithm to compute the Lipschitz constant of a given DNN proposed by Fazlyab et al. [FRH$^+$19]. Basically, this algorithm encodes the condition of Lipschitz continuity as semi-positive definiteness of a matrix, and the calculation of the tight Lipschitz constant is reduced to a semidefinite programming (SDP) problem.

The difficulty of calculating the Lipschitz constant of a DNN is the non-linear activation functions, where we usually have the form $\phi(\bar{x}) = (\varphi(x_1), \ldots, \varphi(x_n))^\top$ of repeated non-linearity, where $\varphi : \mathbb{R} \to \mathbb{R}$ is the activation function like ReLU, sigmoid, $\tanh$, etc. Here we require that the function φ should be increasing. The Lipschitz continuity of a DNN relies on the slope restriction of the activation function involved, defined as:

Definition 6.1. A function $\varphi : \mathbb{R} \to \mathbb{R}$ is *slope-restricted* on the interval $[\alpha, \beta]$, if for any $x, y \in \mathbb{R}$ satisfying $x \neq y$,

$$\alpha \leq \frac{\varphi(y) - \varphi(x)}{y - x} \leq \beta.$$

The activation functions mentioned above are all slope-restricted: Especially, ReLU, $\tanh$, and max pooling are all slope-restricted on $[0, 1]$, and sigmoid is slope-restricted on $[0, 0.25]$. When we have an activation function φ which is slope-restricted on $[\alpha, \beta]$, then the function $\phi(\bar{x}) = (\varphi(x_1), \ldots, \varphi(x_n))^\top$ satisfies the following incremental quadratic constraint:

Lemma 6.1. ([FRH$^+$19]) Suppose $\varphi : \mathbb{R} \to \mathbb{R}$ is slope-restricted on $[\alpha, \beta]$. Define

$$\mathcal{T}_n = \{T \in \mathbb{S}_n \mid T = \sum_{i=1}^{n} \lambda_{ii} e_i e_i^\top + \sum_{1 \leq i < j \leq n} \lambda_{ij}(e_i - e_j)(e_i - e_j)^\top, \lambda_{ij} \geq 0\},$$

where $\mathbb{S}_n$ is the set of symmetric matrices in $\mathbb{R}^{n \times n}$, and $e_i \in \mathbb{R}^n$ is the vector in which the i-th entry is 1 and the others 0. Then for any $T \in \mathcal{T}_n$, the function $\phi(\bar{x}) = (\varphi(x_1) \ \cdots \ \varphi(x_n))^\top : \mathbb{R}^n \to \mathbb{R}^n$ satisfies

$$\begin{pmatrix} \bar{x} - \bar{y} \\ \phi(\bar{x}) - \phi(\bar{y}) \end{pmatrix}^\top \begin{pmatrix} -2\alpha\beta T & (\alpha + \beta)T \\ (\alpha + \beta)T & -2T \end{pmatrix} \begin{pmatrix} \bar{x} - \bar{y} \\ \phi(\bar{x}) - \phi(\bar{y}) \end{pmatrix} \geq 0.$$

Lemma 6.1 provides a connection between Lipschitz continuity and semidefiniteness of a matrix. Now we consider a single-layer DNN $f(\bar{x}) = W^1\phi(W^0\bar{x} + \bar{b}^0) + \bar{b}^1$, where $\phi(\bar{x}) = (\varphi(x_1) \ \cdots \ \varphi(x_n))^\top$ and φ is slope-restricted on $[\alpha, \beta]$. The following theorem shows how the Lipschitz continuity is connected with an SDP problem.

Theorem 6.1. ([FRH$^+$19]) If there exists $\rho > 0$ and $T \in \mathcal{T}_n$, s.t.

$$M(\rho, T) := \begin{pmatrix} -2\alpha\beta W^{0\top} T W^0 - \rho I_{n_0} & (\alpha + \beta) W^{0\top} T \\ (\alpha + \beta) T W^0 & -2T + W^{1\top} W^1 \end{pmatrix} \leq 0,$$

where $\mathcal{T}_n$ is what we define in Lemma 6.1, n_0 is the dimension of the input, and ≤ 0 refers to the semi-negative definiteness of a symmetric matrix, then $\sqrt{\rho}$ is a Lipschitz constant for f.

To calculate a tighter Lipschitz constant, we only need to solve the optimisation problem

$$\min \rho \text{ s.t. } M(\rho, T) \leq 0 \wedge T \in \mathcal{T}_n.$$

For a multi-layer DNN, we can also extend Thm. 6.1 to tackle multiple layers and transform the problem into an SDP. Readers can refer to [FRH$^+$19] for details. In this SDP problem, we have $O(n^2)$ variables λ_{ij} to determine, where n is the number of neurons in the DNN. To make the problem scalable, [FRH$^+$19] proposed the following modes.

- LipSDP-Network: it preserves all possible constraints between pairs and has $O(n^2)$ decision variables.
- LipSDP-Neuron: it ignores constraints between different neurons so that the matrix T is diagonal, which has $O(n)$ decision variables.
- LipSDP-Layer: it ignores the difference of variables in the same layer, i.e., on the same layer the matrix T shares the same parameter, so the matrix $T = \mathrm{diag}(\lambda_1 I_{n_1}, \ldots, \lambda_l I_{n_l})$ has $O(l)$ decision variables, where l is the number of layers in the DNN.

The above approach works on DNNs with only one kind of activation function φ. For those containing more than one kind of activation function, we can make split to the layers of the DNN and consider each sub-network which only has one kind of activation function. After adapting the algorithm for these sub-network, the multiplication of the Lipschitz constants of the sub-networks is a valid Lipschitz constant for the DNN.

6.2. Regionalization of Lipschitz constant by local slope bounds

An observation is that, when we adapt this algorithm to some robustness verification settings, like (δ, ε)-global robustness in a certain region, we do not require the Lipschitz constant over the whole input space $\mathbb{R}^n$. For a fixed region D we are focusing on, we discuss how to obtain a more precise Lipschitz constant, by exploiting the local slope bounds used in SDP solving. By invoking the abstract interpretation techniques (DeepSymbol we proposed), we can obtain the range of all the input variables of the activation function φ, and further calculate the range of the derivative of φ. For many popular activation function φ like ReLU, sigmoid, and $\tanh$, which are monotonic increasing, convex on $(-\infty, 0]$, and concave on $[0, \infty)$, the range of its derivative can be obtained straightforward by the function values at the endpoints of the interval of the input. The following example shows how we compute the local slope bounds and obtain the regional Lipschitz constant. In what follows, we denote by $\mathcal{L}_D$ the lower approximation of the regional Lipschitz constant for an input region D.

Example 6.1. Consider a DNN defined by $f(x) = \tanh\left(\begin{pmatrix}1 & 2\\ 1 & -1\end{pmatrix} x + \begin{pmatrix}-2\\ 0\end{pmatrix}\right)$. We can obtain the Lipschitz constant $\mathcal{L} = 2.303$ by solving the corresponding SDP problem.

Now we further consider the neighbourhood $D = B_\infty((1,0)^\top, 0.1)$, then by using Box as the abstract domain, we obtain the ranges of two $\tanh$ inputs are $[-1.1, -0.9]$ and $[0.9, 1.1]$, and the bounds of the slope of $\tanh$ in this region are

$$\alpha = \varphi'(1.1) = 1 - \tanh^2(1.1) = 0.3592 \qquad \text{and} \qquad \beta = \varphi'(0.9) = 1 - \tanh^2(0.9) = 0.4869.$$

Finally, by using the local slope bounds of the region, we can compute the regional Lipschitz constant $\mathcal{L}_D = 1.122$.

We can see from the example that the bounds of the slope of the activation function involved depend on the region. This regionalization works well when we consider a very small region, and the range of slope is reduced prominently such that we can obtain a more precise Lipschitz constant for a certain region. Furthermore, to maintain the high efficiency, we tend to keep the lower bound α being 0 instead of a more precise but insignificant improvement, which will preserve the sparsity of $M(\rho, T)$.

6.3. Computing the maximum verifiable radius

Focusing on local robustness, once the lower approximation of the Lipschitz constant $\mathcal{L}$ is obtained, we can use it to efficiently compute the maximum verifiable radius for a certain input. A sound method directly follows from the following lemma, whose proof can be obtained by classical technique.

Lemma 6.2. Consider a DNN defined by $f(x) : \mathbb{R}^n \to \mathbb{R}^m$, whose Lipschitz constant is bounded from above by $\mathcal{L}$. Then for an input $\bar{x}$, the DNN is robust in $B_p(\bar{x}, r_p)$ with

$$r_p = \frac{2^{\frac{1}{p}-1} m^{\frac{1}{2}-\frac{1}{p}}}{n^{\frac{1}{2}-\frac{1}{p}}\mathcal{L}} (f_{\tau_1}(\bar{x}) - f_{\tau_2}(\bar{x})),$$

where $f_{\tau_1}(\bar{x})$ and $f_{\tau_2}(\bar{x})$ denote the largest and the second largest elements in $\{f_\tau(\bar{x})\}$, respectively.

