An Abstract Domain for Certifying Neural Networks

GAGANDEEP SINGH, ETH Zurich, Switzerland
TIMON GEHR, ETH Zurich, Switzerland
MARKUS PÜSCHEL, ETH Zurich, Switzerland
MARTIN VECHEV, ETH Zurich, Switzerland

We present a novel method for scalable and precise certification of deep neural networks. The key technical insight behind our approach is a new abstract domain which combines floating point polyhedra with intervals and is equipped with abstract transformers specifically tailored to the setting of neural networks. Concretely, we introduce new transformers for affine transforms, the rectified linear unit (ReLU), sigmoid, tanh, and maxpool functions.

We implemented our method in a system called DeepPoly and evaluated it extensively on a range of datasets, neural architectures (including defended networks), and specifications. Our experimental results indicate that DeepPoly is more precise than prior work while scaling to large networks.

We also show how to combine DeepPoly with a form of abstraction refinement based on trace partitioning. This enables us to prove, for the first time, the robustness of the network when the input image is subjected to complex perturbations such as rotations that employ linear interpolation.

CCS Concepts: • **Theory of computation** → **Program verification**; **Abstraction**; • **Computing methodologies** → **Neural networks**;

Additional Key Words and Phrases: Abstract Interpretation, Deep Learning, Adversarial attacks

1 INTRODUCTION

Over the last few years, deep neural networks have become increasingly popular and have now started penetrating safety critical domains such as autonomous driving [Bojarski et al. 2016] and medical diagnosis [Amato et al. 2013] where they are often relied upon for making important decisions. As a result of this widespread adoption, it has become even more important to ensure that neural networks behave reliably and as expected. Unfortunately, reasoning about these systems is challenging due to their "black box" nature: it is difficult to understand what the network does since it is typically parameterized with thousands or millions of real-valued weights that are hard to interpret. Further, it has been discovered that neural nets can sometimes be surprisingly brittle and exhibit non-robust behaviors, for instance, by classifying two very similar inputs (e.g., images that differ only in brightness or in one pixel) to different labels [Goodfellow et al. 2015].

Authors' addresses: Gagandeep Singh, Department of Computer Science, ETH Zurich, Zürich, Switzerland, gsingh@inf.ethz.ch; Timon Gehr, Department of Computer Science, ETH Zurich, Zürich, Switzerland, timon.gehr@inf.ethz.ch; Markus Püschel, Department of Computer Science, ETH Zurich, Zürich, Switzerland, pueschel@inf.ethz.ch; Martin Vechev, Department of Computer Science, ETH Zurich, Zürich, Switzerland, martin.vechev@inf.ethz.ch.

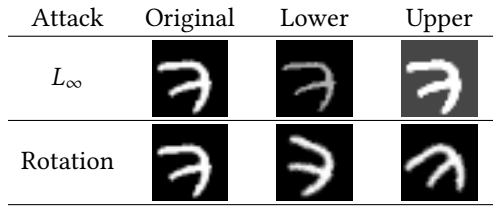| Attack	Original	Lower	Upper
L_∞			
Rotation			

Fig. 1. Two different attacks applied to MNIST images.

To address the challenge of reasoning about neural networks, recent research has started exploring new methods and systems which can *automatically prove* that a given network satisfies a specific property of interest (e.g., robustness to certain perturbations, pre/post conditions). State-of-the-art works include methods based on SMT solving [Katz et al. 2017], linear approximations [Weng et al. 2018], and abstract interpretation [Gehr et al. 2018; Mirman et al. 2018; Singh et al. 2018a].

Despite the progress made by these works, more research is needed to reach the point where we are able to solve the overall neural network reasoning challenge successfully. In particular, we still lack an analyzer that can scale to large networks, is able to handle popular neural architectures (e.g., feedforward, convolutional), and yet is sufficiently precise to prove relevant properties required by applications. For example, the work by Katz et al. [2017] is precise yet can only handle very small networks. At the same time, Gehr et al. [2018] can analyze larger networks than Katz et al. [2017], but relies on existing generic abstract domains which either do not scale to larger neural networks (such as Convex Polyhedra [Cousot and Halbwachs 1978]) or are too imprecise (e.g., Zonotope [Ghorbal et al. 2009]). Recent work by Weng et al. [2018] scales better than Gehr et al. [2018] but only handles feedforward networks and cannot handle the widely used convolutional networks. Both Katz et al. [2017] and Weng et al. [2018] are in fact unsound for floating point arithmetic, which is heavily used in neural nets, and thus they can suffer from false negatives. Recent work by Singh et al. [2018a] handles feedforward and convolutional networks and is sound for floating point arithmetic, however, as we demonstrate experimentally, it can lose significant precision when dealing with larger perturbations.

This work. In this work, we propose a new method and system, called *DeepPoly*, that makes a step forward in addressing the challenge of verifying neural networks with respect to both scalability and precision. The key technical idea behind *DeepPoly* is a novel abstract interpreter specifically tailored to the setting of neural networks. Concretely, our abstract domain is a combination of floating-point polyhedra with intervals, coupled with abstract transformers for common neural network functions such as affine transforms, the rectified linear unit (ReLU), sigmoid and tanh activations, and the maxpool operator. These abstract transformers are carefully designed to exploit key properties of these functions and balance analysis scalability and precision. As a result, *DeepPoly* is more precise than Weng et al. [2018], Gehr et al. [2018] and Singh et al. [2018a], yet can handle large convolutional networks and is sound for floating point arithmetic.

Proving robustness: illustrative examples. To provide an intuition for the kind of problems that *DeepPoly* can solve, consider the images shown in Fig. 1. Here, we will illustrate two kinds of robustness properties: L_∞-norm based perturbations (first row) and image rotations (second row).

In the first row, we are given an image of the digit 7 (under "Original"). Then, we consider an attack where we allow a small perturbation to *every pixel* in the original image (visually this may correspond to darkening or lightening the image). That is, instead of a number, each pixel now contains an interval. If each of these intervals has the same size, we say that we have formed an

L_∞ ball around the image (typically with a given epsilon ϵ). This ball is captured visually by the Lower image (in which each pixel contains the smallest value allowed by its interval) and the Upper image (in which each pixel contains the largest value allowed by its interval). We call the modification of the original image to a perturbed version inside this ball an attack, reflecting an adversary who aims to trick the network. There have been various works which aim to find such an attack, otherwise called an adversarial example (e.g., Carlini and Wagner [2017]), typically using gradient-based methods. For our setting however, the question is: are all possible images "sitting between" the Lower and the Upper image classified to the same label as the original? Or, in other words, is the neural net robust to this kind of attack?

The set of possible images induced by the attack is also called an *adversarial region*. Note that enumerating all possible images in this region and simply running the network on each to check if it is classified correctly, is practically infeasible. For example, an image from the standard MNIST [Lecun et al. 1998] dataset contains 784 pixels and a perturbation that allows for even two values for every pixel will lead to 2^{784} images that one would need to consider. In contrast, our system *DeepPoly* can *automatically prove* that all images in the adversarial region classify correctly (that is, no attack is possible) by soundly propagating the entire input adversarial region through the abstract transformers of the network.

We also consider a more complex type of perturbation in the second row. Here, we rotate the image by an angle and our goal is to show that any rotation up to this angle classifies to the same label. In fact, we consider an even more challenging problem where we not only rotate an image but first form an adversarial region around the image and then reason about all possible rotations of any image in that region. This is challenging, as again, enumeration of images is infeasible when using geometric transformations that perform linear interpolation (which is needed to improve output image quality). Further, because rotation is a transform, the entire set of possible images represented by a rotation up to a given angle needs to somehow be captured. Directly approximating this set is too imprecise and the analysis fails to prove the wanted property. Thus, we introduce a method where we *refine* the initial approximation into smaller regions that correspond to smaller angles (a form of trace partitioning [Rival and Mauborgne 2007]), use *DeepPoly* to prove the property on each smaller region, and then deduce the property holds for the initial, larger approximation. To our best knowledge this is the first work which shows how to prove robustness of a neural network under complex input perturbations such as rotations.

Main contributions. Our main contributions are:

- A new abstract domain for the certification of neural nets. The domain combines floating point polyhedra and intervals with custom abstract transformers for affine transforms, ReLU, sigmoid, tanh, and maxpool functions. These abstract transformers carefully balance scalability and precision of the analysis (Section 4).
- An approach for proving more complex perturbation specifications than considered so far, including rotations using linear interpolation, based on refinement of the abstract input. To our best knowledge, this is the first time such perturbations have been verified (Section 5).
- A complete, parallelized implementation of our approach in a system called *DeepPoly*, which can handle both feedforward and convolutional neural networks (Section 6). Our entire system is fully available at http://safeai.ethz.ch.
- An extensive evaluation on a range of datasets and networks including defended ones, showing *DeepPoly* is more precise than prior work yet scales to large networks (Section 6).

We believe *DeepPoly* is a promising step towards addressing the challenge of reasoning about neural networks and a useful building block for proving complex specifications (e.g., rotations) and other applications of analysis. As an example, a promising direction for a future application is using
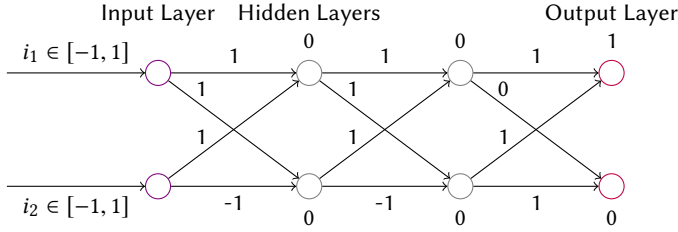
Fig. 2. Example feedforward neural network with ReLU activations.

our analysis during training. Specifically, because our abstract transformers for the output of a neuron are "point-wise" (i.e., can be computed in parallel), we can directly plug in these transformers into the latest systems which train neural networks using abstract interpretation on GPUs [Mirman et al. 2018]. As our transformers are substantially more precise than those of Mirman et al. [2018], we expect they can help improve the overall robustness of the trained network.

2 OVERVIEW

In this section, we provide an overview of our abstract domain on a small illustrative example. Full formal details are provided in later sections.

Running example on a feedforward network with ReLU activation. We consider the simple fully connected feedforward neural network with ReLU activations shown in Fig. 2. This network has already been trained and we have the learned weights shown in the figure. The network consists of four layers: an input layer, two hidden layers, and an output layer with two neurons each. The weights on the edges represent the learned coefficients of the weight matrix used by the affine transformations done at each layer. Note that these values are usually detailed floating point numbers (e.g., 0.031), however, here we use whole numbers to simplify the presentation. The learned bias for each neuron is shown above or below it. All of the biases in one layer constitute the translation vector of the affine transformation.

To compute its output, each neuron in the hidden layer applies an affine transformation based on the weight matrix and bias to its inputs (these inputs are the outputs of the neurons in the previous layer), producing a value v. Then, the neuron applies an activation function to v, in our example ReLU, which outputs v, if $v > 0$, and 0 otherwise. Thus, the input to every neuron goes through two stages: first, an affine transformation, followed by an activation function application. In the last layer, a final affine transform is applied to yield the output of the entire network, typically a class label that describes how the input is classified.

Specification. Suppose we work with a hypothetical image that contains only two pixels and the perturbation is such that it places both pixels in the range $[-1, 1]$ (pixels are usually in the range $[0, 1]$, however, we use $[-1, 1]$ to better illustrate our analysis). Our goal will be to prove that the output of the network at one of the output neurons is always greater than the output at the other one, for any possible input of two pixels in the range $[-1, 1]$. If the proof is successful, it implies that the network produces the same classification label for all of these images.

Abstract domain. To perform the analysis, we introduce an abstract domain with the appropriate abstract transformers that propagate the (abstract) input of the network through the layers, computing an over-approximation of the possible values at each neuron. Concretely, for our example, we need to propagate both intervals $[-1, 1]$ (one for each pixel) simultaneously. We now briefly discuss our abstract domain, which aims to balance analysis scalability and precision. Then we illustrate

$\langle x_1 \geq -1,$ $\quad$ $\langle x_3 \geq x_1 + x_2,$ $\quad$ $\langle x_5 \geq 0,$ $\quad$ $\langle x_7 \geq x_5 + x_6,$ $\quad$ $\langle x_9 \geq x_7,$ $\quad$ $\langle x_{11} \geq x_9 + x_{10} + 1,$
$x_1 \leq 1,$ $\quad$ $x_3 \leq x_1 + x_2,$ $\quad$ $x_5 \leq 0.5 \cdot x_3 + 1,$ $\quad$ $x_7 \leq x_5 + x_6,$ $\quad$ $x_9 \leq x_7,$ $\quad$ $x_{11} \leq x_9 + x_{10} + 1,$
$l_1 = -1,$ $\quad$ $l_3 = -2,$ $\quad$ $l_5 = 0,$ $\quad$ $l_7 = 0,$ $\quad$ $l_9 = 0,$ $\quad$ $l_{11} = 1,$
$u_1 = 1\rangle$ $\quad$ $u_3 = 2\rangle$ $\quad$ $u_5 = 2\rangle$ $\quad$ $u_7 = 3\rangle$ $\quad$ $u_9 = 3\rangle$ $\quad$ $u_{11} = 5.5\rangle$

$\langle x_2 \geq -1,$ $\quad$ $\langle x_4 \geq x_1 - x_2,$ $\quad$ $\langle x_6 \geq 0,$ $\quad$ $\langle x_8 \geq x_5 - x_6,$ $\quad$ $\langle x_{10} \geq 0,$ $\quad$ $\langle x_{12} \geq x_{10},$
$x_2 \leq 1,$ $\quad$ $x_4 \leq x_1 - x_2,$ $\quad$ $x_6 \leq 0.5 \cdot x_4 + 1,$ $\quad$ $x_8 \leq x_5 - x_6,$ $\quad$ $x_{10} \leq 0.5 \cdot x_8 + 1,$ $\quad$ $x_{11} \leq x_{10},$
$l_2 = -1,$ $\quad$ $l_4 = -2,$ $\quad$ $l_6 = 0,$ $\quad$ $l_8 = -2,$ $\quad$ $l_{10} = 0,$ $\quad$ $l_{12} = 0,$
$u_2 = 1\rangle$ $\quad$ $u_4 = 2\rangle$ $\quad$ $u_6 = 2\rangle$ $\quad$ $u_8 = 2\rangle$ $\quad$ $u_{10} = 2\rangle$ $\quad$ $u_{12} = 2\rangle$
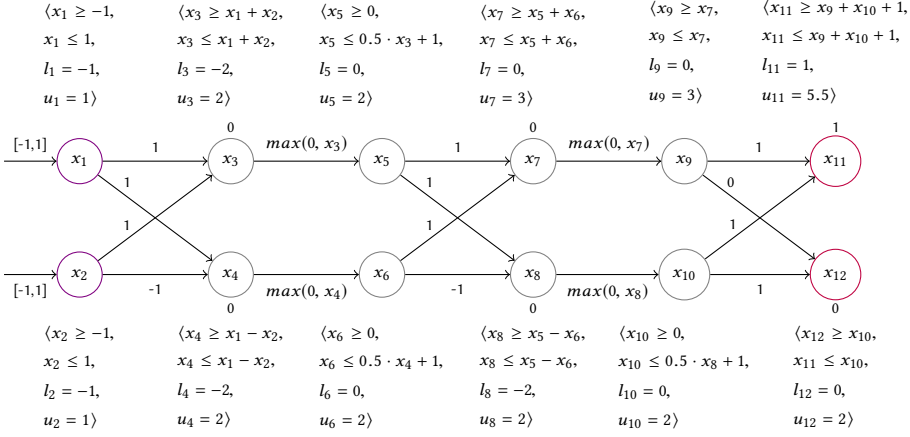
Fig. 3. The neural network from Fig. 2 transformed for analysis with the abstract domain.

its effect on our example network and discuss why we made certain choices in the approximation over others.