Proof. Generally, for L_p-norm robustness, we consider the neighbourhood $B_p(\bar{x}, r)$ of an input $\bar{x}$. According to Inequality (1) on Page 7, we have the Lipschitz condition $m^{-(\frac{1}{p}-\frac{1}{2})}||f(\bar{x}') - f(\bar{x})||_p \le n^{\frac{1}{2}-\frac{1}{p}}\mathcal{L}r$ for any $\bar{x}'$ in $B_p(\bar{x}, r)$. Note that n and m here are the dimensions of $\bar{x}$ and $f(\bar{x})$, viz. the dimensions of the input layer and the output layer.

We denote by τ_i the index of the element which is i-th largest in $\{f_\tau(\bar{x})\}$. Then the DNN is robust over $B_p(\bar{x}, r)$ if and only if $\bigwedge_{\tau\neq\tau_1} f_{\tau_1}(\bar{x}') > f_\tau(\bar{x}')$ for any $\bar{x}' \in B_p(\bar{x}, r)$. With $\Delta_i = f_i(\bar{x}') - f_i(\bar{x})$, the Lipschitz condition is equivalent to

$$(|\Delta_1|^p + |\Delta_2|^p + \cdots + |\Delta_m|^p)^{\frac{1}{p}} \le \frac{n^{\frac{1}{2}-\frac{1}{p}}}{m^{\frac{1}{2}-\frac{1}{p}}}\mathcal{L}r, \tag{3}$$

and the robustness condition can be further rewritten as

$$\bigwedge_{\tau\neq\tau_1} f_{\tau_1}(\bar{x}) - f_\tau(\bar{x}) > \Delta_\tau - \Delta_{\tau_1}. \tag{4}$$

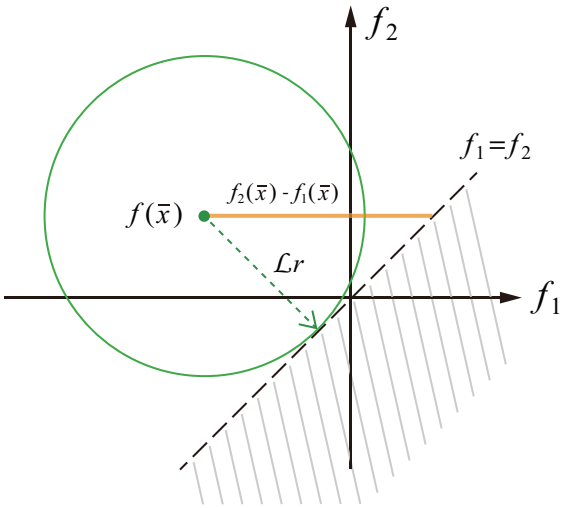


Fig. 7. The L_2-norm radius at the output layer, where the yellow line represents the difference between $f_2(\bar{x})$ and $f_1(\bar{x})$, and the dashed green line shows the distance between $f(\bar{x})$ and the classification boundary, whose upper bound is given by both $\mathcal{L} \cdot r$ and Corollary 6.1.

From (3), we have

$$\Delta_\tau - \Delta_{\tau_1} \leq |\Delta_\tau| + |\Delta_{\tau_1}| \leq \frac{2(|\Delta_\tau|^p + |\Delta_{\tau_1}|^p)^{\frac{1}{p}}}{2^{\frac{1}{p}}} \leq \frac{2n^{\frac{1}{2}-\frac{1}{p}}}{2^{\frac{1}{p}} m^{\frac{1}{2}-\frac{1}{p}}} \mathcal{L}r,$$

which follows from Jensen's inequality and the fact that x^p is convex on $\mathbb{R}_{\geq 0}$. Finally, we can immediately obtain

$$r_p = \frac{2^{\frac{1}{p}-1} m^{\frac{1}{2}-\frac{1}{p}}}{n^{\frac{1}{2}-\frac{1}{p}} \mathcal{L}} (f_{\tau_1}(\bar{x}) - f_{\tau_2}(\bar{x}))$$

as the maximum verifiable radius to ensure the inequality (4) holds. □

Specifically, we have the maximum verifiable radius $r_2 = \frac{f_{\tau_1} - f_{\tau_2}}{\sqrt{2}\mathcal{L}}$ for the L_2-norm. Intuitively, it demonstrates that for any $f(\bar{x}')$, the decrement from $f_{\tau_1}(\bar{x})$ and the increment from $f_{\tau_2}(\bar{x})$ should be bounded by $\frac{\mathcal{L} \cdot r}{\sqrt{2}}$, otherwise the L_2-norm robustness may be destroyed. Similarly, we also have the maximum verifiable radius $r_\infty = \frac{f_{\tau_1} - f_{\tau_2}}{2\mathcal{L}'}$ for L_∞ norm, in which $\mathcal{L}' = \sqrt{\frac{n}{m}}\mathcal{L}$.

Corollary 6.1. Consider a DNN defined by $f(x) : \mathbb{R}^n \to \mathbb{R}^m$. Let $\mathcal{L}_D$ be the upper bound of the regional Lipschitz constant of the neighbourhood $D = B_q(\bar{x}, r)$. Then the DNN is robust in $B_p(\bar{x}, r_p)$ (for $p \leq q$) with

$$r_p = \min\left(r, \frac{2^{\frac{1}{p}-1} m^{\frac{1}{2}-\frac{1}{p}}}{n^{\frac{1}{2}-\frac{1}{p}} \mathcal{L}_D} (f_{\tau_1}(\bar{x}) - f_{\tau_2}(\bar{x}))\right),$$

where $f_{\tau_1}(\bar{x})$ and $f_{\tau_2}(\bar{x})$ denote the largest and the second largest elements in $\{f_\tau(\bar{x})\}$, respectively.

This corollary directly follows from the fact that $\mathcal{L}_D$ only holds in the region D and the fact that $B_p(\bar{x}, r) \subseteq B_q(\bar{x}, r)$ for $p \leq q$.

Example 6.2. Consider the L_2-norm local robustness of the DNN in Example 6.1. To compute the maximum verifiable radius, we first compute the output $f(\bar{x}) = (-1.238, 0.761)^\top$. Then by the formula in Lemma 6.2, we immediately have $r_2 = \frac{0.761-(-1.238)}{\sqrt{2}\cdot 2.303} = 0.613$, which implies that the DNN is robust in $B_\infty((1,0)^\top, 0.613)$. Fig. 7 illustrates the geometric significance of the formula, in which $f_1 = f_2$ is the robustness boundary at the output layer.

Furthermore, by using the regional Lipschitz constant $\mathcal{L}_D$ of $D = B_\infty((1,0)^\top, 0.1)$, we can also compute another robustness radius as $r_2 = \max\left(0.1, \frac{0.761-(-1.238)}{\sqrt{2}\cdot 1.122}\right) = 0.1$. Note that it is not as large as the robustness radius

computed by the global Lipschitz constant. This case shows that regional Lipschitz constant will not always benefit to obtain a better robustness radius, since the choice of the region may become a limitation.

Since the global Lipschitz constant holds for the whole domain of the DNN, Lemma 6.2 can be used to efficiently compute sound robustness radii w.r.t. L_p-norm local robustness for any inputs. If we want to verify local robustness for a large set of inputs, this technique can be used as a filter to quickly recognize the robustness cases and speed up the verification process.

6.4. Verifying (δ, ε)-global robustness in a region

Now, we turn to the (δ, ε)- robustness properties for regions of inputs. The following lemma shows a sufficient condition such that a DNN is globally-robust in an input region. It implies a sound method to verify global robustness.

Lemma 6.3. Let N be a DNN expressing $f : \mathbb{R}^m \to \mathbb{R}^n$, and D be a region of its input with the upper approximation of the regional Lipschitz constant $\mathcal{L}_D$. Then N is (δ, ε)-globally robust in the region D, if $\varepsilon > m^{-\frac{1}{2}}\mathcal{L}_D\delta$.