To perform the analysis, we first rewrite the network by expanding each neuron into two nodes: one for the associated affine transform and one for the ReLU activation. Transforming the network of Fig. 2 in this manner produces the network shown in Fig. 3. Because we assign a variable to each node, the network of Fig. 3 consists of $n = 12$ variables. Our abstract domain, formally described in Section 4, associates two constraints with each variable x_i: an upper polyhedral constraint and a lower polyhedral constraint. Additionally, the domain tracks auxiliary (concrete) bounds, one upper bound and one lower bound for each variable, describing a bounding box of the concretization of the abstract element. Our domain is less expressive than the Polyhedra domain [Cousot and Halbwachs 1978] because it bounds the number of conjuncts that can appear in the overall formula to $2n$ where n is the number of variables of the network. Such careful restrictions are necessary because supporting the full expressive power of convex polyhedra leads to an exponential number of constraints that make the analysis for thousands of neurons practically infeasible. We now discuss the two types of constraints and the two types of bounds, and show how they are computed on our example.

First, the lower ($a_i^{\leq}$) and upper ($a_i^{\geq}$) relational polyhedral constraints associated with x_i have the form $v + \sum_j w_j \cdot x_j$ where $v \in \mathbb{R} \cup \{-\infty, +\infty\}$, $w \in \mathbb{R}^n$, $\forall j \geq i. w_j = 0$. That is, a polyhedral constraint for x_i can consider and refer to variables "before" x_i in the network, but cannot refer to variables "after" x_i (because their coefficient is set to 0). Second, for the concrete lower and upper bounds of x_i, we use $l_i, u_i \in \mathbb{R} \cup \{-\infty, +\infty\}$, respectively. All abstract elements a in our domain satisfy the additional invariant that the interval $[l_i, u_i]$ overapproximates the set of values that the variable x_i can take (we formalize this requirement in Section 4).

Abstract interpretation of the network. We now illustrate the operation of our abstract interpreter (using the abstract domain above) on our example network, abstract input ($[-1, 1]$ for both pixels), and specification (which is to prove that any image in the concretization of $[-1, 1] \times [-1, 1]$ classifies to the same label).

The analysis starts at the input layer, i.e., in our example from x_1 and x_2, and simply propagates the inputs, resulting in $a_1^{\leq} = a_2^{\leq} = -1$, $a_1^{\geq} = a_2^{\geq} = 1$, $l_1 = l_2 = -1$, and $u_1 = u_2 = 1$. Next, the affine transform at the first layer updates the constraints for x_3 and x_4. The abstract transformer first
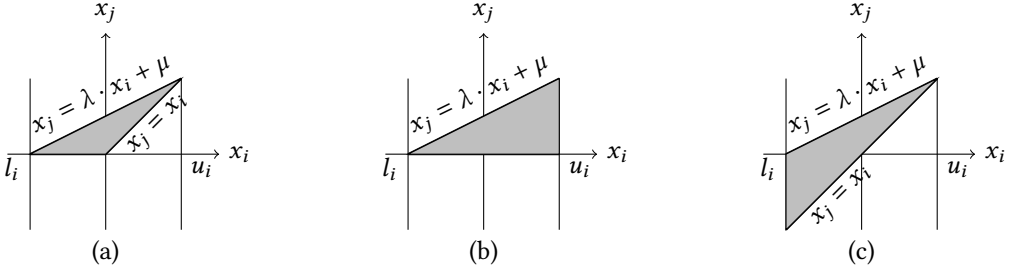
Fig. 4. Convex approximations for the ReLU function: (a) shows the convex approximation with the minimum area in the input-output plane, (b) and (c) show the two convex approximations proposed in this paper. In the figure, $\lambda = u_i/(u_i - l_i)$ and $\mu = -l_i u_i/(u_i - l_i)$.

adds the constraints

$$
\begin{aligned}
x_1 + x_2 &\leq x_3 \leq x_1 + x_2 \\
x_1 - x_2 &\leq x_4 \leq x_1 - x_2
\end{aligned}
\tag{1}
$$

The transformer uses these constraints and the constraints for x_1, x_2 to compute $l_3 = l_4 = -2$ and $u_3 = u_4 = 2$.

Next, the transformer for the ReLU activation is applied. In general, the output x_j of the ReLU activation on variable x_i is equivalent to the assignment $x_j := \max(0, x_i)$. If $u_i \leq 0$, then our abstract transformer sets the state of the variable x_j to $0 \leq x_j \leq 0, l_j = u_j = 0$. In this case, our abstract transformer is exact. If $l_i \geq 0$, then our abstract transformer adds $x_i \leq x_j \leq x_i, l_j = l_i, u_j = u_i$. Again, our abstract transformer is exact in this case.

However, when $l_i < 0$ and $u_i > 0$, the result cannot be captured exactly by our abstraction and we need to decide how to lose information. Fig. 4 shows several candidate convex approximations of the ReLU assignment in this case. The approximation of Fig. 4 (a) minimizes the area in the x_i, x_j plane, and would add the following relational constraints and concrete bounds for x_j:

$$
\begin{aligned}
&x_i \leq x_j, 0 \leq x_j, \\
&x_j \leq u_i(x_i - l_i)/(u_i - l_i). \\
&l_j = 0, u_j = u_i.
\end{aligned}
\tag{2}
$$

However, the approximation in (2) contains two lower polyhedra constraints for x_j, which we disallow in our abstract domain. The reason for this is the potential blowup in the number of constraints as the analysis proceeds. We will explain this effect in more detail later in this section.

To avoid this explosion we further approximate (2) by allowing only one lower bound. There are two ways of accomplish this, shown in Fig. 4 (b) and (c), both of which can be expressed in our domain. During analysis we always consider both and choose the one with the least area.

The approximation from Fig. 4 (b) adds the following constraints and bounds for x_j:

$$
\begin{aligned}
&0 \leq x_j \leq u_i(x_i - l_i)/(u_i - l_i), \\
&l_j = 0, u_j = u_i.
\end{aligned}
\tag{3}
$$

The approximation from Fig. 4 (c) adds the following constraints and bounds:

$$
\begin{aligned}
&x_i \leq x_j \leq u_i(x_i - l_i)/(u_i - l_i), \\
&l_j = l_i, u_j = u_i.
\end{aligned}
\tag{4}
$$

Note that it would be incorrect to set $l_j = 0$ in (4) above (instead of $l_j = l_i$). The reason is that this would break a key domain invariant which we aim to maintain, namely that the concretization of the two symbolic bounds for x_j is contained inside the concretization of the concrete bounds l_j and

u_j (we discuss this domain invariant later in Section 4). In particular, if we only consider the two symbolic bounds for x_j, then x_j would be allowed to take on negative values and these negative values would not be included in the region $[0, u_i]$. This domain invariant is important to ensure efficiency of our transformers and as we prove later, all of our abstract transformers maintain it.

Returning to our example, the area of the approximation in Fig. 4 (b) is $0.5 \cdot u_i \cdot (u_i - l_i)$ whereas the area in Fig. 4 (c) is $0.5 \cdot -l_i \cdot (u_i - l_i)$. We choose the tighter approximation, i.e., when $u_i \leq -l_i$, we add the constraints and the bounds from (3), otherwise we add the constraints and the bounds from (4). We note that the approximations in Fig. 4 (b) and (c) cannot be captured by the Zonotope abstraction as used in [Gehr et al. 2018; Singh et al. 2018a].

In our example, for both x_3 and x_4, we have $l_3 = l_4 = -2$ and $u_3 = u_4 = 2$. The areas are equal in this case; thus we choose (3) and get the following constraints and bounds for x_5 and x_6:

$$0 \leq x_5 \leq 0.5 \cdot x_3 + 1, \quad l_5 = 0, \; u_5 = 2,$$
$$0 \leq x_6 \leq 0.5 \cdot x_4 + 1, \quad l_6 = 0, \; u_6 = 2. \tag{5}$$

Next, we apply the abstract affine transformer, which first adds the following constraints for x_7 and x_8:

$$x_5 + x_6 \leq x_7 \leq x_5 + x_6,$$
$$x_5 - x_6 \leq x_8 \leq x_5 - x_6. \tag{6}$$

It is possible to compute bounds for x_7 and x_8 from the above equations by substituting the concrete bounds for x_5 and x_6. However, the resulting bounds are in general too imprecise. Instead, we can obtain better bounds by recursively substituting the polyhedral constraints until the bounds only depend on the input variables for which we then use their concrete bounds. In our example we substitute the relational constraints for x_5, x_6 from equation (5) to obtain:

$$0 \leq x_7 \leq 0.5 \cdot x_3 + 0.5 \cdot x_4 + 2,$$
$$-0.5 \cdot x_4 - 1 \leq x_8 \leq 0.5 \cdot x_3 + 1. \tag{7}$$

Replacing x_3 and x_4 with the constraints in (1), we get:

$$0 \leq x_7 \leq x_1 + 2,$$
$$-0.5 \cdot x_1 + 0.5 \cdot x_2 - 1 \leq x_8 \leq 0.5 \cdot x_1 + 0.5 \cdot x_2 + 1. \tag{8}$$

Now we use the concrete bounds of ± 1 for x_1, x_2 to obtain $l_7 = 0, u_7 = 3$ and $l_8 = -2, u_8 = 2$. Indeed, this is more precise than if we had directly substituted the concrete bounds for x_5 and x_6 in (6) because that would have produced concrete bounds $l_7 = 0, u_7 = 4$ (which are not as tight as the ones above).

Avoiding exponential blowup of the analysis. As seen above, to avoid the state space explosion, our analysis introduces exactly one polyhedral constraint for the lower bound of a variable. It is instructive to understand the effect of introducing more than one constraint via the ReLU approximation of Fig. 4 (a). This ReLU approximation introduces two lower relational constraints for both x_5 and x_6. Substituting them in (6) would have created four lower relational constraints for x_7. More generally, if the affine expression for a variable x_i contains p variables with positive coefficients and n variables with negative coefficients, then the number of possible lower and upper relational constraints is 2^p and 2^n, respectively, leading to an exponential blowup. This is the reason why we keep only one lower relational constraint for each variable in the network, and use either the ReLU transformer illustrated in Fig. 4 (b) or the one in Fig. 4 (c).

Asymptotic runtime. The computation of concrete bounds by the abstract affine transformer in the hidden layers is the most expensive step of our analysis. If there are L network layers and the maximum number of variables in a layer is n_{max}, then this step for one variable is in $O(n_{max}^2 \cdot L)$. Storing the concrete bounds ensures that the subsequent ReLU transformer has constant cost.

All our transformers work point-wise, i.e., they are independent for different variables since they only read constraints and bounds from the previous layers. This makes it possible to parallelize our analysis on both CPUs and GPUs. The work of [Mirman et al. 2018] defines pointwise Zonotope transformers for training neural networks on GPUs to be more robust against adversarial attacks. Our pointwise transformers are more precise than those used in [Mirman et al. 2018] and can be used to train more robust neural networks.

Precision vs. performance trade-off. We also note that our approach allows one to easily vary the precision-performance knob of the affine transformer in the hidden layers: (i) we can select a subset of variables for which to perform complete substitution all the way back to the first layer (the example above showed this for all variables), and (ii) we can decide at which layer we would like to stop the substitution and select the concrete bounds at that layer.

Returning to our example, next, the ReLU transformers are applied again. Since $l_7 = 0$, the ReLU transformer is exact for the assignment to x_9 and adds the relational constraints $x_7 \leq x_9 \leq x_7$ and the bounds $l_9 = 0, u_9 = 3$ for x_9. However, the transformer is not exact for the assignment to x_{10} and the following constraints and bounds for x_{10} are added:

$$0 \leq x_{10} \leq 0.5 \cdot x_8 + 1,$$
$$l_{10} = 0, u_{10} = 2. \tag{9}$$

Finally, the analysis reaches the output layer and the abstract affine transformer adds the following constraints for x_{11} and x_{12}:

$$x_9 + x_{10} + 1 \leq x_{11} \leq x_9 + x_{10} + 1$$
$$x_{10} \leq x_{12} \leq x_{10} \tag{10}$$

Again, backsubstitution up to the input layer yields $l_{11} = 1, u_{11} = 5.5$ and $l_{12} = 0, u_{12} = 2$. This completes our analysis of the neural network.

Checking the specification. Next, we check our specification, namely whether all concrete output values of one neuron are always greater than all concrete output values of the other neuron, i.e., if

$$\forall i_1, i_2 \in [-1, 1] \times [-1, 1], \ x_{11} > x_{12} \text{ or}$$
$$\forall i_1, i_2 \in [-1, 1] \times [-1, 1], \ x_{12} > x_{11},$$

where $x_{11}, x_{12} = N_{ff}(i_1, i_2)$ are the concrete values for variables x_{11} and x_{12} produced by our small feedforward (ff) neural network N_{ff} for inputs i_1, i_2.

In our simple example, this amounts to proving whether $x_{11} - x_{12} > 0$ or $x_{12} - x_{11} > 0$ holds given the abstract results computed by our analysis. Note that using the concrete bounds for x_{11} and x_{12}, that is, $l_{11}, l_{12}, u_{11},$ and u_{12} leads to the bound $[-1, 5.5]$ for $x_{11} - x_{12}$ and $[-5.5, 1]$ for $x_{12} - x_{11}$ and hence we cannot prove that either constraint holds. To address this imprecision, we first create a new temporary variable x_{13} and apply our abstract transformer for the assignment $x_{13} := x_{11} - x_{12}$. Our transformer adds the following constraint:

$$x_{11} - x_{12} \leq x_{13} \leq x_{11} - x_{12} \tag{11}$$

The transformer then computes bounds for x_{13} by backsubstitution (to the first layer), as described so far, which produces $l_{13} = 1$ and $u_{13} = 4$. As the (concrete) lower bound of x_{13} is greater than 0, our analysis concludes that $x_{11} - x_{12} > 0$ holds. Hence, we have proved our (robustness) specification. Of course, if we had failed to prove the property, we would have tried the same analysis using the

second constraint (i.e., $x_{12} > x_{11}$). And if that would fail, then we would declare that we are unable to prove the property. For our example, this was not needed since we were able to prove the first constraint.

3 BACKGROUND: NEURAL NETWORKS AND ADVERSARIAL REGIONS

In this section, we provide the minimal necessary background on neural networks and adversarial regions. Further, we show how we represent neural networks for our analysis.

Neural networks. Neural networks are functions $N: \mathbb{R}^m \to \mathbb{R}^n$ that can be implemented using straight-line programs (i.e., without loops) of a certain form. In this work, we focus on neural networks that follow a *layered architecture*, but all our methods can be used unchanged for more general neural network shapes. A layered neural network is given by a composition of l *layers* $f_1: \mathbb{R}^m \to \mathbb{R}^{n_1}, \ldots, f_l: \mathbb{R}^{n_{l-1}} \to \mathbb{R}^n$. Each layer f_i is one of the following: (i) an affine transformation $f_i(x) = Ax + b$ for some $A \in \mathbb{R}^{n_i \times n_{i-1}}$ and $b \in \mathbb{R}^{n_i}$ (in particular, convolution with one or more filters is an affine transformation), (ii) the *ReLU* activation function $f(x) = \max(0, x)$, where the maximum is applied componentwise, (iii) the sigmoid ($\sigma(x) = \frac{e^x}{e^x+1}$) or the tanh ($\tanh(x) = \frac{e^x-e^{-x}}{e^x+e^{-x}}$) activation function (again applied componentwise), or (iv) a max pool activation function, which subdivides the input x into multiple parts, and returns the maximal value in each part.

Neurons and activations. Each component of one of the vectors passed along through the layers is called a *neuron*, and its value is called an *activation*. There are three types of neurons: m input neurons whose activations form the input to the network, n output neurons whose activations form the output of the network, and all other neurons, called *hidden*, as they are not directly observed.

Classification. For a neural network that classifies its inputs to multiple possible labels, n is the number of distinct classes, and the neural network classifies a given input x to a given class k if $N(x)_k > N(x)_j$ for all j with $1 \leq j \leq n$ and $j \neq k$.

Adversarial region. In our evaluation, we consider the following (standard, e.g., see Carlini and Wagner [2017]) threat model: an input is drawn from the input distribution, perturbed by an adversary and then classified by the neural network. The perturbations that the adversary can perform are restricted and the set $X \subseteq \mathbb{R}^n$ of possible perturbations for a given input is called an *adversarial region*. The maximal possible error (i.e., the fraction of misclassified inputs) that the adversary can obtain by picking a worst-case input from each adversarial region is called the *adversarial error*. A neural network is *robust* for a given adversarial region if it classifies all inputs in that region the same way. This means that it is impossible for an adversary to influence the classification by picking an input from the adversarial region.

In our evaluation, we focus on verifying robustness for adversarial regions that can be represented using a set of interval constraints, i.e., $X = \bigtimes_{i=1}^{m} [l_i, u_i]$ for $l_i, u_i \in \mathbb{R} \cup \{-\infty, +\infty\}$. We also show how to use our analyzer to verify robustness against rotations which employ linear interpolation.

Network representation. For our analysis, we represent neural networks as a sequence of assignments, one per hidden and per output neuron. We need four kinds of assignments: ReLU assignments $x_i \leftarrow \max(0, x_j)$, sigmoid/tanh assignments $x_i \leftarrow g(x_j)$ for $g = \sigma$ or $g = \tanh$, max pool assignments $x_i \leftarrow \max_{j \in J} x_j$ and affine assignments $x_i \leftarrow v + \sum_j w_j \cdot x_j$. Convolutional layers can be described with affine assignments [Gehr et al. 2018]).

For example, we represent the neural network from Fig. 2 as the following program:

$$x_3 \leftarrow x_1 + x_2, x_4 \leftarrow x_1 - x_2, x_5 \leftarrow \max(0, x_3), x_6 \leftarrow \max(0, x_4),$$
$$x_7 \leftarrow x_5 + x_6, x_8 \leftarrow x_5 - x_6, x_9 \leftarrow \max(0, x_7), x_{10} \leftarrow \max(0, x_8),$$
$$x_{11} \leftarrow x_9 + x_{10} + 1, x_{12} \leftarrow x_{10}.$$

In Fig. 2, the adversarial region is given by $X = [-1, 1] \times [-1, 1]$. The variables x_1 and x_2 are the input to the neural network, and the variables x_{11} and x_{12} are the outputs of the network. Therefore, the final class of an input (x_1, x_2) is 1 if $x_{11} > x_{12}$, and 2 if $x_{11} < x_{12}$. To prove robustness we either need to prove that $\forall (x_1, x_2) \in X. \, x_{11} > x_{12}$ or that $\forall (x_1, x_2) \in X. \, x_{12} > x_{11}$.

We note that even though our experimental evaluation focuses on different kinds of robustness, our method and abstract domain are general and can be used to prove other properties as well.

4 ABSTRACT DOMAIN AND TRANSFORMERS

In this section, we introduce our abstract domain as well as the abstract transformers needed to analyze the four kinds of assignment statements mentioned previously.

Elements in our abstract domain $\mathcal{A}_n$ consist of a set of polyhedral constraints of a specific form, over n variables. Each constraint relates one variable to a linear combination of the variables of a smaller index. Each variable has two associated polyhedral constraints: one lower bound and one upper bound. In addition, the abstract element records derived interval bounds for each variable. Formally, an abstract element $a \in \mathcal{A}_n$ over n variables can be written as a tuple $a = \langle a^{\leq}, a^{\geq}, l, u \rangle$ where

$$a_i^{\leq}, a_i^{\geq} \in \{x \mapsto v + \sum_{j \in [i-1]} w_j \cdot x_j \mid v \in \mathbb{R} \cup \{-\infty, +\infty\}, w \in \mathbb{R}^{i-1}\} \text{ for } i \in [n]$$

and $l, u \in (\mathbb{R} \cup \{-\infty, +\infty\})^n$. Here, we use the notation $[n] := \{1, 2, \ldots, n\}$. The concretization function $\gamma_n \colon \mathcal{A}_n \to \mathcal{P}(\mathbb{R}^n)$ is then given by

$$\gamma_n(a) = \{x \in \mathbb{R}^n \mid \forall i \in [n]. \, a_i^{\leq}(x) \leq x_i \wedge a_i^{\geq}(x) \geq x_i\}.$$

Domain Invariant. All abstract elements in our domain additionally satisfy the following invariant: $\gamma_n(a) \subseteq \bigtimes_{i \in [n]} [l_i, u_i]$. In other words, every abstract element in our domain maintains concrete lower and upper bounds which over-approximate the two symbolic bounds. This property is essential for creating efficient abstract transformers.

To simplify our exposition of abstract transformers, we will only consider the case where all variables are bounded, which is always the case when our analysis is applied to neural networks. Further, we require that variables are assigned exactly once, in increasing order of their indices. Our abstract transformers $T_f^{\#}$ for a deterministic function $f \colon \mathcal{A}^m \to \mathcal{A}^n$ satisfy the following soundness property: $T_f(\gamma_m(a)) \subseteq \gamma_n(T_f^{\#}(a))$ for all $a \in \mathcal{A}^m$, where T_f is the corresponding concrete transformer of f, given by $T_f(X) = \{f(x) \mid x \in X\}$.

4.1 ReLU Abstract Transformer

Let $f \colon \mathbb{R}^{i-1} \to \mathbb{R}^i$ be a function that executes the assignment $x_i \leftarrow \max(0, x_j)$ for some $j < i$.

The corresponding abstract ReLU transformer is $T_f^{\#}(\langle a^{\leq}, a^{\geq}, l, u \rangle) = \langle a'^{\leq}, a'^{\geq}, l', u' \rangle$ where $a_k'^{\leq} = a_k^{\leq}$, $a_k'^{\geq} = a_k^{\geq}$, $l_k' = l_k$ and $u_k' = u_k$ for $k < i$. For the new component i, there are three cases. If $u_j \leq 0$, then $a_i'^{\leq}(x) = a_i'^{\geq}(x) = 0$ and $l_i' = u_i' = 0$. If $0 \leq l_j$, then $a_i'^{\leq}(x) = a_i'^{\geq}(x) = x_j$, $l_i' = l_j$ and $u_i' = u_j$.

Otherwise, the abstract ReLU transformer approximates the assignment by a set of linear constraints forming its convex hull when it is restricted to the interval $[l_j, u_j]$:

$$0 \leq x_i, \; x_j \leq x_i,$$
$$x_i \leq u_j(x_j - l_j)/(u_j - l_j).$$

As there is only one upper bound for x_i, we obtain the following rule:

$$a_i'^{\geq}(x) = u_j(x_j - l_j)/(u_j - l_j).$$

On the other hand, we have two lower bounds for x_i: x_j and 0. Any convex combination of those two constraints is still a valid lower bound. Therefore, we can set

$$a_i'^{\leq}(x) = \lambda \cdot x_j,$$

for any $\lambda \in [0, 1]$. We select the $\lambda \in \{0, 1\}$ that minimizes the area of the resulting shape in the (x_i, x_j)-plane. Finally, we set $l_i' = \lambda \cdot l_j$ and $u_i' = u_j$.

4.2 Sigmoid and Tanh Abstract Transformers

Let $g \colon \mathbb{R} \to \mathbb{R}$ be a continuous, twice-differentiable function with $g'(x) > 0$ and $0 \leq g''(x) \Leftrightarrow x \leq 0$ for all $x \in \mathbb{R}$ where g' and g'' are the first and second derivatives of g. The sigmoid function $\sigma(x) = \frac{e^x}{e^x + 1}$ and the tanh function $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ both satisfy these conditions. For such a function g, let $f \colon \mathbb{R}^{i-1} \to \mathbb{R}^i$ be the function that executes the assignment $x_i \leftarrow g(x_j)$ for $j < i$.

The corresponding abstract transformer is $T_f^{\#}(\langle a^{\leq}, a^{\geq}, l, u \rangle) = \langle a'^{\leq}, a'^{\geq}, l', u' \rangle$ where $a_k'^{\leq} = a_k^{\leq}$, $a_k'^{\geq} = a_k^{\geq}$, $l_k' = l_k$ and $u_k' = u_k$ for $k < i$. For the new component i, we set $l_i' = g(l_j)$ and $u_i' = g(u_j)$. If $l_j = u_j$, then $a_i'^{\leq}(x) = a_i'^{\geq}(x) = g(l_j)$. Otherwise, we consider $a_i'^{\leq}(x)$ and $a_i'^{\geq}(x)$ separately. Let $\lambda = (g(u_j) - g(l_j))/(u_j - l_j)$ and $\lambda' = \min(g'(l_j), g'(u_j))$. If $0 < l_j$, then $a_i'^{\leq}(x) = g(l_j) + \lambda \cdot (x_j - l_j)$, otherwise $a_i'^{\leq}(x) = g(l_j) + \lambda' \cdot (x_j - l_j)$. Similarly, if $u_j \leq 0$, then $a_i'^{\geq}(x) = g(u_j) + \lambda \cdot (x_j - u_j)$ and $a_i'^{\geq}(x) = g(u_j) + \lambda' \cdot (x_j - u_j)$ otherwise.

4.3 Max Pool Abstract Transformer

Let $f \colon \mathbb{R}^{i-1} \to \mathbb{R}^i$ be a function that executes $x_i \leftarrow \max_{j \in J} x_j$ for some $J \subseteq [i-1]$. The corresponding abstract max pool transformer is $T_f^{\#}(\langle a^{\leq}, a^{\geq}, l, u \rangle) = \langle a'^{\leq}, a'^{\geq}, l', u' \rangle$ where $a_k'^{\leq} = a_k^{\leq}$, $a_k'^{\geq} = a_k^{\geq}$, $l_k' = l_k$ and $u_k' = u_k$ for $k < i$. For the new component i, there are two cases. If there is some $k \in J$ with $u_j < l_k$ for all $j \in J \setminus \{k\}$, then $a_i'^{\leq}(x) = a_i'^{\geq}(x) = x_k$, $l_i' = l_k$ and $u_i' = u_k$. Otherwise, we choose $k \in J$ such that l_k is maximized and set $a_i'^{\leq}(x) = x_k$, $l_i' = l_k$ and $a_i'^{\geq}(x) = u_i' = \max_{j \in J} u_j$.

4.4 Affine Abstract Transformer

Let $f \colon \mathbb{R}^{i-1} \to \mathbb{R}^i$ be a function that executes $x_i \leftarrow v + \sum_{j \in [i-1]} w_j \cdot x_j$ for some $w \in \mathbb{R}^{i-1}$. The corresponding abstract affine transformer is $T_f^{\#}(\langle a^{\leq}, a^{\geq}, l, u \rangle) = \langle a'^{\leq}, a'^{\geq}, l', u' \rangle$ where $a_k'^{\leq} = a_k^{\leq}$, $a_k'^{\geq} = a_k^{\geq}$, $l_k' = l_k$ and $u_k' = u_k$ for $k < i$. Further, $a_i'^{\leq}(x) = a_i'^{\geq}(x) = v + \sum_{j \in [i-1]} w_j \cdot x_j$.

To compute l_i and u_i, we repeatedly substitute bounds for x_j into the constraint, until no further substitution is possible. Formally, if we want to obtain l_i', we start with $b_1(x) = a_i'^{\leq}(x)$. If we have $b_s(x) = v' + \sum_{j \in [k]} w_j' \cdot x_j$ for some $k \in [i-1]$, $v' \in \mathbb{R}$, $w' \in \mathbb{R}^k$, then

$$b_{s+1}(x) = v' + \sum_{j \in [k]} \left(\max(0, w_j') \cdot a_j'^{\leq}(x) + \min(w_j', 0) \cdot a_j'^{\geq}(x) \right).$$

We iterate until we reach $b_{s'}$ with $b_{s'}(x) = v''$ (i.e., s' is the smallest number with this property). We then set $l_i' = v''$.

We compute u'_i in an analogous fashion: to obtain u'_i, we start with $c_1(x) = a'^{\geq}_i(x)$. If we have $c_t(x) = v' + \sum_{j \in [k]} w'_j \cdot x_j$ for some $k \in [i-1]$, $v' \in \mathbb{R}$, $w' \in \mathbb{R}^k$, then

$$c_{t+1}(x) = v' + \sum_{j \in [k]} \left(\max(0, w'_j) \cdot a'^{\geq}_j(x) + \min(w'_j, 0) \cdot a'^{\leq}_j(x) \right).$$

We iterate until we reach $c_{t'}$ with $c_{t'}(x) = v''$. We then set $u'_i = v''$.

4.5 Neural Network Robustness Analysis

We now show how to use our analysis to prove robustness of a neural network with p inputs, q hidden activations and r output classes, resulting in a total of $p + q + r$ activations. More explicitly, our goal is to prove that the neural network classifies all inputs satisfying the given interval constraints (the adversarial region) to a particular class k.

We first create an abstract element $a = \langle a^{\leq}, a^{\geq}, l, r \rangle$ over p variables, where $a^{\leq}_i(x) = l_i$ and $a^{\geq}_i(x) = u_i$ for all i. The bounds l_i and u_i are initialized such that they describe the adversarial region. For example, for the adversarial region in Fig. 2, we get

$$a = \langle (x \mapsto l_1, x \mapsto l_2), (x \mapsto u_1, x \mapsto u_2), (-1, -1), (1, 1) \rangle.$$

Then, the analysis proceeds by processing assignments for all q hidden activations and the r output activations of the neural network, layer by layer, processing nodes in ascending order of variable indices, using their respective abstract transformers. Finally, the analysis executes the following $r - 1$ (affine) assignments in the abstract:

$$x_{p+q+r+1} \leftarrow x_{p+q+k} - x_{p+q+1}, \ldots, x_{p+q+r+(k-1)} \leftarrow x_{p+q+k} - x_{p+q+(k-1)},$$
$$x_{p+q+r+k} \leftarrow x_{p+q+k} - x_{p+q+(k+1)}, \ldots, x_{p+q+r+(r-1)} \leftarrow x_{p+q+k} - x_{p+q+r}.$$

As output class k has the highest activation if and only if those differences are all positive, the neural network is proved robust if for all $i \in \{p + q + r + 1, \ldots, p + q + r + (r-1)\}$ we have $0 < l_i$. Otherwise, our robustness analysis fails to verify.