Proof. We have $C(N, \bar{x}, \ell) = f_\ell(\bar{x})$. By Inequality (1), we have

$$m^{\frac{1}{2}}||f(\bar{x}_1) - f(\bar{x}_2)||_\infty \leq \mathcal{L}_D||\bar{x}_1 - \bar{x}_2||_2.$$

Furthermore, the post-condition $\forall \ell \in L,\ |C(N, \bar{x}_1, \ell) - C(N, \bar{x}_2, \ell)| < \varepsilon$ is exactly equivalent to an $\mathcal{L}_\infty$-norm bound on the difference of two outputs, viz. $||f(\bar{x}_1) - f(\bar{x}_2)||_\infty < \varepsilon$. Then combining the pre-condition in the definition of (δ, ε)-global robustness, we immediately have the lemma. □

Example 6.3. Here, we verify the global robustness of the DNN of Example 6.1 in the box region $D = [0.9, 1.1] \times [-0.1, 0.1]$. Consider the global robustness with parameters $\delta = 0.06$, $\varepsilon = 0.04$, and L_2-norm in the pre-condition. With the upper approximation of the regional Lipschitz constant $\mathcal{L}_D = 1.122$ for the region D, it is easy to verify that the DNN is $(0.04, 0.06)$-globally robust in D, since $0.06 > \frac{1}{\sqrt{2}} \cdot 1.122 \cdot 0.04$ holds. Note that in this example, we fail to verify global robustness using the global Lipschitz constant. Compared with global Lipschitz constant, a regional Lipschitz constant will always show greater advantages for verifying (δ, ε)-global robustness, since the region in which we expect to verify is known.

7. Experimental Evaluation

In this section, we present the design and results of our experiments.

7.1. Experimental setup

Implementation AI^2[GMDC$^+$18] is the first to utilize abstract interpretation to verify DNNs, and has implemented all the transformers mentioned in Section 4. We have re-implemented these transformers and refer to them as AI^2-r. We then implement our symbolic propagation technique based on AI^2-r and use AI^2-r as the baseline comparison in the experiments. Both implementations use general frameworks and thus can run on various abstract domains. In this paper, we choose Box (from Apron [1]), T-Zonotope (Zonotope from Apron [1]) and E-Zonotope (Elina Zonotope with the join operator [2]) as the underlying domains. Our DNN verification tool PRODeep [LLcH$^+$20] integrates these methods. We use the code [3] from [FRH$^+$19] to calculate the Lipschitz constant.

Datasets We use MNIST [LBBH98] and ACAS Xu [JGK$^+$15, vEG14] as the datasets in our experiments. MNIST contains $60,000$ 28×28 grayscale handwritten digits. We can train DNNs to classify the pictures by the written digits on them. The ACAS Xu system is aimed to avoid airborne collisions and it uses an observation table to make decisions for the aircraft. In [JKO18], the observation tables are realized by training a DNN instead of storing it.

[1] https://github.com/ljlin/Apron_Elina_fork
[2] https://github.com/eth-sri/ELINA/commit/152910bf35ff037671c99ab019c1915e93dde57f
[3] https://github.com/arobey1/LipSDP

	AI^2-r		Symb			Planet
	TZono	EZono	Box	TZono	EZono	
FNN1	28.23348%	28.02098%	9.69327%	9.69327%	9.69327%	7.05553%
FNN2	24.16382%	22.13319%	1.76704%	1.76704%	1.76704%	0.89089%
FNN3	26.66453%	26.30852%	6.88656%	6.88656%	6.88656%	4.51223%
FNN4	28.47243%	28.33535%	5.13645%	5.13645%	5.13645%	2.71537%
FNN5	35.61163%	35.27187%	3.34578%	3.34578%	3.34578%	0.14836%
FNN6	38.71020%	38.57376%	7.12480%	7.12480%	7.12480%	1.94230%
FNN7	41.76517%	41.59382%	5.52267%	5.52267%	5.52267%	1h TIMEOUT
CNN1	24.19607%	24.13725%	21.78279%	7.58917%	7.56223%	8h TIMEOUT
CNN2	OOM	OOM	1.09146%	OOM	OOM	8h TIMEOUT

(a) Bound proportions (smaller is better) of different abstract interpretation approaches with the robustness bound $\delta \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6\}$, and the fixed input pictures 767, 1955, and 2090;

	AI^2-r						Symb						Planet	
	Box		TZono		EZono		Box		TZono		EZono			
FNN1	11.168	0.2	13.482	0.5	44.05	0.5	12.935	0.6	17.144	0.6	45.88	0.6	20.179	0.6
FNN2	12.559	0	16.636	0.2	50.59	0.2	15.075	0.5	22.333	0.5	49.92	0.5	35.84	0.6
FNN3	12.699	0.2	18.748	0.3	49.812	0.3	19.042	0.6	28.128	0.6	54.77	0.6	76.106	0.6
FNN4	15.583	0.1	29.495	0.3	58.892	0.3	37.716	0.6	56.47	0.6	76.00	0.6	351.139	0.6
FNN5	28.963	0	81.49	0.2	149.791	0.2	90.268	0.4	154.222	0.4	173.263	0.4	1297.485	0.6
FNN6	62.766	0	398.565	0.1	538.076	0.1	323.328	0.3	650.629	0.3	745.454	0.3	15823.208	0.3
FNN7	111.955	0	1674.465	0	1627.72	0	642.978	0.3	1524.975	0.3	1489.604	0.3	1h TIMEOUT	
CNN1	2340.828	0	6717.57	0.2	94504.195	0.2	5124.681	0.2	8584.555	0.3	45452.102	0.3	8h TIMEOUT	
CNN2	41292.291	0	OOM	0	OOM	0	105850.271	0.3	OOM	0	OOM	0	8h TIMEOUT	

(b) The time (in second) and the maximum robustness bound δ which can be verified through the abstract interpretation technique and the planet bound, with optional $\delta \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6\}$ and the fixed input picture 2090;

	AI^2-r									Symb									Planet
	Box			TZono			EZono			Box			TZono			EZono			
FNN1(60)	57	44	34	59	52	38	59	52	38	59	53	44	59	53	44	59	53	44	59 56 55
FNN2(120)	103	59	38	118	109	66	118	111	66	118	113	107	118	113	107	118	113	107	119 114 110
FNN3(150)	136	93	66	141	127	85	141	127	85	143	133	110	143	133	110	143	133	110	146 142 135
FNN4(300)	250	144	105	294	209	130	294	209	130	295	254	182	295	254	182	295	254	182	296 276 254
FNN5(600)	289	160	106	513	200	125	513	200	125	589	510	236	589	510	236	589	510	236	593 558 493
FNN6(1200)	472	247	181	782	339	195	782	339	195	1176	790	250	1176	790	250	1176	790	250	1189 1089 772
FNN7(1800)	469	271	177	770	350	200	775	350	200	1773	741	263	1773	741	263	1773	741	263	1h TIMEOUT
CNN1(12412)	12226	11788	11280	12371	12119	11786	12371	12122	11786	12373	12094	11659	12376	12193	11877	12376	12196	11877	8h TIMEOUT
CNN2(89572)	85793	77241	70212	OOM			OOM			89190	86910	81442	OOM			OOM			8h TIMEOUT

(c) The number of hidden ReLU neurons whose behavior can be decided with the bounds our abstract interpretation technique and Planet provide, with optional robustness bound $\delta \in \{0.1, 0.4, 0.6\}$ and the fixed input picture 767.

Table 1. Experimental results of abstract interpretation for MNIST DNNs with different approaches

7.2. Symbolic propagation versus other abstract interpretation based methods

We compare seven approaches: AI^2-r with Box, T-Zonotope and E-zonotope as underlying domains and Symb (i.e., our enhanced abstract interpretation with symbolic propagation) with Box, T-Zonotope and E-zonotope as underlying domains, and Planet [Ehl17], which serves as the benchmark verification approach (for its ability to compute bounds). All the experiments are conducted on an openSUSE Leap 15.0 machine with Intel i7-4790 CPU@3.60GHz and 16GB memory.

On MNIST, we train seven FNNs and two CNNs. The seven FNNs are of sizes 3×20, 6×20, 3×50, 3×100, 6×100, 6×200, and 9×200, where $m \times n$ refers to m hidden layers with n neurons in each hidden layer. CNN1 consists of 2 convolutional, 1 max-pooling, 2 convolutional, 1 max-pooling, and 3 fully connected layers in sequence, for a total of 12,412 neurons. CNN2 has 4 convolutional and 3 fully connected layers (89572 neurons). On ACAS Xu, we use the same networks as those in [KBD$^+$17b].

We consider the local robustness property with respect to the input region defined as follows:

$$X_{\bar{x},\delta} = \{\bar{x}' \in \mathbb{R}^m \mid \forall i. 1 - \delta \leq x_i \leq x_i' \leq 1 \vee x_i = x_i'\}.$$

In the experiments, the optional robustness bounds are 0.1, 0.2, 0.3, 0.4, 0.5, and 0.6.

Improvement on Bounds To see the improvement on bounds, we compare the output ranges of the above seven approaches on different inputs $\bar{x}$ and different tolerances δ. Table 1 (a) reports the results on three inputs $\bar{x}$ (No.767, No.1955 and No.2090 in the MNIST training dataset) and six tolerances $\delta \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6\}$. In all our

δ	AI^2-r		Symb			Planet
	TZono	EZono	Box	TZono	EZono	
0.1	7.13046%	7.08137%	6.15622%	6.15622%	6.15622%	5.84974%
0.2	11.09230%	10.88775%	6.92011%	6.92011%	6.92011%	6.11095%
0.3	18.75853%	18.32059%	8.21241%	8.21241%	8.21241%	6.50692%
0.4	30.11872%	29.27580%	10.31225%	10.31225%	10.31225%	7.04413%
0.5	45.13963%	44.25026%	14.49276%	14.49276%	14.49276%	7.96402%
0.6	55.67772%	54.88288%	20.03251%	20.03251%	20.03251%	9.02688%

Table 2. Bound proportions (smaller is better) for 1000 randomly sampled pictures from MNIST testing set on FNN1 with $\delta \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6\}$.

experiments, we set TIMEOUT as one hour for each FNN and eight hours for each CNN for a single run with an input, and a tolerance δ. In the table, TZono and EZono are abbreviations for T-Zonotope and E-Zonotope.

For each running we get a gap with an upper and lower bound for each neuron. Here we define the *bound proportion* to statistically describe how precise a range an approach gives. Basically given an approach (like Symb with Box domain), the bound proportion of this approach is the average of the ratio of the gap length of the neurons on the output layer and that obtained using AI^2-r with Box. Naturally AI^2-r with Box always has the bound proportion 1, and the smaller the bound proportion is, the more precise the ranges the approach gives are.

In Table 1 (a), every entry is the average bound proportion over three different inputs and six different tolerances. OOM stands for out-of-memory, 1h TIMEOUT for the one-hour timeout, and 8h TIMEOUT for the eight-hour timeout. We can see that, in general, Symb with Box, T-Zonotope and E-zonotope can achieve much better bounds than AI^2-r with Box, T-Zonotope and E-zonotope do. These bounds are closer to what Planet gives, except for FNN5 and FNN6. E-zonotope is slightly more precise than T-Zonotope. On the other hand, while Symb can return in a reasonable time in most cases, Planet cannot terminate in one hour (resp. eight hours) for FNN7 (resp. CNN1 and CNN2), which have $1,800$, $12,412$ and $89,572$ hidden neurons, respectively. Also we can see that results related to Thm. 5.1 are illustrated here. More specifically, (1) Symb with Box domain is more precise than AI^2-r with T-Zonotope and E-Zonotope on FNNs; (2) Symb with Box, T-Zonotope and E-Zonotope have the same precision on FNNs; (3) Symb with T-Zonotope and E-Zonotope are more precise than Symb with Box on CNNs.

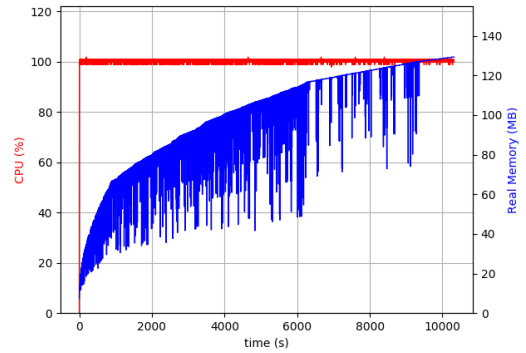
According to the memory footprint, both AI^2-r and Symb with T-Zonotope or E-Zonotope need more memory than the same approaches with Box do, and will crash on large networks, such as CNN2, because they run out of memory. Figure 8 shows how CPU and resident memory usage change over time. The horizontal axis in the figure is the time, in seconds, the vertical axis corresponding to the red line is the CPU usage percentage, and the vertical axis corresponding to the blue line is the memory usage, in MB.

Greater Verifiable Robustness Bounds Table 1 (b) shows the results of using the obtained bounds to help verify the robustness property. We consider a few thresholds for robustness tolerance, i.e., $\{0.1, 0.2, 0.3, 0.4, 0.5, 0.6\}$, and find that Symb can verify many more cases than AI^2-r can with comparable time consumption (less than 2x in most cases, and sometimes even faster).

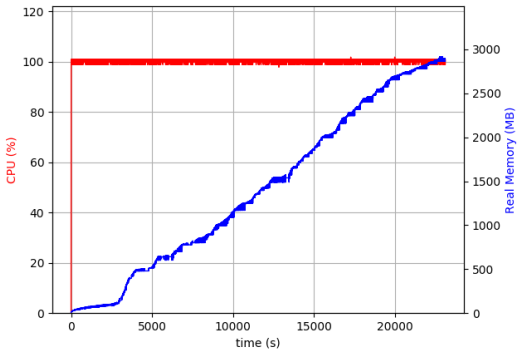
Proportion of Activated/Deactivated ReLU Nodes Table 1 (c) reports the number of hidden neurons whose ReLU behaviour (i.e., activated or deactivated) has been consistent within the tolerance δ. Compared to AI^2-r, our Symb can decide the ReLU behaviour with a much higher percentage.

We remark that, although the experimental results presented above are based on 3 fixed inputs, more extensive experiments have already been conducted to confirm that the conclusions are general. We randomly sample 1000 pictures (100 pictures per label) from the MNIST dataset, and compute the bound proportion for each of the pair (m, δ) where m refers to the seven approaches in Table 1 and $\delta \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6\}$ on FNN1. Each entry corresponding to (m, δ) in Table 2 is the average of bound proportions of approach m over 1000 pictures and fixed tolerance δ. Then we get the average of the bound proportion of AI^2-r with TZono/EZono, Symb with Box/TZono/EZono, and Planet over six different tolerances, 27.98623%, 27.44977%, 11.02104%, 11.02104%, 11.02104%, 7.08377%, respectively, which are very close to the first row of Table 1 (a).

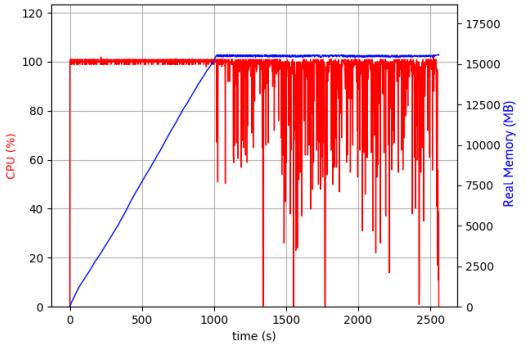
Comparison with the bounded powerset domain In AI^2 [GMDC$^+$18], the bounded powerset domains are used to improve the precision. In AI^2-r, we also implemented such bounded powerset domains instantiated by Box, T-



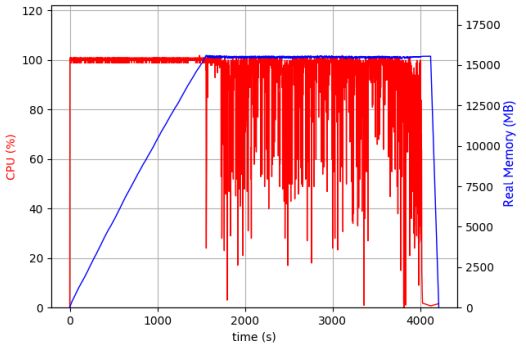
(a) Box



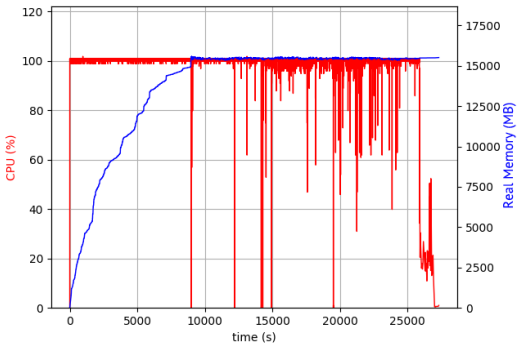
(b) SymBox



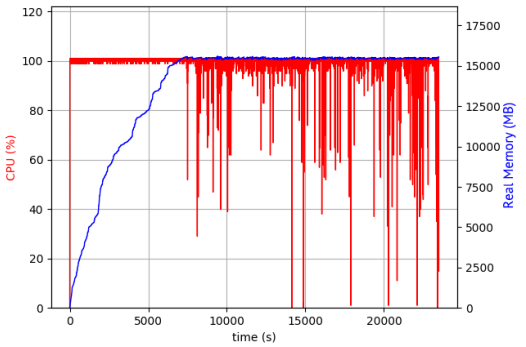
(c) TZono



(d) SymTZono



(e) EZono



(f) SymEZono

Fig. 8. CPU and memory usage

Zonotope and E-Zonotope domains, with 32 as the bound of the number of abstract elements in a disjunction. The comparison of the performance on the powerset domains with our symbolic propagation technique (with the underlying domains rather than powerset domains) is shown in Table 3. We can see that our technique is much more precise than the powerset domains. The time and memory consumptions of the powerset domains are both around 32 times greater than for the underlying domains, which are more than those of our technique.