For the neural network in Fig. 2, if we want to prove that class 1 is most likely, this means we execute one additional assignment $x_{13} \leftarrow x_{11} - x_{12}$. Abstract interpretation derives the bounds $l_{13} = 1$, $u_{13} = 4$. The neural network is proved robust, because l_{13} is positive.

The above discussion showed how to use our abstract transformers to prove robustness. However, a similar procedure could be used to prove standard pre/post conditions (by performing the analysis starting with the pre-condition).

4.6 Correctness of Abstract Transformers

In this section, we prove that our abstract transformers are sound, and that they preserve the invariant. Formally, for $T^{\#}_f(a) = a'$ we have $T_f(\gamma_{i-1}(a)) \subseteq \gamma_i(a')$ and $\gamma_i(a') \subseteq \bigtimes_{k \in [i]} [l'_k, u'_k]$.

Soundness. We first prove a lemma that is needed to prove soundness of our ReLU transformer.

LEMMA 4.1. *For $l < 0$, $0 < u$, $l \leq x \leq u$, and $\lambda \in [0, 1]$ we have $\lambda \cdot x \leq \max(0, x) \leq u \cdot \frac{x-l}{u-l}$.*

PROOF. If $x < 0$, then $\lambda \cdot x < 0 = \max(0, x)$. If $x \geq 0$, then $\lambda \cdot x \leq x = \max(0, x)$. If $x < 0$, then $\max(0, x) = 0 \leq u \cdot \frac{x-l}{u-l}$. If $x \geq 0$ then $\max(0, x) = x \leq u \cdot \frac{x-l}{u-l}$ because $x \cdot (-l) \leq u \cdot (-l) \Leftrightarrow x \cdot u - x \cdot l \leq x \cdot u - u \cdot l \Leftrightarrow x \cdot (u - l) \leq u \cdot (x - l)$. □

THEOREM 4.2. *The ReLU abstract transformer is sound.*

PROOF. Let $f\colon \mathbb{R}^{i-1} \to \mathbb{R}^i$ execute the assignment $x_i \leftarrow \max(0, x_j)$ for some $j < i$, and let $a \in \mathcal{A}_{i-1}$ be arbitrary. We have $\gamma_{i-1}(a) \subseteq \bigtimes_{k \in [i-1]}[l_k, u_k]$ and

$$
\begin{aligned}
T_f(\gamma_{i-1}(a)) &= \{f(x) \mid x \in \gamma_{i-1}(a)\} \\
&= \{(x_1, \dots, x_{i-1}, \max(0, x_j)) \mid (x_1, \dots, x_{i-1}) \in \gamma_{i-1}(a)\} \\
&= \{x \in \mathbb{R}^i \mid (x_1, \dots, x_{i-1}) \in \gamma_{i-1}(a) \land x_i = \max(0, x_j)\} \\
&= \{x \in \mathbb{R}^i \mid (\forall k \in [i-1].\ a_k^{\leq}(x) \leq x_k \land a_k^{\geq}(x) \geq x_k) \land x_i = \max(0, x_j)\}.
\end{aligned}
$$

If $u_j \leq 0$, we have that $(\forall k \in [i-1].\ a_k^{\leq}(x) \leq x_k \land a_k^{\geq}(x) \geq x_k)$ implies $x_j \leq 0$, and

$$
\begin{aligned}
T_f(\gamma_{i-1}(a)) &= \{x \in \mathbb{R}^i \mid (\forall k \in [i-1].\ a_k^{\leq}(x) \leq x_k \land a_k^{\geq}(x) \geq x_k) \land x_i = \max(0, x_j) \land x_j \leq 0\} \\
&= \{x \in \mathbb{R}^i \mid (\forall k \in [i-1].\ a_k^{\leq}(x) \leq x_k \land a_k^{\geq}(x) \geq x_k) \land x_i = 0\} \\
&= \{x \in \mathbb{R}^i \mid \forall k \in [i].\ a_k'^{\leq}(x) \leq x_k \land a_k'^{\geq}(x) \geq x_k\} \\
&= \gamma_i(T_f^{\#}(a)).
\end{aligned}
$$

Otherwise, if $0 \leq l_j$, we have that $(\forall k \in [i-1].\ a_k^{\leq}(x) \leq x_k \land a_k^{\geq}(x) \geq x_k)$ implies $0 \leq x_j$, and

$$
\begin{aligned}
T_f(\gamma_{i-1}(a)) &= \{x \in \mathbb{R}^i \mid (\forall k \in [i-1].\ a_k^{\leq}(x) \leq x_k \land a_k^{\geq}(x) \geq x_k) \land x_i = \max(0, x_j) \land 0 \leq x_j\} \\
&= \{x \in \mathbb{R}^i \mid (\forall k \in [i-1].\ a_k^{\leq}(x) \leq x_k \land a_k^{\geq}(x) \geq x_k) \land x_i = x_j\} \\
&= \{x \in \mathbb{R}^i \mid \forall k \in [i].\ a_k'^{\leq}(x) \leq x_k \land a_k'^{\geq}(x) \geq x_k\} \\
&= \gamma_i(T_f^{\#}(a)).
\end{aligned}
$$

Otherwise, we have $l_j < 0$ and $0 < u_j$ and that $(\forall k \in [i-1].\ a_k^{\leq}(x) \leq x_k \land a_k^{\geq}(x) \geq x_k)$ implies $l_j \leq x_j \leq u_j$ and therefore

$$
\begin{aligned}
T_f(\gamma_{i-1}(a)) &= \{x \in \mathbb{R}^i \mid (\forall k \in [i-1].\ a_k^{\leq}(x) \leq x_k \land a_k^{\geq}(x) \geq x_k) \land x_i = \max(0, x_j)\} \\
&\subseteq \{x \in \mathbb{R}^i \mid (\forall k \in [i-1].\ a_k^{\leq}(x) \leq x_k \land a_k^{\geq}(x) \geq x_k) \land x_i \leq u_j \cdot \frac{x_j - l_j}{u_j - l_j} \land x_i \geq \lambda \cdot x_j\} \\
&= \{x \in \mathbb{R}^i \mid \forall k \in [i].\ a_k'^{\leq}(x) \leq x_k \land a_k'^{\geq}(x) \geq x_k\} \\
&= \gamma_i(T_f^{\#}(a)).
\end{aligned}
$$

Therefore, in all cases, $T_f(\gamma_{i-1}(a)) \subseteq \gamma_i(T_f^{\#}(a))$. Note that we lose precision only in the last case. □

THEOREM 4.3. *The sigmoid and tanh abstract transformers are sound.*

PROOF. A function $g\colon \mathbb{R} \to \mathbb{R}$ with $g'(x) > 0$ and $0 \leq g''(x) \Leftrightarrow 0 \leq x$ is monotonically increasing, and furthermore, $g|_{(-\infty, 0]}$ (the restriction to $(-\infty, 0]$) is convex and $g|_{(0, \infty)}$ is concave.

Let $f\colon \mathbb{R}^{i-1} \to \mathbb{R}^i$ execute the assignment $x_i \leftarrow g(x_j)$ for some $j < i$, and let $a \in \mathcal{A}_{i-1}$ be arbitrary. We have

$$
\begin{aligned}
T_f(\gamma_{i-1}(a)) &= \{f(x) \mid x \in \gamma_{i-1}(a)\} \\
&= \{(x_1, \dots, x_{i-1}, g(x_j)) \mid (x_1, \dots, x_{i-1}) \in \gamma_{i-1}(a)\} \\
&= \{x \in \mathbb{R}^i \mid (\forall k \in [i-1].\ a_k^{\leq}(x) \leq x_k \land a_k^{\geq}(x) \geq x_k) \land x_i = g(x_j)\}.
\end{aligned}
$$

If $l_j = u_j$, then $(\forall k \in [i-1].\, a_k^{\leq}(x) \leq x_k \wedge a_k^{\geq}(x) \geq x_k)$ implies $x_j = l_j$ and therefore

$$
\begin{aligned}
T_f(\gamma_{i-1}(a)) &= \{x \in \mathbb{R}^i \mid (\forall k \in [i-1].\, a_k^{\leq}(x) \leq x_k \wedge a_k^{\geq}(x) \geq x_k) \wedge x_i = g(x_j)\}. \\
&= \{x \in \mathbb{R}^i \mid (\forall k \in [i-1].\, a_k^{\leq}(x) \leq x_k \wedge a_k^{\geq}(x) \geq x_k) \wedge x_i = g(l_j)\} \\
&= \{x \in \mathbb{R}^i \mid (\forall k \in [i-1].\, a_k^{\leq}(x) \leq x_k \wedge a_k^{\geq}(x) \geq x_k) \wedge g(l_j) \leq x_i \wedge x_j \leq g(l_j)\} \\
&= \{x \in \mathbb{R}^i \mid \forall k \in [i].\, a_k'^{\leq}(x) \leq x_k \wedge a_k'^{\geq}(x) \geq x_k\} \\
&= \gamma_i(T_f^{\#}(a)).
\end{aligned}
$$

Therefore, the transformer is exact in this case.

Otherwise, we need to show that $(\forall k \in [i-1].\, a_k^{\leq}(x) \leq x_k \wedge a_k^{\geq}(x) \geq x_k) \wedge x_i = g(x_j)$ implies $a_i'^{\leq}(x) \leq x_i$ and $a_i'^{\geq}(x) \geq x_i$. We let $x \in \mathbb{R}^i$ be arbitrary with $(\forall k \in [i-1].\, a_k^{\leq}(x) \leq x_k \wedge a_k^{\geq}(x) \geq x_k) \wedge x_i = g(x_j)$ and consider $a_i'^{\leq}(x)$ and $a_i'^{\geq}(x)$ separately. Recall that $\lambda = (g(u_j) - g(l_j))/(u_j - l_j)$ and $\lambda' = \min(g'(l_j), g'(u_j))$. If $0 \leq l_j$, then, because g is concave on positive inputs,

$$
\begin{aligned}
a_i'^{\leq}(x) &= g(l_j) + \lambda \cdot (x_j - l_j) = \left(1 - \frac{x_j - l_j}{u_j - l_j}\right) \cdot g(l_j) + \frac{x_j - l_j}{u_j - l_j} \cdot g(u_j) \\
&\leq g\left(\left(1 - \frac{x_j - l_j}{u_j - l_j}\right) \cdot l_j + \frac{x_j - l_j}{u_j - l_j} \cdot u_j\right) = g(x_j) = x_i.
\end{aligned}
$$

Otherwise, because g' is non-decreasing on $(-\infty, 0]$ and decreasing on $(0, \infty)$, we have that $\lambda' = \min(g'(l_j), g'(u_j)) \leq g'(\xi)$ for all $\xi \in [l_j, u_j]$. Therefore,

$$
a_i'^{\leq}(x) = g(l_j) + \lambda' \cdot (x_j - l_j) = g(l_j) + \int_{l_j}^{x_j} \lambda' \mathrm{d}\xi \leq g(l_j) + \int_{l_j}^{x_j} g'(\xi) \mathrm{d}\xi = g(x_j).
$$

The proof of $a_i'^{\geq}(x) \geq x_i$ is analogous.

We conclude

$$
\begin{aligned}
T_f(\gamma_{i-1}(a)) &= \{x \in \mathbb{R}^i \mid (\forall k \in [i-1].\, a_k^{\leq}(x) \leq x_k \wedge a_k^{\geq}(x) \geq x_k) \wedge x_i = g(x_j)\}. \\
&\subseteq \{x \in \mathbb{R}^i \mid (\forall k \in [i-1].\, a_k^{\leq}(x) \leq x_k \wedge a_k^{\geq}(x) \geq x_k) \wedge a_k'^{\leq}(x) \leq x_i \wedge a_i'^{\geq}(x) \geq x_i\} \\
&= \{x \in \mathbb{R}^i \mid (\forall k \in [i].\, a_k'^{\leq}(x) \leq x_k \wedge a_k'^{\geq}(x) \geq x_k)\} \\
&= \gamma_i(T_f^{\#}(a)),
\end{aligned}
$$

where the inclusion is strict because we have dropped the constraint $x_i = g(x_j)$. Therefore, the abstract transformer is sound. □

THEOREM 4.4. *The max pool abstract transformer is sound.*

PROOF. Let $f \colon \mathbb{R}^{i-1} \to \mathbb{R}^i$ execute the assignment $x_i \leftarrow \max_{j \in J}(0, x_j)$ for some $J \subseteq [i-1]$, and let $a \in \mathcal{A}_{i-1}$ be arbitrary. We have

$$
\begin{aligned}
T_f(\gamma_{i-1}(a)) &= \{f(x) \mid x \in \gamma_{i-1}(a)\} \\
&= \{(x_1, \ldots, x_{i-1}, \max_{j \in J} x_j) \mid (x_1, \ldots, x_{i-1}) \in \gamma_{i-1}(a)\} \\
&= \{x \in \mathbb{R}^i \mid (\forall k \in [i-1].\, a_k^{\leq}(x) \leq x_k \wedge a_k^{\geq}(x) \geq x_k) \wedge x_i = \max_{j \in J} x_j\}.
\end{aligned}
$$

There are two cases. If there is some $k \in J$ with $u_j < l_k$ for all $j \in J \setminus \{k\}$, then $(\forall k \in [i-1]. \, a_k^{\leq}(x) \leq x_k \land a_k^{\geq}(x) \geq x_k)$ implies that $\max_{j \in J} x_j = x_k$ and therefore

$$
\begin{aligned}
T_f(\gamma_{i-1}(a)) &= \{x \in \mathbb{R}^i \mid (\forall k \in [i-1]. \, a_k^{\leq}(x) \leq x_k \land a_k^{\geq}(x) \geq x_k) \land x_i = \max_{j \in J} x_j\} \\
&= \{x \in \mathbb{R}^i \mid (\forall k \in [i-1]. \, a_k^{\leq}(x) \leq x_k \land a_k^{\geq}(x) \geq x_k) \land x_i = x_k\} \\
&= \{x \in \mathbb{R}^i \mid \forall k \in [i]. \, a_k'^{\leq}(x) \leq x_k \land a_k'^{\geq}(x) \geq x_k\} \\
&= \gamma_i(T_f^{\#}(a)).
\end{aligned}
$$

Otherwise, the transformer chooses a k with maximal l_k. We also know that $(\forall k \in [i-1]. \, a_k^{\leq}(x) \leq x_k \land a_k^{\geq}(x) \geq x_k)$ implies $x_j \leq u_j$ for all $j \in J$, and therefore

$$
\begin{aligned}
T_f(\gamma_{i-1}(a)) &= \{x \in \mathbb{R}^i \mid (\forall k \in [i-1]. \, a_k^{\leq}(x) \leq x_k \land a_k^{\geq}(x) \geq x_k) \land x_i = \max_{j \in J} x_j)\} \\
&\subseteq \{x \in \mathbb{R}^i \mid (\forall k \in [i-1]. \, a_k^{\leq}(x) \leq x_k \land a_k^{\geq}(x) \geq x_k) \land x_k \leq x_i \land \max_{j \in J} u_j \geq x_i)\} \\
&= \{x \in \mathbb{R}^i \mid \forall k \in [i]. \, a_k'^{\leq}(x) \leq x_k \land a_k'^{\geq}(x) \geq x_k\} \\
&= \gamma_i(T_f^{\#}(a)).
\end{aligned}
$$

In summary, in both cases, $T_f(\gamma_{i-1}(a)) \subseteq \gamma_i(T_f^{\#}(a))$. □

THEOREM 4.5. *The affine abstract transformer is sound and exact.*

PROOF. Let $f : \mathbb{R}^{i-1} \to \mathbb{R}^i$ execute the assignment $x_i \leftarrow v + \sum_{j \in [i-1]} w_j \cdot x_j$ for some $v \in \mathbb{R}, w \in \mathbb{R}^{i-1}$, and let $a \in \mathcal{A}_{i-1}$ be arbitrary. We have

$$
\begin{aligned}
T_f(\gamma_{i-1}(a)) &= \{f(x) \mid x \in \gamma_{i-1}(a)\} \\
&= \{(x_1, \ldots, x_{i-1}, v + \sum_{j \in [i-1]} w_j \cdot x_j) \mid (x_1, \ldots, x_{i-1}) \in \gamma_{i-1}(a)\} \\
&= \{x \in \mathbb{R}^i \mid (x_1, \ldots, x_{i-1}) \in \gamma_{i-1}(a) \land x_i = v + \sum_{j \in [i-1]} w_j \cdot x_j)\} \\
&= \{x \in \mathbb{R}^i \mid (\forall k \in [i-1]. \, a_k^{\leq}(x) \leq x_k \land a_k^{\geq}(x) \geq x_k) \land x_i = v + \sum_{j \in [i-1]} w_j \cdot x_j)\} \\
&= \{x \in \mathbb{R}^i \mid \forall k \in [i]. \, a_k'^{\leq}(x) \leq x_k \land a_k'^{\geq}(x) \geq x_k\} \\
&= \gamma_i(T_f^{\#}(a)).
\end{aligned}
$$

Thus, $T_f(\gamma_{i-1}(a)) = \gamma_i(T_f^{\#}(a))$. □

Invariant. We now prove that our abstract transformers preserve the invariant. For each of our abstract transformers $T_f^{\#}$, we have to show that for $T_f^{\#}(a) = a'$, we have $\gamma_i(a') \subseteq \bigtimes_{j \in [i]}[l_j', u_j']$. Note that the constraints $(\forall k \in [i]. \, a_k'^{\leq}(x) \leq x_k \land a_k'^{\geq}(x) \geq x_k)$ include all constraints of a. We first assume that the invariant holds for a, thus $(\forall k \in [i-1]. \, a_k^{\leq}(x) \leq x_k \land a_k^{\geq}(x) \geq x_k)$ implies the bounds $(\forall k \in [i-1]. \, l_k \leq x_k \leq u_k)$, which are equivalent to $(\forall k \in [i-1]. \, l_k' \leq x_k \leq u_k')$, because our abstract transformers preserve the bounds of existing variables. It therefore suffices to show that $(\forall k \in [i]. \, a_k'^{\leq}(x) \leq x_k \land a_k'^{\geq}(x) \geq x_k)$ implies $l_i' \leq x_i \leq u_i'$.

THEOREM 4.6. *The ReLU abstract transformer preserves the invariant.*

PROOF. If $u_j \leq 0$, we have $a_i'^{\leq}(x) = a_i'^{\geq}(x) = 0$ and therefore $(\forall k \in [i]. \, a_k'^{\leq}(x) \leq x_k \land a_k'^{\geq}(x) \geq x_k)$ implies $0 = l_i' = a_i'^{\leq}(x) \leq x_i \leq a_i'^{\geq}(x) = u_i' = 0$. If $0 \leq l_j$, we have $a_i'^{\leq}(x) = a_i'^{\geq}(x) = x_j$ and therefore $(\forall k \in [i]. \, a_k'^{\leq}(x) \leq x_k \land a_k'^{\geq}(x) \geq x_k)$ implies $l_i' = l_j \leq x_j = x_i \leq u_j = u_i'$. Otherwise, we have $l_j < 0$ and $0 < u_j$, as well as $a'^{\leq}(x)_i = \lambda \cdot x_j$, $a'^{\geq}(x)_i = u_j \cdot \frac{x_j - l_j}{u_j - l_j}$, and so $(\forall k \in [i]. \, a_k'^{\leq}(x) \leq x_k \land a_k'^{\geq}(x) \geq x_k)$ implies $l_i' = \lambda \cdot l_j \leq x_i \leq u_j = u_i'$. □

THEOREM 4.7. *The sigmoid and tanh abstract transformers preserve the invariant.*

PROOF. The constraints $(\forall k \in [i].\, a_k'^{\le}(x) \le x_k \wedge a_k'^{\ge}(x) \ge x_k)$ imply $l_j \le x_j \le u_j$ and by monotonicity of g, we obtain $l_i' = g(l_j) \le x_i \le g(u_j) = u_j'$ using $x_i = g(x_j)$. □

THEOREM 4.8. *The max pool abstract transformer preserves the invariant.*

PROOF. The max pool transformer either sets $a_i'^{\le}(x) = a_i'^{\ge}(x) = x_k$ and $l_i' = l_k$ and $u_i' = u_k$, in which case $(\forall k \in [i].\, a_k'^{\le}(x) \le x_k \wedge a_k'^{\ge}(x) \ge x_k)$ implies $l_i' = l_k \le x_k = x_i \le u_k = u_k'$, or it sets $a_i'^{\le}(x) = x_k, l_i' = l_k$ and $u_i' = a_i'^{\ge}(x)$, such that $(\forall k \in [i].\, a_k'^{\le}(x) \le x_k \wedge a_k'^{\ge}(x) \ge x_k)$, which implies $l_i' \le x_k \le u_i'$ □

THEOREM 4.9. *The affine abstract transformer preserves the invariant.*

PROOF. Note that s' and t' are finite, because in each step, the maximal index of a variable whose coefficient in, respectively, b_s and c_t is nonzero decreases by at least one. Assume $\forall k \in [i].\, a_k'^{\le}(x) \le x_k \wedge a_k'^{\ge}(x) \ge x_k$. We have to show that $b_{s'}(x) \le x_i$ and $c_{t'}(x) \ge x_i$. It suffices to show that $\forall s \in [s'].\, b_s(x) \le x_i$ and $\forall t \in [t'].\, c_t(x) \ge x_i$.

To show $\forall s \in [s'].\, b_s(x) \le x_i$, we use induction on s. We have $b_1(x) = a_i'^{\le}(x) \le x_i$. Assuming $b_s(x) \le x_i$ and $b_s(x) = v' + \sum_{j \in [k]} w_j' \cdot x_j$ for some $k \in [i-1], v' \in \mathbb{R}, w' \in \mathbb{R}^k$, we have

$$
\begin{aligned}
x_i \ge b_s(x) &= v' + \sum_{j \in [k]} w_j' \cdot x_j \\
&= v' + \sum_{j \in [k]} \underbrace{(\max(0, w_j'))}_{\ge 0} \cdot x_j + \underbrace{\min(w_j', 0)}_{\le 0} \cdot x_j \\
&\ge v' + \sum_{j \in [k]} (\max(0, w_j') \cdot a_j'^{\le}(x) + \min(w_j', 0) \cdot a_j'^{\ge}(x)) \\
&= b_{s+1}(x).
\end{aligned}
$$

To show $\forall t \in [t'].\, c_t(x) \ge x_i$, we use induction on t. We have $c_1(x) = a_i'^{\ge}(x) \ge x_i$. Assuming $c_t(x) \ge x_i$ and $c_t(x) = v' + \sum_{j \in [k]} w_j' \cdot x_j$ for some $k \in [i-1], v' \in \mathbb{R}, w' \in \mathbb{R}^k$, we have

$$
\begin{aligned}
x_i \le c_t(x) &= v' + \sum_{j \in [k]} w_j' \cdot x_j \\
&= v' + \sum_{j \in [k]} \underbrace{(\max(0, w_j'))}_{\ge 0} \cdot x_j + \underbrace{\min(w_j', 0)}_{\le 0} \cdot x_j \\
&\le v' + \sum_{j \in [k]} (\max(0, w_j') \cdot a_j'^{\ge}(x) + \min(w_j', 0) \cdot a_j'^{\le}(x)) \\
&= c_{t+1}(x).
\end{aligned}
$$

Therefore, $(\forall k \in [i].\, a_k'^{\le}(x) \le x_k \wedge a_k'^{\ge}(x) \ge x_k)$ implies $l_i' \le x_i \le u_i'$. □

4.7 Soundness under Floating Point Arithmetic

The abstract domain and its transformers above are sound under real arithmetic but unsound under floating point arithmetic if one does not take care of the rounding errors. To obtain soundness, let $\mathbb{F}$ be the set of floating point values and $\oplus_f, \ominus_f, \otimes_f, \oslash_f$ be the floating point interval addition, subtraction, multiplication, and division, respectively, as defined in [Miné 2004] with lower bounds rounded towards $-\infty$ and upper bounds rounded towards $+\infty$. For a real constant c, we use $c^-, c^+ \in \mathbb{F}$ to denote the floating point representation of c with rounding towards $-\infty$ and $+\infty$ respectively.

We use the standard interval linear form, where the coefficients in the constraints are intervals instead of scalars, to define an abstract element $a \in \mathcal{A}_n$ over n variables in our domain as a tuple $a = \langle a^{\leq}, a^{\geq}, l, u \rangle$ where for $i \in [n]$:

$$a_i^{\leq}, a_i^{\geq} \in \{x \mapsto [v^-, v^+] \oplus_f \sum_{j \in [i-1]} [w_j^-, w_j^+] \otimes_f x_j \mid v^-, v^+ \in \mathbb{F} \cup \{-\infty, +\infty\}, w^-, w^+ \in \mathbb{F}^{i-1}\}$$

and $l, u \in (\mathbb{F} \cup \{-\infty, +\infty\})^n$. For a floating point interval $[l, u]$, let inf and sup be functions that return its lower and upper bound. The concretization function $\gamma_n \colon \mathcal{A}_n \to \mathcal{P}(\mathbb{F}^n)$ is given by

$$\gamma_n(a) = \{x \in \mathbb{F}^n \mid \forall i \in [n]. \inf(a^{\leq}(x)) \leq x_i \wedge x_i \leq \sup(a^{\geq}(x))\}.$$

We next modify our abstract transformers for soundness under floating point arithmetic. It is straightforward to modify the maxpool transformer so we only show our modifications for the ReLU, sigmoid, tanh, and affine abstract transformers assigning to the variable x_i.

ReLU abstract transformer. It is straightforward to handle the cases $l_j \geq 0$ or $u_j \leq 0$. For the remaining case, we add the following constraints:

$$[\lambda, \lambda] \otimes_f x_j \leq [1, 1] \otimes_f x_i,$$
$$[1, 1] \otimes_f x_i \leq [\psi^-, \psi^+] \otimes_f x_j \oplus_f [\mu^-, \mu^+],$$

where $\lambda \in \{0, 1\}$ and $[\psi^-, \psi^+] = [u_j^-, u_j^+] \oslash_f ([u_j^-, u_j^+] \ominus_f [l_j^-, l_j^+])$, $[\mu^-, \mu^+] = ([-l_j^+, -l_j^-] \otimes_f [u_j^-, u_j^+]) \oslash_f ([u_j^-, u_j^+] \ominus_f [l_j^-, l_j^+])$. Finally, we set $l_i = \lambda \cdot l_j$ and $u_i = u_j$.

Sigmoid and tanh abstract transformers. We consider the case when $l_j < 0$. We soundly compute an interval for the possible values of λ under any rounding mode as $[\lambda^-, \lambda^+] = ([g(u_j)^-, g(u_j)^+] \ominus_f [g(l_j)^-, g(l_j)^+]) \oslash_f ([u_j^-, u_j^+] \ominus_f [l_j^-, l_j^+])$. Similarly, both $g'(l_j)$ and $g'(u_j)$ are soundly abstracted by the intervals $[g'(l_j)^-, g'(l_j)^+]$ and $[g'(u_j)^-, g'(u_j)^+]$, respectively. Because of the limitations of the floating point format, it can happen that the upper polyhedral constraint with slope λ passing through l_j intersects the curve at a point $< u_j$. This happens frequently for smaller perturbations. To ensure soundness, we detect such cases and return the box $[g(l_j)^-, g(u_j)^+]$. Other computations for the transformers can be handled similarly.

Affine abstract transformer. The affine abstract transformer $x_i \leftarrow v + \sum_{j \in [i-1]} w_j \cdot x_j$ for some $w \in \mathbb{F}^{i-1}$ first adds the interval linear constraints $a_i'^{\leq}(x) = a_i'^{\geq}(x) = [v^-, v^+] \oplus_f \sum_{j \in [i-1]} [w_j^-, w_j^+] \otimes_f x_j$.

We modify the backsubstitution for the computation of l_i and u_i. Formally, if we want to obtain l_i', we start with $b_1(x) = a_i'^{\leq}(x)$. If we have $b_s(x) = [v'^-, v'^+] \oplus_f \sum_{j \in [k]} [w_j'^-, w_j'^+] \otimes_f x_j$ for some $k \in [i-1]$, $v'^-, v'^+, w_j'^-, w_j'^+ \in \mathbb{F}$, then

$$b_{s+1}(x) = [v'^-, v'^+] \oplus_f \sum_{j \in [k]} \begin{cases} [w_j'^-, w_j'^+] \otimes_f a_i'^{\leq}(x), & \text{if } w_j'^- \geq 0, \\ [w_j'^-, w_j'^+] \otimes_f a_i'^{\geq}(x), & \text{if } w_j'^+ \leq 0, \\ [\theta_l^-, \theta_l^+], & \text{otherwise.} \end{cases}$$

Here, $[\theta_l^-, \theta_l^+] \in \mathbb{F}$ are the floating point values of the lower bound of the interval $[w_j'^-, w_j'^+] \otimes_f [l_j, u_j]$ rounded towards $-\infty$ and $+\infty$ respectively. We iterate until we reach $b_{s'}$ with $b_{s'}(x) = [v''^-, v''^+]$, i.e., s' is the smallest number with this property. We then set $l_i' = v''^-$. We compute u_i' analogously.

5 REFINEMENT OF ANALYSIS RESULTS

In this section, we show how to apply a form of abstraction refinement based on trace partitioning [Rival and Mauborgne 2007] in order to verify robustness for more complex adversarial regions, which cannot be accurately represented using a set of interval constraints. In particular, we will show how to handle adversarial regions that, in addition to permitting small perturbations to each pixel, allow the adversary to rotate the input image by an angle $\theta \in [\alpha, \beta]$ within an interval.