Faster Verification In this part we use the networks of ACAS Xu. In order to evaluate the benefits of tighter bounds for SMT-based tools, we give the bounds obtained by abstract interpretation (on Box domain with symbolic propagation)

	AI^2-r			Symb			Planet
	Box32	TZono32	EZono32	Box	TZono	EZono	
FNN1	89.65790%	20.68675%	15.87726%	9.69327%	9.69327%	9.69327%	7.05553%
FNN2	89.42070%	16.27651%	8.18317%	1.76704%	1.76704%	1.76704%	0.89089%
FNN3	89.43396%	21.98109%	12.42840%	6.88656%	6.88656%	6.88656%	4.51223%
FNN4	89.44806%	25.97855%	13.05969%	5.13645%	5.13645%	5.13645%	2.71537%
FNN5	89.16034%	29.61022%	17.88676%	3.34578%	3.34578%	3.34578%	0.14836%
FNN6	89.30790%	OOM	22.60030%	7.12480%	7.12480%	7.12480%	1.94230%
FNN7	88.62267%	OOM	1h TIMEOUT	5.52267%	5.52267%	5.52267%	1h TIMEOUT

Table 3. Bound proportions (smaller is better) of different abstract interpretation approaches with the robustness bound $\delta \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6\}$, and the fixed input pictures 767, 1955, and 2090. Note that each entry gives the average bound proportion over six different tolerance and three pictures.

		$\delta = 0.1$		$\delta = 0.075$		$\delta = 0.05$		$\delta = 0.025$		$\delta = 0.01$		Total
		Result	Time	Result	Time	Result	Time	Result	Time	Result	Time	Time
Point 1	Reluplex	SAT	39	SAT	123	SAT	14	UNSAT	638	UNSAT	64	879
	Reluplex + ABS	SAT	45	SAT	36	SAT	14	UNSAT	237	UNSAT	36	368
Point 2	Reluplex	UNSAT	6513	UNSAT	1559	UNSAT	319	UNSAT	49	UNSAT	11	8451
	Reluplex + ABS	UNSAT	141	UNSAT	156	UNSAT	75	UNSAT	40	UNSAT	0	412
Point 3	Reluplex	UNSAT	1013	UNSAT	422	UNSAT	95	UNSAT	79	UNSAT	6	1615
	Reluplex + ABS	UNSAT	44	UNSAT	71	UNSAT	0	UNSAT	0	UNSAT	0	115
Point 4	Reluplex	SAT	3	SAT	5	SAT	1236	UNSAT	579	UNSAT	8	1831
	Reluplex + ABS	SAT	3	SAT	7	UNSAT	442	UNSAT	31	UNSAT	0	483
Point 5	Reluplex	UNSAT	14301	UNSAT	4248	UNSAT	1392	UNSAT	269	UNSAT	6	20216
	Reluplex + ABS	UNSAT	2002	UNSAT	1402	UNSAT	231	UNSAT	63	UNSAT	0	3698

Table 4. The satisfiability on given δ, and the time (in second) with and without bounds generated by abstract interpretation with symbolic propagation on the Box domain.

to Reluplex [KBD+17b] and observe the performance difference. The results are shown in Table 4. Each cell shows the satisfiability (i.e., SAT if an adversarial example is found) and the running time without or with given bounds. The experiments are conducted on different δ values (as in [KBD+17b]) and a fixed network (nnet1_1 [KBD+17b]) and 5 fixed points (Point 1 to 5 in [KBD+17b]). The time our technique spends on deriving the bounds is all less than 1 second. Table 4 shows that tighter initial bounds bring significant benefits to Reluplex with an overall $(\frac{1}{5076} - \frac{1}{32992})/\frac{1}{32992} = 549.43\%$ speedup (9.16 hours compared to 1.41 hours). However, it should be noted that, on one specific case (i.e., $\delta = 0.1$ at Point 1 and $\delta = 0.075$ at point 4), the tighter initial bounds slow Reluplex, which means that the speedup is not guaranteed on all cases. [4]

7.3. Lipschitz constant based method

In order to demonstrate the performance of the Lipschitz constant based method, we trained more MNIST networks including three FNNs with the sigmoid activation function (sigmoid-FNN1∼3), three FNNs with the $\tanh$ activation function (tanh-FNN1∼3), and two FNNs with the ReLU activation function (FNN8 and FNN9), of sizes 1×20, 1×50, $1\times30+1\times20$, 1×20, 1×50, $1\times30+1\times20$, 7×1000, and 14×500, respectively, where $m\times n$ refers to m hidden layers with n neurons in each hidden layer as mentioned before.

The experimental results are shown in Table 5, in which (a) and (b) show the Lipschitz constants computed by SDP and (c) shows the comparisons between Lip. methods and Symb. approach for computing the maximum verifiable radius.

Effect of regionalization via slope bounds. In Table 5 (a), we show the global Lipschitz constants, the regional Lipschitz constants for three randomly chosen inputs, and the computation times. The global Lipschitz constants are computed via the global slope bounds ($[0, 1]$ for $\tanh$ and $[0, 0.25]$ for sigmoid). The regional Lipschitz constants are computed via the local slope bounds in the region of L_p-norm ball with the radius 0.001. By applying local bounds,

[4] For the case $\delta = 0.05$ at point 4, Reluplex gives SAT and Reluplex+ABS gives UNSAT. This may be the result of a floating point arithmetic error.

	Global		Local 1		Local 2		Local 3	
	L. const	Time	L. const	Time	L. const	Time	L. const	Time
tanh-FNN1	37.428	23.673	37.404	124.85	37.321	113.34	37.407	111.26
tanh-FNN2	39.723	44.717	39.611	258.42	39.704	234.2	39.716	245.32
tanh-FNN3	89.664	30.705	89.440	167.55	89.471	154.29	89.472	158.65
sigmoid-FNN1	26.039	22.013	26.029	105.93	26.032	107.3	25.809	103.01
sigmoid-FNN2	24.636	43.511	24.625	237.07	24.629	238.82	24.572	223.32
sigmoid-FNN3	63.695	30.716	63.364	152.48	63.573	155.08	63.578	157.57

(a) Comparison of precision and time (in seconds) of computing a global Lipschitz constant and regional Lipschitz constants on small networks.

	L. const	Time (s)	Split	Mode
FNN7 (9×200)	240.987	608.13	1	LipSDP-Neuron
FNN8 (7×1000)	4.60310E8	7.3674	1	LipSDP-Layer
FNN9 (14×500)	2.45117E13	6.185	1	LipSDP-Layer
FNN9 (14×500)	5.00261E12	261.57	2	LipSDP-Neuron
CNN1-Subnet1	41.020	39.13	1	LipSDP-Network
CNN1-Subnet2	11.259	7.0061	1	LipSDP-Network
CNN1-Subnet3	1.532	10.608	-	LipSDP-Neuron
CNN1 (12412)	707.545	56.74	-	Subnets

(b) SDP works on large networks.

	FNN1			FNN7		
	Input 1	Input 2	Input 3	Input 1	Input 2	Input 3
L. const		23.928			240.987	
SDP safety L_∞ ball	0.020	0.016	0.018	0.0024	0.0016	0.0024
Symb. safety L_∞ ball	0.037	0.018	0.031	0.007	0.004	0.008

(c) Comparison of precision of the SDP methods and Symb.

Table 5. Experimental results of SDP methods for MNIST DNNs.

the regional Lipschitz constants become tighter. Note that the effect depends on many factors, including the size of the DNN, the chosen input, and the size of the region. Although the effect is not significant, it may improve the precision of robustness verification. Furthermore, it brings a side effect on efficiency, i.e., increasing the time consumption. A reasonable explanation is that it makes the matrix involved less sparse when we assigns a positive number to α instead of 0.

SDP works efficiently on large networks. Table 5 (b) gives the experimental results of computing the Lipschitz constants by SDP on large DNNs. If we choose the SDP mode properly and make some necessary splits on the DNNs, then it is quite efficient to obtain a global Lipschitz constant. As we can see, the SDP methods are strong in its high efficiency, applicability to more activation functions, and the globalization of the Lipschitz constants.

Lip. is not as precise as Symb. Here, we compare the performance of Lip. methods with our Symb. approach for computing the maximum verifiable radius, and the experimental results are shown in Table 5 (c). Basically, once we obtain a Lipschitz constant $\mathcal{L}$ of a DNN f, then we can obtain a safe L_∞ ball. Also, through a simple binary search, we can use Symb. to get the largest robustness L_∞ ball. Table 5 (c) gives the results of the two methods working on FNN1 and FNN7. We can see that abstract interpretation based methods gives larger radius, so generally abstract interpretation is more precise than Lip. methods in DNN verificaiton. However, from the perspective of efficiency, since the Lipschitz constant can be reused to compute the maximum verifiable radius for different inputs and the procedure only invokes simple arithmetic operations in a non-iterative way. In other words, Lip. method can be used to compute a rougher robustness bounds, less precisely but more efficiently than the Symb. approach.