Algorithm 1 Rotate image I by θ degrees.

---

procedure ROTATE(I, θ)
    Input: $I \in [0, 1]^{m \times n}, \theta \in [-\pi, \pi]$,   **Output:** $R \in [0, 1]^{m \times n}$
    for $i \in \{1, \ldots, m\}, j \in \{1, \ldots, n\}$ **do**
        $(x, y) \leftarrow (j - (n + 1)/2, (m + 1)/2 - i)$
        $(x', y') \leftarrow (\cos(-\theta) \cdot x - \sin(-\theta) \cdot y, \sin(-\theta) \cdot x + \cos(-\theta) \cdot y)$
        $(i'_{\text{low}}, i'_{\text{high}}) \leftarrow (\max(1, \lfloor (m + 1)/2 - y' \rfloor), \min(m, \lceil (m + 1)/2 - y' \rceil))$
        $(j'_{\text{low}}, j'_{\text{high}}) \leftarrow (\max(1, \lfloor x' + (n + 1)/2 \rfloor), \min(n, \lceil x' + (n + 1)/2 \rceil))$
        $t \leftarrow \sum_{i'=i'_{\text{low}}}^{i'_{\text{high}}} \sum_{j'=j'_{\text{low}}}^{j'_{\text{high}}} \max(0, 1 - \sqrt{(j' - x')^2 + (i' - y')^2})$
        if $t \neq 0$ **then**
            $R_{i,j} \leftarrow (1/t) \cdot \sum_{i'=i'_{\text{low}}}^{i'_{\text{high}}} \sum_{j'=j'_{\text{low}}}^{j'_{\text{high}}} \max(0, 1 - \sqrt{(j' - x')^2 + (i' - y')^2}) \cdot I_{i',j'}$
        else
            $R_{i,j} \leftarrow 0$
        end if
    end for
    return R
end procedure

Verifying robustness against image rotations. Consider Algorithm 1, which rotates an $m \times n$-pixel (grayscale) image by an angle θ. To compute the intensity $R_{i,j}$ of a given output pixel, it first computes the (real-valued) position (x', y') that would be mapped to the position of the center of the pixel. Then, it performs linear interpolation: it forms a convex combination of pixels in the neighborhood of (x', y'), such that the contribution of each pixel is proportional to the distance to (x', y'), cutting off contributions at distance 1.

Our goal is to verify that a neural network $N \colon \mathbb{R}^{m \times n} \to \mathbb{R}^r$ classifies all images obtained by rotating an input image using Algorithm 1 with an angle $\theta \in [\alpha, \beta] \subseteq [-\pi, \pi]$ in the same way. More generally, if we have an adversarial region $X \subseteq \mathbb{R}^{m \times n}$ (represented using componentwise interval constraints), we would like to verify that for any image $I \in X$ and any angle $\theta \in [\alpha, \beta]$, the neural network N classifies ROTATE(I, θ) to a given class k. This induces a new adversarial region $X' = \{\text{ROTATE}(I, \theta) \mid I \in X, \theta \in [\alpha, \beta]\}$. Note that because we deal with regions (and not only concrete images) as well as rotations that employ linear interpolation, we cannot simply enumerate all possible rotations as done for simpler rotation algorithms and concrete images [Pei et al. 2017b].

Interval specification of X'. We verify robustness against rotations by deriving lower and upper bounds on the intensities of all pixels of the rotated image. We then verify that the neural network classifies all images satisfying those bounds to class k. To obtain bounds, we apply abstract interpretation to Algorithm 1, using the interval domain (more powerful numerical domains could be applied). We use standard interval domain transformers, except to derive bounds on t and $R_{i,j}$, which we compute (at the same time), by enumerating all possible integer values of $i'_{\text{low}}, i'_{\text{high}}, j'_{\text{low}}$ and j'_{high} (respecting the constraints $i'_{\text{low}} + 1 \geq i'_{\text{high}}$ and $j'_{\text{low}} + 1 \geq j'_{\text{high}}$, and refining the intervals for x' and y' based on the known values i'_{low} and j'_{low}) and joining the intervals resulting from each case. For each case, we compute intervals for $R_{i,j}$ in two ways: once using interval arithmetic, restricting partial sums to the interval $[0, 1]$, and once by observing that a convex combination of pixel values will be contained in the union of intervals for the individual values. We intersect the intervals resulting from both approaches.

Refinement of abstract inputs by trace partitioning. For large enough intervals $[\alpha, \beta]$, the derived bounds often become too imprecise. Thus, when our analyzer is invoked with these bounds, it may fail to verify the property, even though it actually holds. We can make the following simple observation: if we have n sets $X'_1, \ldots, X'_n$ that cover the adversarial region X', i.e. $X' \subseteq \bigcup_{i=1}^{n} X'_i$, then it suffices to verify that the neural network N classifies all input images to class k for each individual input region X'_i for $i \in \{1, \ldots, n\}$. We obtain $X'_1, \ldots, X'_n$ by subdividing the interval $[\alpha, \beta]$ into n equal parts: $\{\text{Rotate}(I, \theta) \mid I \in X, \theta \in [(i-1)/n \cdot (\beta - \alpha) + \alpha, i/n \cdot (\beta - \alpha) + \alpha]\} \subseteq X'_i$. Note that each X'_i is obtained by running the interval analysis on the rotation code with the given angle interval and the adversarial region X. After obtaining all X'_i's, we run our neural network analyzer separately with each X'_i as input.

Batching. As interval analysis tends to be imprecise for large input intervals, we usually need to subdivide the interval $[\alpha, \beta]$ into many parts to obtain precise enough output intervals from the interval analysis (a form of trace partitioning [Rival and Mauborgne 2007]). Running our neural network analysis for each of these can be too expensive. Instead, we use a separate refinement step to obtain more precise interval bounds for larger input intervals. We further subdivide each of the n intervals into m parts each, for a total of $n \cdot m$ intervals in n batches. For each of the n batches, we then run interval analysis m times, once for each part, and combine the results using a join, i.e., we compute the smallest common bounding box of all output regions in a batch. The additional refinement within each batch preserves dependencies between variables that a plain interval analysis would ignore, and thus yields more precise boxes $X'_1, \ldots, X'_n$, on which we run the neural network analysis.

Using the approach outlined above, we were able to verify, for the first time, that the neural network is robust to non-trivial rotations of all images inside an adversarial region.

6 EXPERIMENTAL EVALUATION

In this section we evaluate the effectiveness of our approach for verifying the robustness properties of large, challenging, and diverse set of neural networks. We implemented our method in an analyzer called *DeepPoly*. The analyzer is written in Python and the abstract transformers of our domain are implemented on top of the ELINA library [Singh et al. 2017, 2018b] for numerical abstractions. We have implemented both a sequential and a parallel version of our transformers. All code, networks, datasets, and results used in our evaluation are available at http://safeai.ethz.ch. We compared the precision and performance of *DeepPoly* against the three state-of-the-art systems that can scale to larger networks:

- AI^2 by Gehr et al. [2018] uses the Zonotope abstract domain [Ghorbal et al. 2009] implemented in ELINA for performing abstract interpretation of feedforward and convolutional ReLU networks. Their transformers are generic and do not exploit the structure of ReLU. As a result, AI^2 is often slow and imprecise.
- Fast-Lin by Weng et al. [2018] performs layerwise linear approximations tailored to exploit the structure of ReLU feedforward networks. We note that Fast-Lin is not sound under floating point arithmetic and does not support convolutional networks. Nonetheless, we still compare to it despite the fact it may contain false negatives (adapting their method to be sound in floating point arithmetic is non-trivial).
- *DeepZ* by Singh et al. [2018a] provides specialized Zonotope transformers for handling ReLU, sigmoid, and tanh activations, and supports both feedforward and convolutional networks. It is worth mentioning that although Fast-Lin and *DeepZ* employ very different techniques for robustness analysis, both can be shown to have the same precision on feedforward neural networks with ReLU activations. On our benchmarks, *DeepZ* was often faster than Fast-Lin.

Table 1. Neural network architectures used in our experiments.

Dataset	Model	Type	#Hidden units	#Hidden layers
MNIST	FFNNSmall	fully connected	610	6
	FFNNMed	fully connected	1 810	9
	FFNNBig	fully connected	4 106	4
	FFNNSigmoid	fully connected	3 010	6
	FFNNTanh	fully connected	3 010	6
	ConvSmall	convolutional	3 604	3
	ConvBig	convolutional	34 688	6
	ConvSuper	convolutional	88 500	6
CIFAR10	FFNNSmall	fully connected	6 10	6
	FFNNMed	fully connected	1 810	9
	FFNNBig	fully connected	7 178	7
	ConvSmall	convolutional	4 852	3
	ConvBig	convolutional	62 464	6

Our experimental results indicate that *DeepPoly* is always more precise than all three competing tools on our benchmarks while maintaining scalability. This demonstrates the suitability of *DeepPoly* for the task of robustness verification of larger neural networks.

6.1 Experimental Setup

All of our experiments for the feedforward networks were run on a 3.3 GHz 10 core Intel i9-7900X Skylake CPU with a main memory of 64 GB; our experiments for the convolutional networks were run on a 2.6 GHz 14 core Intel Xeon CPU E5-2690 with 512 GB of main memory. We next describe our experimental setup including the datasets, neural networks, and robustness properties.

Evaluation datasets. We used the popular MNIST [Lecun et al. 1998] and CIFAR10 [Krizhevsky 2009] image datasets for our experiments. MNIST contains grayscale images of size 28×28 pixels and CIFAR10 consists of RGB images of size 32×32 pixels. For our evaluation, we chose the first 100 images from the test set of each dataset. For the task of robustness certification, out of these 100 images, we considered only those that were correctly classified by the neural network.

Neural networks. Table 1 shows the MNIST and the CIFAR10 neural network architectures used in our experiments. The architectures considered in our evaluation contain up to 88K hidden units. We use networks trained with adversarial training, i.e., defended against adversarial attacks, as well as undefended networks. We used DiffAI by Mirman et al. [2018] and projected gradient descent (PGD) from Dong et al. [2018] for adversarial training. In our evaluation, when we consider the certified robustness of the defended and undefended networks with the same architecture together, we append the suffix *Point* to the name of a neural network trained without adversarial training and the name of the training procedure (either DiffAI or PGD) to the name of a defended network. In the table, the FFNNSigmoid and FFNNTanh networks use sigmoid and tanh activations, respectively. All other networks use ReLU activations. The FFNNSmall and FFNNMed network architectures for both MNIST and CIFAR10 datasets were taken from Gehr et al. [2018] whereas the FFNNBig architectures were taken from Weng et al. [2018]. The ConvSmall, ConvBig, and ConvSuper architectures were taken from Mirman et al. [2018].

Robustness properties. We consider the following robustness properties:

(1) L_∞-norm [Carlini and Wagner 2017]: This attack is parameterized by a constant ϵ. The adversarial region contains all perturbed images $\overline{x}'$ where each pixel $\overline{x}'_i$ has a distance of at most ϵ from the corresponding pixel $\overline{x}_i$ in the original input $\overline{x}$. We use different values of ϵ in our experiments. In general, we use smaller ϵ values for the CIFAR10 dataset compared to the MNIST dataset since the CIFAR10 networks are known to be less robust against L_∞-norm attacks for larger ϵ values [Weng et al. 2018].

(2) Rotation: The input image is first perturbed using a perturbation bounded by ϵ in the L_∞-norm. The resulting image is then rotated by Algorithm 1 using an arbitrary $\theta \in [\alpha, \beta]$. The region $\mathcal{R}_{\overline{x}, \epsilon, [\alpha, \beta]}$ contains all images that can be obtained in this way.
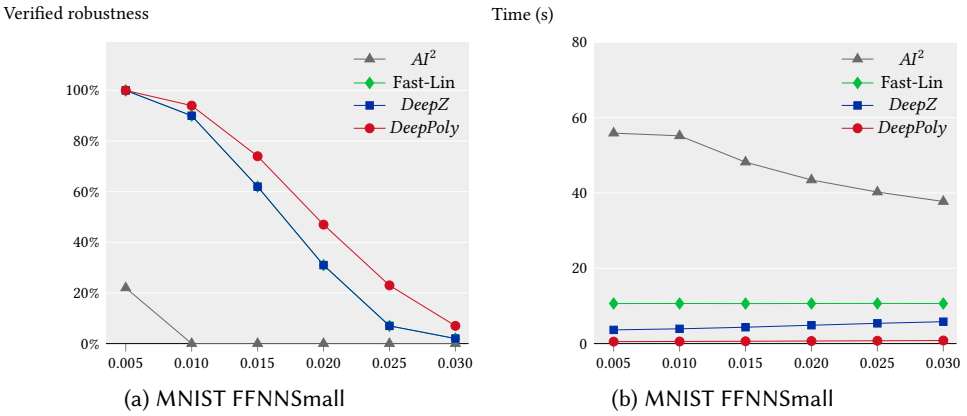
Fig. 5. Verified robustness and runtime for L_∞-norm perturbations by *DeepPoly* against AI^2, Fast-Lin, and *DeepZ* on the MNIST FFNNSmall. *DeepZ* and Fast-Lin are equivalent in robustness.

6.2 L_∞-Norm Perturbation

We first compare the precision and performance of *DeepPoly* vs AI^2, Fast-Lin, and *DeepZ* for robustness certification against L_∞-norm based adversarial attacks on the MNIST FFNNSmall network. We note that it is straightforward to parallelize Fast-Lin, *DeepZ*, and *DeepPoly*. However, the abstract transformers in AI^2 cannot be efficiently parallelized. To ensure fairness, we ran all four analyzers in single threaded mode. Fig. 5 compares the percent of robustness properties proved and the average runtime per ϵ-value of all four analyzers. We used six different values for ϵ shown on the x-axis. For all analyzers, the number of proved properties decreases with increasing values of ϵ. As can be seen, *DeepPoly* is the fastest and the most precise analyzer on the FFNNSmall network. *DeepZ* has the exact same precision as Fast-Lin but is up to 2.5x faster. AI^2 has significantly worse precision and higher runtime than all other analyzers.

For all of our remaining experiments, we compare the precision and performance of the parallelized versions of *DeepPoly* and *DeepZ*.

MNIST fully connected feedforward networks. Fig. 6 compares the percentage of verified robustness properties and the average runtime of *DeepPoly* against *DeepZ* on the MNIST FFNNMed and FFNNBig neural networks. Both networks were trained without adversarial training. *DeepPoly* proves more properties than *DeepZ* on both networks. As an example, considering the verified
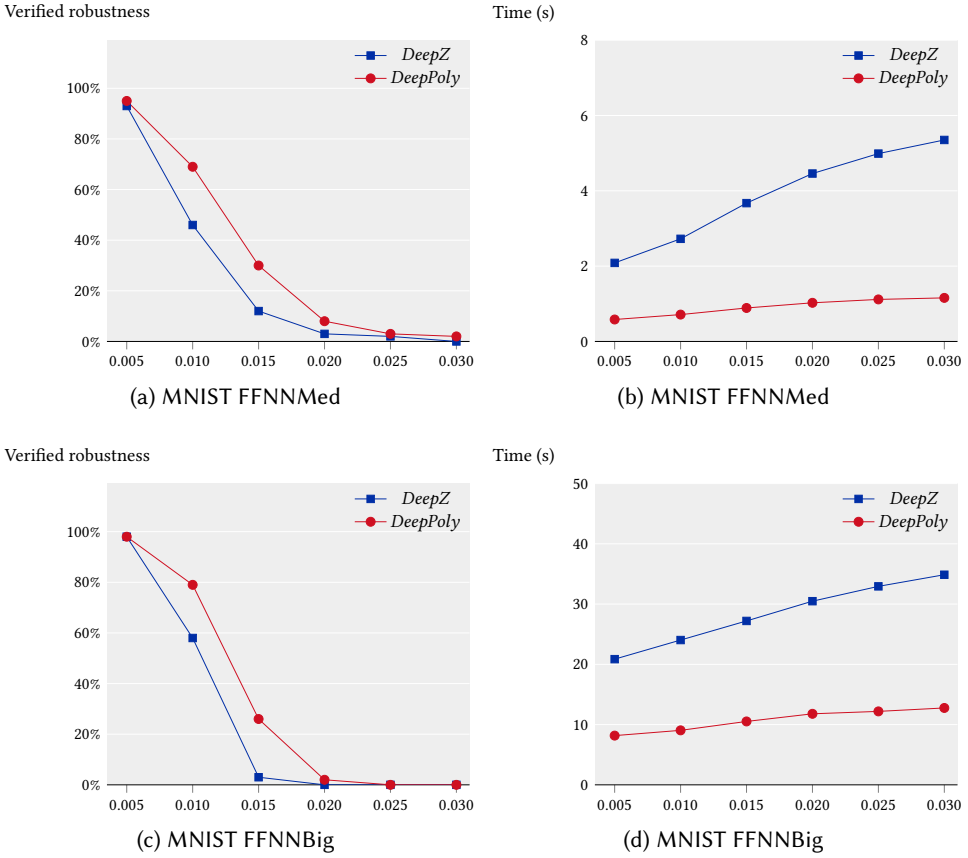
(a) MNIST FFNNMed

(b) MNIST FFNNMed

(c) MNIST FFNNBig

(d) MNIST FFNNBig

Fig. 6. Verified robustness and runtime for L_∞-norm perturbations by *DeepPoly* vs. *DeepZ* on the MNIST FFNNMed and FFNNBig networks.

robustness for $\epsilon = 0.01$, we notice that *DeepPoly* proves 69% of properties on the FFNNMed network whereas *DeepZ* proves 46%. The corresponding numbers on the FFNNBig network are 79% and 58% respectively. *DeepPoly* is also significantly faster than *DeepZ* on both networks achieving a speedup of up to 4x and 2.5x on the FFNNMed and FFNNBig networks.

We compare the average percentage of the number of hidden units that can take both positive and negative values per ϵ-value for the MNIST FFNNSmall and FFNNMed neural networks in Fig. 7. Since the ReLU transformer in both *DeepPoly* and *DeepZ* is inexact for such hidden units, it is important to reduce their percentage. For both networks, *DeepPoly* produces strictly less hidden units for which the ReLU transformer is inexact than *DeepZ*.

In Fig. 8, we compare the precision of *DeepPoly* and *DeepZ* on the MNIST FFNNSigmoid and FFNNTanh networks. Both networks were trained using PGD-based adversarial training. On both networks, *DeepPoly* is strictly more precise than *DeepZ*. For the FFNNSigmoid network, there is a sharp decline in the number of proved properties by *DeepZ* starting at $\epsilon = 0.02$. *DeepZ* proves only 23% of the properties when $\epsilon = 0.03$; in contrast, *DeepPoly* proves 80%. Similarly, for the FFNNTanh network, *DeepZ* only proves 1% of properties when $\epsilon = 0.015$, whereas *DeepPoly* proves 94%. We also note that *DeepPoly* is more than 2x faster than *DeepZ* on both these networks (we omit the

% hidden units

(a) MNIST FFNNSmall

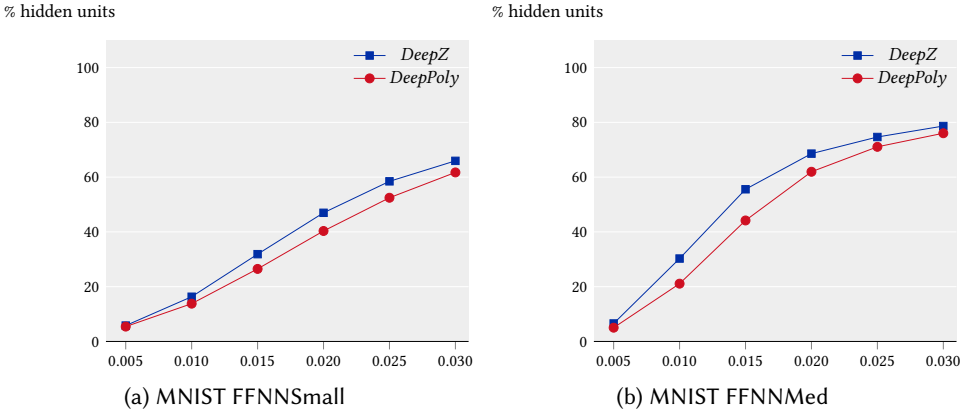% hidden units

(b) MNIST FFNNMed

Fig. 7. Average percentage of hidden units that can take both positive and negative values for *DeepPoly* vs. *DeepZ* on the MNIST FFNNSmall and FFNNMed networks.
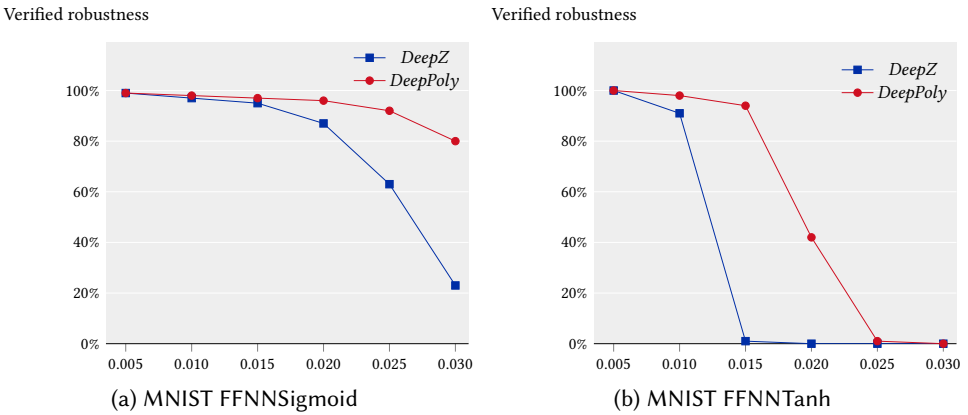
Verified robustness

(a) MNIST FFNNSigmoid

Verified robustness

(b) MNIST FFNNTanh

Fig. 8. Verified robustness and runtime for L_∞-norm perturbations by *DeepPoly* vs. *DeepZ* on the MNIST FFNNSigmoid and FFNNTanh networks.

relevant plots here as timings do not change with increasing values of ϵ): *DeepZ* has an average runtime of ≤ 35 seconds on both networks whereas *DeepPoly* has an average runtime of ≤ 15 seconds on both.

MNIST convolutional networks. Fig. 9 compares the precision and the average runtime of *DeepPoly* vs *DeepZ* on the MNIST ConvSmall networks. We consider three types of ConvSmall networks based on their training method: (a) undefended (Point), (b) defended with PGD (PGD), and (c) defended with DiffAI (DiffAI). Note that our convolutional networks are more robust than the feedforward networks and thus the values of ϵ considered in our experiments are higher than those for feedforward networks.

As expected, both *DeepPoly* and *DeepZ* prove more properties on the defended networks than on the undefended one. We notice that the ConvSmall network trained with DiffAI is provably more
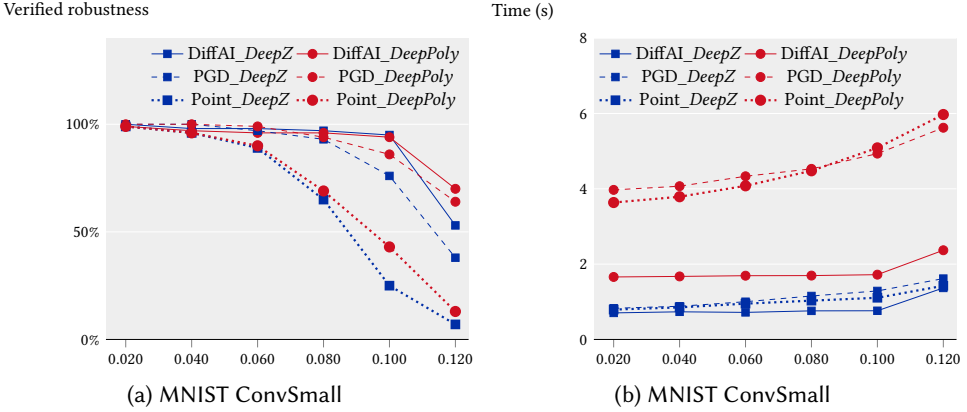
Fig. 9. Verified robustness and runtime for L_∞-norm perturbations by *DeepPoly* vs. *DeepZ* on the MNIST ConvSmall networks.

Table 2. Verified robustness by *DeepZ* and *DeepPoly* on the large convolutional networks trained with DiffAI.

Dataset	Model	ϵ	% Verified robustness		Average runtime	
			DeepZ	*DeepPoly*	*DeepZ*	*DeepPoly*
MNIST	ConvBig	0.1	97	97	5	50
	ConvBig	0.2	79	78	7	61
	ConvBig	0.3	37	43	17	88
	ConvSuper	0.1	97	97	133	400
CIFAR10	ConvBig	0.006	50	52	39	322
	ConvBig	0.008	33	40	46	331

robust. Overall, *DeepPoly* proves more properties than *DeepZ* on all neural networks. The difference between the number of properties proved by *DeepPoly* and *DeepZ* is higher for larger values of ϵ. It is interesting to note that on the DiffAI defended network, *DeepZ* proves slightly more properties than *DeepPoly* for $\epsilon \leq 0.10$. However, for $\epsilon = 0.12$, the percentage of properties proved by *DeepZ* drops to 53% whereas *DeepPoly* proves 70% of the robustness properties. We notice that *DeepPoly* is slower than *DeepZ* on all ConvSmall networks. This is due to the fact that the assigned expressions during affine transforms in the convolutional layers are sparse. The Zonotope representation in *DeepZ* allows the corresponding transformers to utilize this sparsity better than our domain. We also note that *DeepPoly* is up to 2x faster on the DiffAI network as compared to the other two networks.

Table 2 shows our experimental results on the larger MNIST convolutional networks trained using DiffAI. For the ConvBig network, *DeepPoly* proves significantly more than *DeepZ* for $\epsilon = 0.3$. For the ConvSuper network, *DeepPoly* has the same precision as *DeepZ* for $\epsilon = 0.1$ and proves 97% of robustness properties. On both these networks, *DeepPoly* is slower than *DeepZ*.

CIFAR10 feedforward networks. Fig. 10 compares *DeepPoly* against *DeepZ* on the CIFAR10 feed-forward networks. As with the MNIST feedforward networks, *DeepPoly* verifies more properties
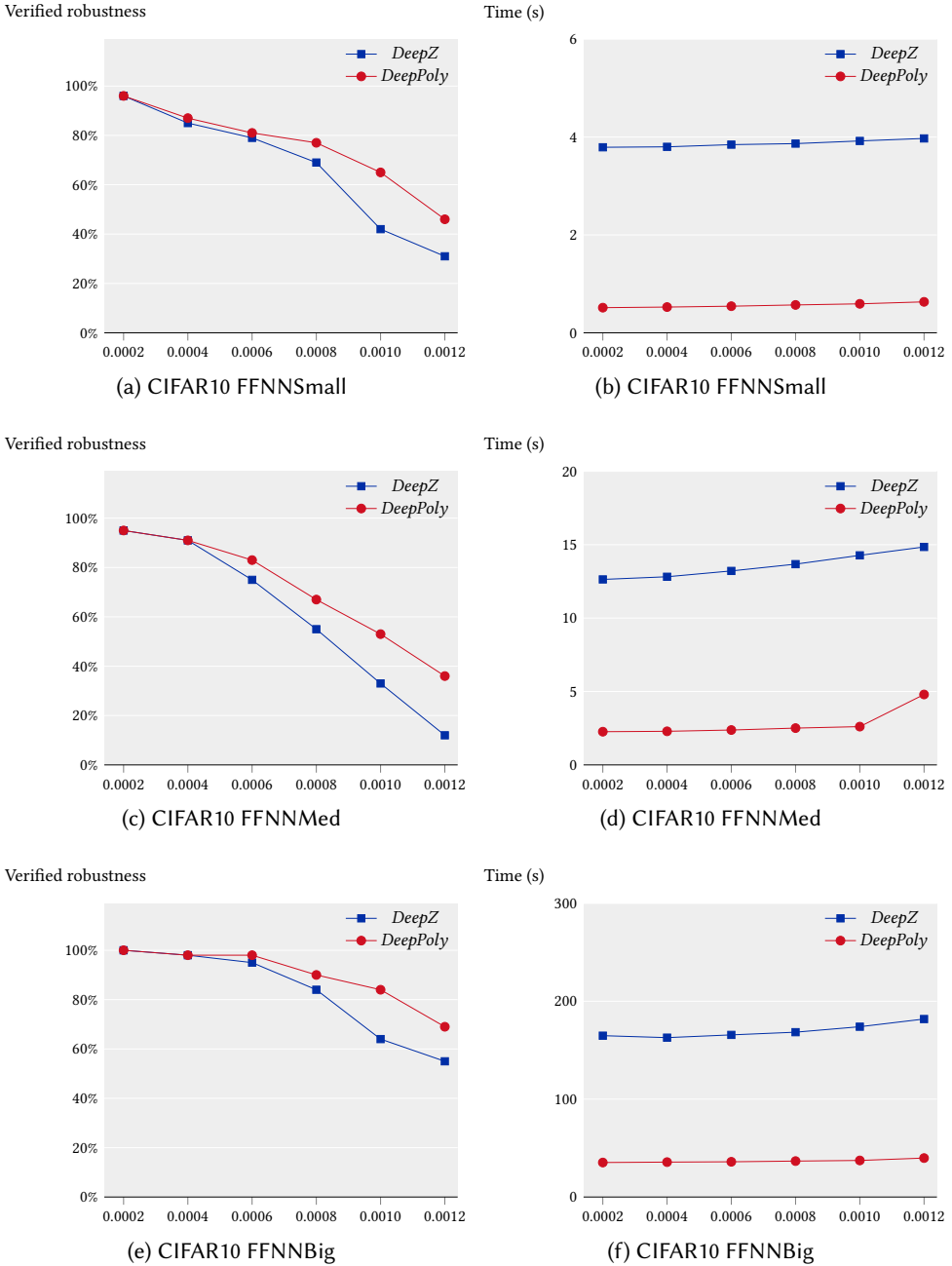
Fig. 10. Verified robustness and runtime for L_∞-norm perturbations by *DeepPoly* vs. *DeepZ* on the CIFAR10 fully connected feedforward networks.

than *DeepZ* and is faster on all the considered networks. Considering $\epsilon = 0.001$, *DeepPoly* proves 65%, 53%, and 84% of properties on the FFNNSmall, FFNNMed, and FFNNBig networks respectively

whereas *DeepZ* proves 42%, 33%, and 64% of properties. Notice that the average runtime of both *DeepPoly* and *DeepZ* on the CIFAR10 FFNNMed is higher than on the MNIST FFNNMed network even though the number of hidden units is the same. The slowdown on the CIFAR10 networks is due to the higher number of input pixels. *DeepPoly* is up to 7x, 5x, and 4.5x faster than *DeepZ* on the FFNNSmall, FFNNMed, and FFNNBig networks, respectively.