7.4. Verifying robustness on batch inputs using SDP

We also use the Lipschitz constant based method to verify local robustness of batch inputs. We randomly select 10,000 inputs of MNIST as the batch, and verify them with the framework of Sect. 6.3 on the networks tanh-FNN1$\sim$3 and sigmoid-FNN1$\sim$3. The results are shown in Fig. 9 and Table 6.

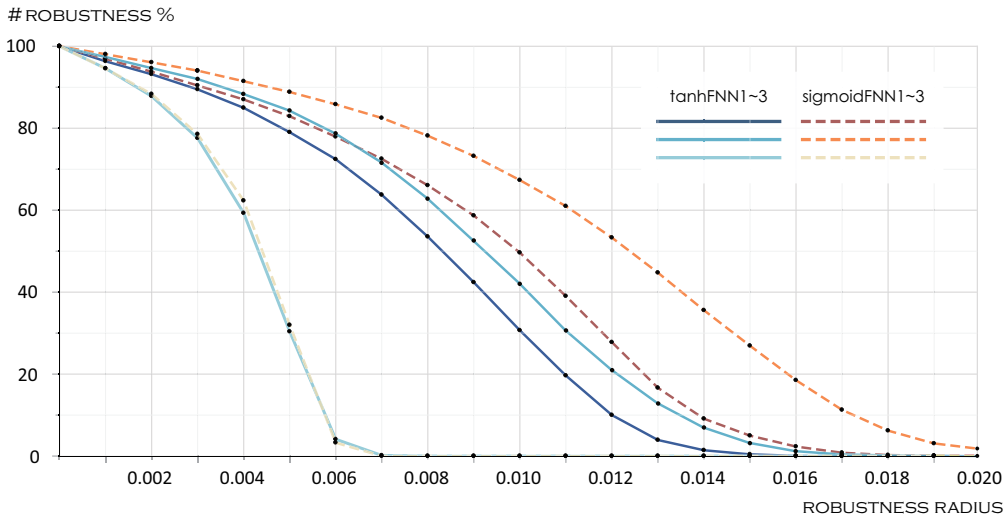


Fig. 9. Results of batch local robustness filter using SDP.

	Computing Lip.	Verifying	Total
tanh-FNN1	23.673	2.78	26.453
tanh-FNN2	44.717	14.48	59.197
tanh-FNN3	30.705	3.76	34.465
sigmoid-FNN1	22.013	2.86	24.873
sigmoid-FNN2	43.511	14.48	57.991
sigmoid-FNN3	30.716	3.61	34.326

Table 6. The time (in seconds) of computing the Lipschitz constant and verifying the L_∞ robustness for ten thousand inputs with twenty δ values from 0.01 to 0.2.

As can be seen from Fig. 9, the Lipschitz based method verifies a substantial number of cases in this experiment. Moreover, Table 6 shows that the time costs of batch local robustness verification are less than one minute in these situation and mainly lie on the computation of Lipschitz constants of the networks by using SDP. So as a filter to speed up the verification proceed, this method is very efficient and scalable.

8. Discussion and Related Work

In this section, we discuss the soundness guarantee of the abstraction based approach and the precision of the abstract domains, and then provide some closely related work.

8.1. Discussion on Soundness and Precision

Soundness is an essential property of formal verification. Abstract interpretation is known for its soundness guarantee for analysis and verification [Min17], since it conducts over-approximation to enclose all the possible behaviors of the original system. Computing over-approximations for a DNN is thus our soundness guarantee in this paper. As shown in Thm. 4.1, if the results of abstract interpretation show that the property C holds (i.e., $\gamma(X^\sharp{}_N) \subseteq C$ in Equation 2), then the property also holds for the set of actual executions of the DNN (i.e., $f(X_0) \subseteq C$). If the results of abstract interpretation can not prove that the property C holds, however, the verification is inconclusive. In this case, the results of the chosen abstract domain are not precise enough to prove the property, and thus more powerful abstract domains are needed. Moreover, our symbolic propagation also preserves soundness, since it uses symbolic substitution to compute the composition of linear transformations.

On the other hand, many existing DNN verification tools do not guarantee soundness. For example, Reluplex [KBD+17b] (using GLPK), Planet [Ehl17] (using GLPK), and Sherlock [DJST18] (using Gurobi) all rely on the floating point implementation of linear programming solvers, which is unsound. Actually, most state-of-the-art linear programming

solvers use floating-point arithmetic and only give approximate solutions which may not be the actual optimum solution or may even lie outside the feasible space [NS04]. It may happen that a linear programming solver implemented via floating point arithmetic wrongly claims that a feasible linear system is infeasible or the other way round. In fact, the paper [DJST18] reports several false positive results in Reluplex, and mentions that this comes from unsound floating point implementation.

Different abstract domains have different precision in verification. In our DNN verification settings, if the input region is a box, then the precision has the following order: Box $<$ T-Zonotope $<$ E-Zonotope $<$ DeepPoly $<$ Polyhedra. DeepPoly [SGPV19b] is a specialized abstract domain for DNN verification. An abstract element in DeepPoly is a tuple $a = (a^{\leq}, a^{\geq}, \bar{l}, \bar{u})$, 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 $\bar{l}, \bar{u} \in \mathbb{R}^n$ give the lower bound and upper bound of each variable, respectively. Box loses precision on both affine transformations and non-linear activation functions, while the others only lose precision on non-linear activation functions. With symbolic propagation, Box does not lose precision on affine transformations any more, and Zonotope has better precision on uncertain ReLU neurons. Octagon is not often used in DNN verification, since the weights in a DNN are generally not uniform and they do not fit for the constraints of Octagon.

8.2. Related Work

Verification of neural networks can be traced back to [PT10], where the network is encoded after approximating every sigmoid activation function with a set of piecewise linear constraints and then solved with an SMT solver. It works with a network of 6 hidden nodes. More recently, by considering DNNs with ReLU activation functions, the verification approaches include constraint-solving [KBD$^+$17b, LM18, Ehl17, NKR$^+$17], layer-by-layer exhaustive search [HKWW17], global optimisation [RHK18a, DJST18, RWS$^+$19], abstract interpretation [GMDC$^+$18, SGM$^+$18, SGPV19b], functional approximation [XTJ18], and reduction to two-player game [WHK18, WmWR$^+$19], etc.

More specifically, In [KBD$^+$17b] an SMT solver Reluplex is presented to verify properties on DNNs with fully-connected layers. Independently [Ehl17] presents another SMT solver Planet which combines linear approximation and interval arithmetic to work with fully connected and max pooling layers. Later, Reluplex has been extended in Marabou [KHI$^+$19] to support piece-wise linear activation functions and improves the efficiency. Also, an abstraction-refinement framework is proposed in [EGK20], which helps reduce the size of DNNs to verify. Methods based on SMT solvers do not scale well, e.g., Reluplex can only work with DNNs with a few hidden neurons.

The above target mainly the verification of local robustness. Research has been conducted to compute other properties, e.g., the output reachability. An exact computation of output reachability can be utilised to verify local robustness. In [DJST18], Sherlock, an algorithm based on local and global search and mixed integer linear programming (MILP), is put forward to calculate the output range of a given label when the inputs are restricted to a small subspace. [RHK18a] presents another algorithm for output range analysis, and their algorithm is suitable for all Lipschitz continuous DNNs, including all layers and activation functions mentioned before. In [WPW$^+$18], the authors use symbolic interval propagation to calculate output range. Compared with [WPW$^+$18], our approach is adequate for general abstract domains, while their symbolic interval propagation is designed specifically for symbolic intervals. Further, methods based on star sets have been developed in [TLM$^+$19, TBXJ20] to compute a more precise reachablity for DNNs. Abstraction based output range analysis is proposed in [PA19] to deal with larger models.

AI^2[GMDC$^+$18] is the first tool to use abstract interpretation to verify DNNs. They define a class of functions called conditional affine transformations (CAT) to characterize DNNs containing fully connected, convolutional and max pooling layers with the ReLU activation function. They use Interval and Zonotope as the abstract domains and the powerset technique on Zonotope. Compared with AI^2, we use symbolic propagation rather than powerset extension techniques to enhance the precision of abstract interpretation based DNN verification. Symbolic propagation is more lightweight than powerset extension. Moreover, we also use the bounds information given by abstract interpretation to accelerate SMT based DNN verification. DeepZ [SGM$^+$18] and DeepPoly [SGPV19b] propose two specific abstract domains tailored to DNN verification in order to improve the precision of abstract interpretation on the verification on DNNs, and [SGPV19a] improves DeepPoly in performing linear over-approximation over multiple uncertain ReLU nodes together instead of one by one. In contrast to these works, our work is a general approach that can be applied on various domains. Recently [APDC19] has proposed an approach which uses machine learning to recommend the choice of split and abstract interpretation in DNN verification. This combines the various abstract interpretation techniques in a clever and effective manner. Abstract interpretation based DNN verification can be performed in parallel on GPU and it improves the size of the DNNs that can be treated to one million neurons [MSPV20].