CIFAR10 convolutional networks. Fig. 11 evaluates *DeepPoly* and *DeepZ* on the CIFAR10 ConvSmall networks. We again consider undefended (Point) networks and networks defended with PGD and DiffAI. We again notice that the ConvSmall network trained with DiffAI is the most provably robust network. *DeepPoly* overall proves more properties than *DeepZ* on all networks. As with the MNIST ConvSmall network defended with DiffAI, *DeepZ* is slightly more precise than *DeepPoly* for $\epsilon = 0.008$; however, *DeepPoly* proves more properties for $\epsilon = 0.012$. *DeepPoly* is slower than *DeepZ* on all the considered ConvSmall networks.

The last two rows in Table 2 compare the precision and performance of *DeepPoly* and *DeepZ* on the CIFAR10 ConvBig convolutional network trained with DiffAI. It can be seen that *DeepPoly* proves more properties than *DeepZ* for both $\epsilon = 0.006$ and $\epsilon = 0.008$.
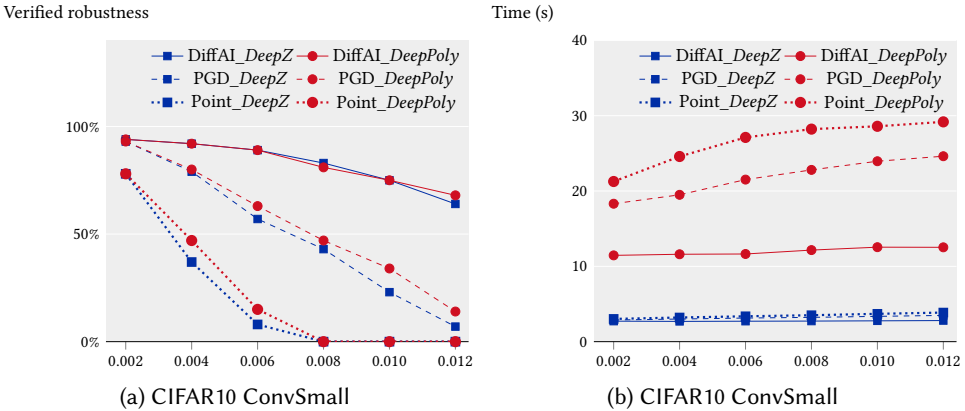

Fig. 11. Verified robustness and runtime for L_∞-norm perturbations by *DeepPoly* vs. *DeepZ* on the CIFAR10 ConvSmall networks.

6.3 Rotation Perturbation

As described in earlier sections, we can apply refinement to the input so to prove a neural network robust against rotations of a certain input image. Specifically, our analysis can prove that the MNIST FFNNSmall network classifies a given image of the digit 3 correctly, even if each pixel is first L_∞-perturbed with $\epsilon \leq 0.001$ and then rotated using an arbitrary angle θ between -45 and 65 degrees. Fig. 12 shows example regions and analysis times for a number of choices of parameters to the refinement approach. For example, #Batches = 220, Batch Size = 300 means that we split the interval $[\alpha, \beta]$ into $n = 220$ batches. To analyze a batch, we split the corresponding interval into $m = 300$ input intervals for interval analysis, resulting in 300 regions for each batch. We then run *DeepPoly* on the smallest common bounding boxes of all regions in each batch, 220 times in total. Fig. 12 shows a few such bounding boxes in the Regions column. Note that it is not sufficient to compute a single region that captures all rotated images. Fig. 12 shows two such attempts: one where we did not use batching (therefore, our interval analysis approach was applied
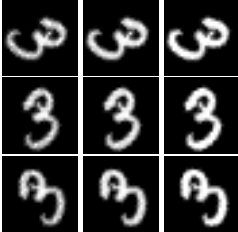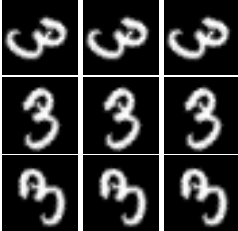
#Batches	Batch Size	Region(s) $(l, \frac{1}{2}(l+u), u)$	Analysis time	Verified?
1	1		0.5s + 1.9s	No
1	10000		22.2s + 1.8s	No
220	1		1.2s + 5m51s	No
220	300		2m29s + 5m30s	Yes

Fig. 12. Results for robustness against rotations with the MNIST FFNNSmall network. Each row shows a different attempt to prove that the given image of the digit 3 can be perturbed within an L_∞ ball of radius $\epsilon = 0.001$ and rotated by an arbitrary angle θ between -45 to 65 degrees without changing its classification. For the second two attempts, we show 4 representative combined regions (out of 220, one per batch). The running time is split into two components: (i) the time used for interval analysis on the rotation algorithm and (ii), the time used to prove the neural network robust with all of the computed bounding boxes (using *DeepPoly*).

to the rotation algorithm using an abstract θ covering the entire range), and one where we used a batch size of 10,000 to compute the bounding box of the perturbations rather precisely. However, those perturbations cannot be captured well using interval constraints, therefore the bounding box contains many spurious inputs and the verification fails.

We then considered two verification attempts with 220 batches, with each batch covering a range of θ of length 0.5 degrees. It was not sufficient to use a batch size of 1, as some input intervals become large. Using a batch size of 300, the neural network can be proved robust for this perturbation.

7 RELATED WORK

We already extensively discussed the works that are most closely related throughout the paper, here we additionally elaborate on several others.

Generating adversarial examples. There is considerable interest in constructing examples that make the neural network misclassify an input. Nguyen et al. [2015] find adversarial examples without starting from a test point, Tabacof and Valle [2016] use random perturbations for generating adversarial examples, Sabour et al. [2015] demonstrate non-robustness of intermediate layers, and

Grosse et al. [2016] generate adversarial examples for malware classification. Pei et al. [2017a] systematically generate adversarial examples covering all neurons in the network. Bastani et al. [2016] under-approximate the behavior of the network under L_∞-norm based perturbation and formally define metrics of adversarial frequency and adversarial severity to evaluate the robustness of a neural network against adversarial attack.

Formal verification of neural network robustness. Existing formal verifiers of neural network robustness can be broadly classified as either *complete* or *incomplete*. Complete verifiers do not have false positives but have limited scalability and cannot handle neural networks containing more than a few hundred hidden units whereas incomplete verifiers approximate for better scalability. Complete verifiers are based on SMT solving [Ehlers 2017; Katz et al. 2017], mixed integer linear programming [Tjeng and Tedrake 2017] or input refinement [Wang et al. 2018] whereas existing incomplete verifiers are based on duality [Dvijotham et al. 2018; Raghunathan et al. 2018], abstract interpretation [Gehr et al. 2018; Singh et al. 2018a], and linear approximations [Weng et al. 2018; Wong and Kolter 2018]. We note that although our verifier is designed to be incomplete for better scalability, it can be made complete by refining the input iteratively.

Adversarial training. There is growing interest in *adversarial training* where neural networks are trained against a model of adversarial attacks. Gu and Rigazio [2014] add Gaussian noise to the training set and remove it statistically for defending against adversarial examples. The approach of Goodfellow et al. [2015] first generates adversarial examples misclassified by neural networks and then designs a defense against this attack by explicitly training against perturbations generated by the attack. Madry et al. [2018] shows that training against an optimal attack also guards against non-optimal attacks. While this was effective in experiments, Carlini et al. [2017] demonstrated an attack for the safety-critical problem of ground-truthing, where this defense occasionally exacerbated the problem. Mirman et al. [2018] train neural networks against adversarial attacks using abstract transformers for the Zonotope domain. As mentioned earlier, our abstract transformers can be plugged into such a framework to potentially improve the training results.

8 CONCLUSION

We introduced a new method for certifying deep neural networks which balances analysis precision and scalability. The core idea is an abstract domain based on floating point polyhedra and intervals equipped with abstract transformers specifically designed for common neural network functions such as affine transforms, ReLU, sigmoid, tanh, and maxpool. These abstract transformers enable us to soundly handle both, feed-forward and convolutional networks.

We implemented our method in an analyzer, called *DeepPoly*, and evaluated it extensively on a wide range of networks of different sizes including defended and undefended networks. Our experimental results demonstrate that *DeepPoly* is more precise than prior work yet can handle large networks.

We also showed how to use *DeepPoly* to prove, for the first time, the robustness of a neural network when the input image is perturbed by complex transformations such as rotations employing linear interpolation.

We believe this work is a promising step towards more effective reasoning about deep neural networks and a useful building block for proving interesting specifications as well as other applications of analysis (for example, training more robust networks).

ACKNOWLEDGMENTS

REFERENCES

Filippo Amato, Alberto López, Eladia María Peña-Méndez, Petr Vaňhara, Aleš Hampl, and Josef Havel. 2013. Artificial neural networks in medical diagnosis. *Journal of Applied Biomedicine* 11, 2 (2013), 47 – 58.

Osbert Bastani, Yani Ioannou, Leonidas Lampropoulos, Dimitrios Vytiniotis, Aditya V. Nori, and Antonio Criminisi. 2016. Measuring Neural Net Robustness with Constraints. In *Proc. Neural Information Processing Systems (NIPS)*. 2621–2629.

Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. 2016. End to End Learning for Self-Driving Cars. *CoRR* abs/1604.07316 (2016).

Nicholas Carlini, Guy Katz, Clark Barrett, and David L. Dill. 2017. Ground-Truth Adversarial Examples. *CoRR* abs/1709.10207 (2017).

Nicholas Carlini and David A. Wagner. 2017. Towards Evaluating the Robustness of Neural Networks. In *Proc. IEEE Symposium on Security and Privacy (SP)*. 39–57.

Patrick Cousot and Nicolas Halbwachs. 1978. Automatic Discovery of Linear Restraints Among Variables of a Program. In *Proc. Principles of Programming Languages (POPL)*. 84–96.

Yinpeng Dong, Fangzhou Liao, Tianyu Pang, Hang Su, Jun Zhu, Xiaolin Hu, and Jianguo Li. 2018. Boosting adversarial attacks with momentum. In *Proc. Computer Vision and Pattern Recognition (CVPR)*.

Krishnamurthy Dvijotham, Robert Stanforth, Sven Gowal, Timothy Mann, and Pushmeet Kohli. 2018. A Dual Approach to Scalable Verification of Deep Networks. In *Proc. Uncertainty in Artificial Intelligence (UAI)*. 162–171.

Rüdiger Ehlers. 2017. Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks. In *Proc. Automated Technology for Verification and Analysis (ATVA)*. 269–286.

Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. 2018. AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation. In *Proc. IEEE Symposium on Security and Privacy (SP)*, Vol. 00. 948–963.

Khalil Ghorbal, Eric Goubault, and Sylvie Putot. 2009. The Zonotope Abstract Domain Taylor1+. In *Proc. Computer Aided Verification (CAV)*. 627–633.

Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. In *Proc. International Conference on Learning Representations (ICLR)*.

Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick D. McDaniel. 2016. Adversarial Perturbations Against Deep Neural Networks for Malware Classification. *CoRR* abs/1606.04435 (2016). http://arxiv.org/abs/1606.04435

Shixiang Gu and Luca Rigazio. 2014. Towards deep neural network architectures robust to adversarial examples. *arXiv preprint arXiv:1412.5068* (2014).

Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. 2017. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In *Proc. International Conference on Computer Aided Verification (CAV)*. 97–117.

Alex Krizhevsky. 2009. *Learning multiple layers of features from tiny images*. Technical Report.

Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. In *Proc. of the IEEE*. 2278–2324.

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

Antoine Miné. 2004. Relational Abstract Domains for the Detection of Floating-Point Run-Time Errors. In *Proc. European Symposium on Programming (ESOP)*. 3–17.

Matthew Mirman, Timon Gehr, and Martin Vechev. 2018. Differentiable Abstract Interpretation for Provably Robust Neural Networks. In *Proc. International Conference on Machine Learning (ICML)*. 3575–3583.

Anh Mai Nguyen, Jason Yosinski, and Jeff Clune. 2015. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proc. IEEE Computer Vision and Pattern Recognition (CVPR)*. 427–436.

Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017a. DeepXplore: Automated Whitebox Testing of Deep Learning Systems. In *Proc. Symposium on Operating Systems Principles (SOSP)*. 1–18.

Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017b. Towards Practical Verification of Machine Learning: The Case of Computer Vision Systems. *CoRR* abs/1712.01785 (2017).

Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. 2018. Certified Defenses against Adversarial Examples. In *Proc. International Conference on Machine Learning (ICML)*.

Xavier Rival and Laurent Mauborgne. 2007. The Trace Partitioning Abstract Domain. *ACM Trans. Program. Lang. Syst.* 29, 5 (2007).

Sara Sabour, Yanshuai Cao, Fartash Faghri, and David J. Fleet. 2015. Adversarial Manipulation of Deep Representations. *CoRR* abs/1511.05122 (2015).

Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. 2018a. Fast and Effective Robustness Certification. In *Proc. Neural Information Processing Systems (NIPS)*.

Gagandeep Singh, Markus Püschel, and Martin Vechev. 2017. Fast Polyhedra Abstract Domain. In *Proc. Principles of Programming Languages (POPL)*. 46–59.

Gagandeep Singh, Markus Püschel, and Martin Vechev. 2018b. A Practical Construction for Decomposing Numerical Abstract Domains. *Proc. ACM Program. Lang.* 2, POPL (2018), 55:1–55:28.

Pedro Tabacof and Eduardo Valle. 2016. Exploring the space of adversarial images. In *Proc. International Joint Conference on Neural Networks (IJCNN)*. 426–433.

Vincent Tjeng and Russ Tedrake. 2017. Verifying Neural Networks with Mixed Integer Programming. *CoRR* abs/1711.07356 (2017).

Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. 2018. Formal Security Analysis of Neural Networks using Symbolic Intervals. In *Proc. USENIX Security Symposium (USENIX Security 18)*. 1599–1614.

Tsui-Wei Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Luca Daniel, Duane Boning, and Inderjit Dhillon. 2018. Towards Fast Computation of Certified Robustness for ReLU Networks. In *Proc. International Conference on Machine Learning (ICML)*. 5273–5282.

Eric Wong and Zico Kolter. 2018. Provable Defenses against Adversarial Examples via the Convex Outer Adversarial Polytope. In *Proc. International Conference on Machine Learning (ICML)*. 5283–5292.