Besides verification methods, adversarial attack and robustness training are also closely related to our work. Adver-

sarial examples and adversarial attack were first put forward in [SZS+14], and some famous robustness attack methods include Fast Gradient Sign [MFF16], Jacobian-based saliency map approach [PMJ+16], C&W attack [CW17], etc. In recent years a great number of works like [KSDG20, CH20, XD20, RCBG20, HZ20, ZFY+19] have developed adversarial attack with a variety of methods and techniques. Robustness training for DNNs can be traced back to [NW15, ARA+16], and the latest works include [MGN+20, WMB+19, YGZ18, GDV20, RCJ19, ZG19]. These works improves the robustness of deep neural networks in the training step in different ways for various DNN models and applications.

9. Conclusion

In this paper, we have studied different a variety of local robustness properties and the (δ, ε)-global robustness property for deep neural networks, and the corresponding verification methods, based on abstract interpretation, SMT, as well as Lipschitz constants.

We explore the potential of abstract interpretation for the verification of DNNs. We have proposed to use symbolic propagation on abstract interpretation to take advantage of the linearity in most part of the DNNs, which achieved significant improvements in terms of the precision and memory usage. This is based on a key observation that, for local robustness verification of DNNs where a small region of the input space is concerned, a considerable percentage of hidden neurons remain active or inactive for all possible inputs in the region. For these neurons, their ReLU activation function can be replaced by a linear function. Our symbolic propagation iteratively computes for each neuron this information and utilize the computed information to improve the performance. This paper has presented with formal proofs three somewhat surprising theoretical results, which are then confirmed by our experiments. These results have enhanced our theoretical and practical understanding about the abstract interpretation based DNN verification and symbolic propagation.

We apply the tighter bounds of variables on hidden neurons from our approach to improve the performance of the state-of-the-art SMT based DNN verification tools, like Reluplex. The speed-up rate is up to 549% in our experiments. We believe this result sheds some light on the potential for improving the scalability of SMT-based DNN verification: In addition to improving the performance through enhancing the SMT solver for DNNs, an arguably easier way is to take an abstract interpretation technique (or other techniques that can refine the constraints) as a pre-processing.

Furthermore, we propose a Lipschitz constant based verification framework. It is more efficient and scalable to verify local or global robustness of DNNs with Lipschitz constants solved by semidefinite programming. We present a method to compute a maximum verifiable radius for verifying the local robustness properties related to L_p-norms. We also present a method to verify the (δ, ε)-global robustness of a DNN in a given region. The maximum verifiable radii obtained can be used as a filter in batch tasks to speed up the verification process of DeepSymbol, which meet the demand of working on large complex networks and more general robustness properties.

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, 61872445, 62002363), and Natural Science Foundation of Guangdong Province, China (Grant No. 2019A1515011689).

Declarations: Conflicts of interest / Competing interests (Not applicable)

We claim no conflicts of interests.

Declarations: Availability of data and material (Not applicable)

We use MNIST and ACAS Xu as datasets. See Section 7.1 for details. All the networks we trained on MNIST are contained in `https://github.com/CAS-LRJ/LipSDP/tree/master/LipSDP/examples`.

Declarations: Code availability (Not applicable)

PRODeep is open source in `https://iscasmc.ios.ac.cn/prodeep`. For other abstract domains and the SDP based Lipschitz constant calculation, see Section 7.1 for details. All the experimental materials on Lipschitz constant based verification are included in `https://github.com/CAS-LRJ/LipSDP/tree/master/LipSDP/examples`.

References

[APDC19] Greg Anderson, Shankara Pailoor, Isil Dillig, and Swarat Chaudhuri. Optimization and abstraction: a synergistic approach for analyzing neural network robustness. In Kathryn S. McKinley and Kathleen Fisher, editors, *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2019, Phoenix, AZ, USA, June 22-26, 2019*, pages 731–744. ACM, 2019.

[ARA+16] Sheikh Waqas Akhtar, Saad Rehman, Mahmood Akhtar, Muazzam Ali Khan, Farhan Riaz, Qaiser Chaudry, and Rupert C. D. Young. Improving the robustness of neural networks using k-support norm based adversarial training. *IEEE Access*, 4:9501–9511, 2016.

[CC77] Patrick Cousot and Radhia Cousot. 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)*, pages 238–252, 1977.

[CH20] Francesco Croce and Matthias Hein. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. *CoRR*, abs/2003.01690, 2020.

[CW17] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *Security and Privacy (SP), 2017 IEEE Symposium on*, pages 39–57. IEEE, 2017.

[DJST18] Souradeep Dutta, Susmit Jha, Sriram Sankaranarayanan, and Ashish Tiwari. Output range analysis for deep feedforward neural networks. In *NASA Formal Methods - 10th International Symposium, NFM 2018, Newport News, VA, USA, April 17-19, 2018, Proceedings*, pages 121–138, 2018.

[DSG+18] Krishnamurthy Dvijotham, Robert Stanforth, Sven Gowal, Timothy A. Mann, and Pushmeet Kohli. A dual approach to scalable verification of deep networks. *CoRR*, abs/1803.06567, 2018.

[EGK20] Yizhak Yisrael Elboher, Justin Gottschlich, and Guy Katz. An abstraction-based framework for neural network verification. In Shuvendu K. Lahiri and Chao Wang, editors, *Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part I*, volume 12224 of *Lecture Notes in Computer Science*, pages 43–65. Springer, 2020.

[Ehl17] Rüdiger Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In *15th International Symposium on Automated Technology for Verification and Analysis (ATVA2017)*, pages 269–286, 2017.

[FRH+19] Mahyar Fazlyab, Alexander Robey, Hamed Hassani, Manfred Morari, and George J. Pappas. Efficient and accurate estimation of lipschitz constants for deep neural networks. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pages 11423–11434, 2019.

[GDV20] Sidharth Gupta, Parijat Dube, and Ashish Verma. Improving the affordability of robustness training for dnns. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR Workshops 2020, Seattle, WA, USA, June 14-19, 2020*, pages 3383–3392. IEEE, 2020.

[GGP09] Khalil Ghorbal, Eric Goubault, and Sylvie Putot. The zonotope abstract domain taylor1+. In *International Conference on Computer Aided Verification*, pages 627–633. Springer, 2009.

[GGP10] Khalil Ghorbal, Eric Goubault, and Sylvie Putot. A logical product approach to zonotope intersection. In Tayssir Touili, Byron Cook, and Paul B. Jackson, editors, *Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings*, volume 6174 of *Lecture Notes in Computer Science*, pages 212–226. Springer, 2010.

[GMDC+18] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev. AI^2: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy (S&P 2018)*, pages 948–963, 2018.

[HDY+12] Geoffrey Hinton, Li Deng, Dong Yu, George E. Dahl, Abdel rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N. Sainath, and Brian Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Process. Mag.*, 29(6):82–97, 2012.

[HKWW17] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks. In *29th International Conference on Computer Aided Verification (CAV2017)*, pages 3–29, 2017.

[HZ20] Zhichao Huang and Tong Zhang. Black-box adversarial attack with transferable model-based embedding. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[JGK+15] Jean-Baptiste Jeannin, Khalil Ghorbal, Yanni Kouskoulas, Ryan Gardner, Aurora Schmidt, Erik Zawadzki, and André Platzer. Formal verification of ACAS x, an industrial airborne collision avoidance system. In *2015 International Conference on Embedded Software, EMSOFT 2015, Amsterdam, Netherlands, October 4-9, 2015*, pages 127–136, 2015.

[JKO18] Kyle D. Julian, Mykel J. Kochenderfer, and Michael P. Owen. Deep neural network compression for aircraft collision avoidance systems. *CoRR*, abs/1810.04240, 2018.

[KBD+17a] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Towards proving the adversarial robustness of deep neural networks. *arXiv*, 2017.

[KBD+17b] Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In *29th International Conference on Computer Aided Verification (CAV2017)*, pages 97–117, 2017.

[KHI+19] Guy Katz, Derek A. Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljic, David L. Dill, Mykel J. Kochenderfer, and Clark W. Barrett. The marabou framework for verification and analysis of deep neural networks. In Isil Dillig and Serdar Tasiran, editors, *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I*, volume 11561 of *Lecture Notes in Computer Science*, pages 443–452. Springer, 2019.

[KSDG20] Xu Kang, Bin Song, Xiaojiang Du, and Mohsen Guizani. Adversarial attacks for image segmentation on multiple lightweight models. *IEEE Access*, 8:31359–31370, 2020.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.*, pages 1106–1114, 2012.

[LBBH98] Yann Lécun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[LLcH+20] Renjue Li, Jianlin Li, Cheng chao Huang, Pengfei Yang, Xiaowei Huang, Lijun Zhang, Bai Xue, and Holger Hermanns. Prodeep: a platform for robustness verification of deep neural networks. In *ESEC/FSE 2020*, 2020.

[LLY+19] Jianlin Li, Jiangchao Liu, Pengfei Yang, Liqian Chen, Xiaowei Huang, and Lijun Zhang. Analyzing deep neural networks with symbolic propagation: Towards higher precision and faster verification. In Bor-Yuh Evan Chang, editor, *Static Analysis - 26th International Symposium, SAS 2019, Porto, Portugal, October 8-11, 2019, Proceedings*, volume 11822 of *Lecture Notes in Computer Science*, pages 296–319. Springer, 2019.

[LM18] Alessio Lomuscio and Lalit Maganti. An approach to reachability analysis for feed-forward ReLU neural networks. In *KR2018*, 2018.

[MFF16] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: A simple and accurate method to fool deep neural networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 2574–2582. IEEE Computer Society, 2016.

[MGN+20] Chengzhi Mao, Amogh Gupta, Vikram Nitin, Baishakhi Ray, Shuran Song, Junfeng Yang, and Carl Vondrick. Multitask learning strengthens adversarial robustness. *CoRR*, abs/2007.07236, 2020.

[Min17] Antoine Miné. Tutorial on static inference of numeric invariants by abstract interpretation. *Foundations and Trends in Programming Languages*, 4(3-4):120–372, 2017.

[MMS+18] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[MSPV20] Christoph Müller, Gagandeep Singh, Markus Püschel, and Martin T. Vechev. Neural network robustness verification on gpus. *CoRR*, abs/2007.10868, 2020.

[NKR+17] Nina Narodytska, Shiva Prasad Kasiviswanathan, Leonid Ryzhyk, Mooly Sagiv, and Toby Walsh. Verifying properties of binarized deep neural networks. *arXiv preprint arXiv:1709.06662*, 2017.

[NS04] A. Neumaier and O. Shcherbina. Safe bounds in linear and mixed-integer linear programming. *Math. Program.*, 99(2):283–296, 2004.

[NW15] Arun Narayanan and DeLiang Wang. Improving robustness of deep neural network acoustic models via speech separation and joint adaptive training. *IEEE ACM Trans. Audio Speech Lang. Process.*, 23(1):92–101, 2015.

[NWL19] JAY NANDY, HSU WYNNE, and LEE MONG LI. Robustness for adversarial $l_{p\geq 1}$ perturbations. In *NeurIPS 2019 Workshop on Machine Learning with Guarantees, Vancouver, Canada*, 2019.

[NYC15] Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 427–436, 2015.

[PA19] Pavithra Prabhakar and Zahra Rahimi Afzal. Abstraction based output range analysis for neural networks. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pages 15762–15772, 2019.

[PMJ+15] Nicolas Papernot, Patrick D. McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. *CoRR*, abs/1511.07528, 2015.

[PMJ+16] Nicolas Papernot, Patrick D. McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *IEEE European Symposium on Security and Privacy, EuroS&P 2016, Saarbrücken, Germany, March 21-24, 2016*, pages 372–387. IEEE, 2016.

[PT10] Luca Pulina and Armando Tacchella. 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*, pages 243–257, 2010.

[RCBG20] Binxin Ru, Adam D. Cobb, Arno Blaas, and Yarin Gal. Bayesopt adversarial attack. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[RCJ19] Eitan Rothberg, Tingting Chen, and Hao Ji. Towards better accuracy and robustness with localized adversarial training. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 10017–10018. AAAI Press, 2019.

[RHK18a] Wenjie Ruan, Xiaowei Huang, and Marta Kwiatkowska. Reachability analysis of deep neural networks with provable guarantees. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden.*, pages 2651–2659, 2018.

[RHK18b] Wenjie Ruan, Xiaowei Huang, and Marta Kwiatkowska. Reachability analysis of deep neural networks with provable guarantees. In *IJCAI2018*, pages 2651–2659, 2018.

[RWS+19] Wenjie Ruan, Min Wu, Youcheng Sun, Xiaowei Huang, Daniel Kroening, and Marta Kwiatkowska. Global robustness evaluation of deep neural networks with provable guarantees for the hamming distance. In *IJCAI2019*, 2019.

[SGM+18] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin T. Vechev. 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.*, pages 10825–10836, 2018.

[SGPV19a] Gagandeep Singh, Rupanshu Ganvir, Markus Püschel, and Martin T. Vechev. Beyond the single neuron convex barrier for neural network certification. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman

Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pages 15072–15083, 2019.

[SGPV19b] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin T. Vechev. An abstract domain for certifying neural networks. *PACMPL*, 3(POPL):41:1–41:30, 2019.

[SHM+16] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[SRBB19] Lukas Schott, Jonas Rauber, Matthias Bethge, and Wieland Brendel. Towards the first adversarially robust neural network model on MNIST. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[SZS+14] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations (ICLR2014)*, 2014.

[TB19] Florian Tramèr and Dan Boneh. Adversarial training and robustness for multiple perturbations. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pages 5858–5868, 2019.

[TBXJ20] Hoang-Dung Tran, Stanley Bak, Weiming Xiang, and Taylor T. Johnson. Verification of deep convolutional neural networks using imagestars. In Shuvendu K. Lahiri and Chao Wang, editors, *Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part I*, volume 12224 of *Lecture Notes in Computer Science*, pages 18–42. Springer, 2020.

[TLM+19] Hoang-Dung Tran, Diego Manzanas Lopez, Patrick Musau, Xiaodong Yang, Luan Viet Nguyen, Weiming Xiang, and Taylor T. Johnson. Star-based reachability analysis of deep neural networks. In Maurice H. ter Beek, Annabelle McIver, and José N. Oliveira, editors, *Formal Methods - The Next 30 Years - Third World Congress, FM 2019, Porto, Portugal, October 7-11, 2019, Proceedings*, volume 11800 of *Lecture Notes in Computer Science*, pages 670–686. Springer, 2019.

[vEG14] Christian von Essen and Dimitra Giannakopoulou. Analyzing the next generation airborne collision avoidance system. In *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*, pages 620–635, 2014.

[WHK18] Matthew Wicker, Xiaowei Huang, and Marta Kwiatkowska. Feature-guided black-box safety testing of deep neural networks. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS2018)*, pages 408–426. Springer, 2018.

[WK18] Eric Wong and J. Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 5283–5292, 2018.

[WMB+19] Yisen Wang, Xingjun Ma, James Bailey, Jinfeng Yi, Bowen Zhou, and Quanquan Gu. On the convergence and robustness of adversarial training. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 6586–6595. PMLR, 2019.

[WmWR+19] Min Wu, matthew Wicker, Wenjie Ruan, Xiaowei Huang, and Marta Kwiatkowska. A game-based approximate verification of deep neural networks with provable guarantees. *Theoretical Computer Science*, 5 2019.

[WPW+18] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Formal security analysis of neural networks using symbolic intervals. *CoRR*, abs/1804.10829, 2018.

[WZC+18] Tsui-Wei Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Luca Daniel, Duane S. Boning, and Inderjit S. Dhillon. Towards fast computation of certified robustness for relu networks. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 5273–5282, 2018.

[XD20] Jincheng Xu and Qingfeng Du. Texttricker: Loss-based and gradient-based adversarial attacks on text classification models. *Eng. Appl. Artif. Intell.*, 92:103641, 2020.

[XTJ18] W. Xiang, H. Tran, and T. T. Johnson. Output reachable set estimation and verification for multilayer neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 29(11):5777–5783, Nov 2018.

[YGZ18] Ziang Yan, Yiwen Guo, and Changshui Zhang. Deep defense: Training dnns with improved adversarial robustness. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*, pages 417–426, 2018.

[ZFY+19] Chenxiao Zhao, P. Thomas Fletcher, Mixue Yu, Yaxin Peng, Guixu Zhang, and Chaomin Shen. The adversarial attack and detection under the fisher information metric. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 5869–5876. AAAI Press, 2019.

[ZG19] Daniel Zügner and Stephan Günnemann. Certifiable robustness and robust training for graph convolutional networks. In Ankur Teredesai, Vipin Kumar, Ying Li, Rómer Rosales, Evimaria Terzi, and George Karypis, editors, *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*, pages 246–256. ACM, 2019.