ROBUSTNESS CERTIFICATION WITH REFINEMENT

Anonymous authors
Paper under double-blind review

ABSTRACT

We present a novel approach for verification of neural networks which combines scalable over-approximation methods with precise (mixed integer) linear programming. This results in significantly better precision than state of the art verifiers on feed forward neural networks with piecewise linear activation functions.

1 INTRODUCTION

Neural networks are increasingly applied in critical domains such as autonomous driving Bojarski et al. (2016), medical diagnosis Amato et al. (2013), and speech recognition Hinton et al. (2012). However, it has been shown by Goodfellow et al. (2014) that neural networks can be non-robust against *adversarial* attacks, i.e., small input perturbations cause neural networks to misclassify. To address this challenge and prove a network is free of adversarial examples (usually, in a region around a given input), recent work has started investigating the use of verification techniques. Current verifiers can be broadly classified in complete or incomplete ones.

Complete verifiers are exact, i.e., if the verifier fails to check a property then the network is non-robust and vice-versa. Existing complete verifiers are based on Mixed Integer Linear Programming (MILP) Lomuscio & Maganti (2017); Fischetti & Jo (2018); Dutta et al. (2018); Cheng et al. (2017); Tjeng & Tedrake (2017), SMT Katz et al. (2017); Ehlers (2017), or input refinement Wang et al. (2018). Although precise, these verifies have limited scalability and can only handle networks with a small number of layers and neurons. To scale, incomplete verifiers usually employ over-approximation methods and hence they are sound (no false negatives) but may fail to verify a robust network. Incomplete verifiers are based on methods such as duality Raghunathan et al. (2018); Dvijotham et al. (2018), abstract interpretation Gehr et al. (2018), linear approximations Weng et al. (2018); Wong & Kolter (2018), combination of linear and non-linear approximation Xiang et al. (2017), or search space discretization Huang et al. (2017). Incomplete verifiers are more scalable than complete ones, but can suffer from precision loss. In principle, incomplete verifiers can be trivially made complete by iteratively refining the input space; however, at the limit, this eliminates any scalability gains and defeats the purpose of using over-approximations in the first place.

This work. A key challenge then is to design a verifier which improves both, the precision of incomplete verifiers and the scalability of complete ones. In this work, we take a step towards addressing this challenge. We present a verification method that is fast and exact for small networks, yet can analyze larger networks beyond the reach of complete verifiers and compute more precise bounds than incomplete ones. Our method is based on two key ideas: (i) a combination of state-of-the-art over-approximation methods used in incomplete verifiers together with MILP solvers and LP relaxations often employed in complete verifiers; (ii) a novel heuristic which points to neurons whose over-approximation bounds should be refined. Taken together, these ideas result in a significant precision increase for deeper networks (up to 9 layers and 1,800 neurons) over state-of-the-art incomplete verifiers (no complete verifier can handle these networks, even when run for days).

Main contributions. Our main contributions are:

- A refinement-based approach for verifying neural network robustness that combines the strengths of over-approximation methods with MILP solvers and LP relaxations.
- A novel heuristic for selecting neurons whose bounds should be further refined.
- An evaluation showing our system is more precise than existing incomplete verifiers on larger networks and faster (while being complete) than complete ones on small networks.

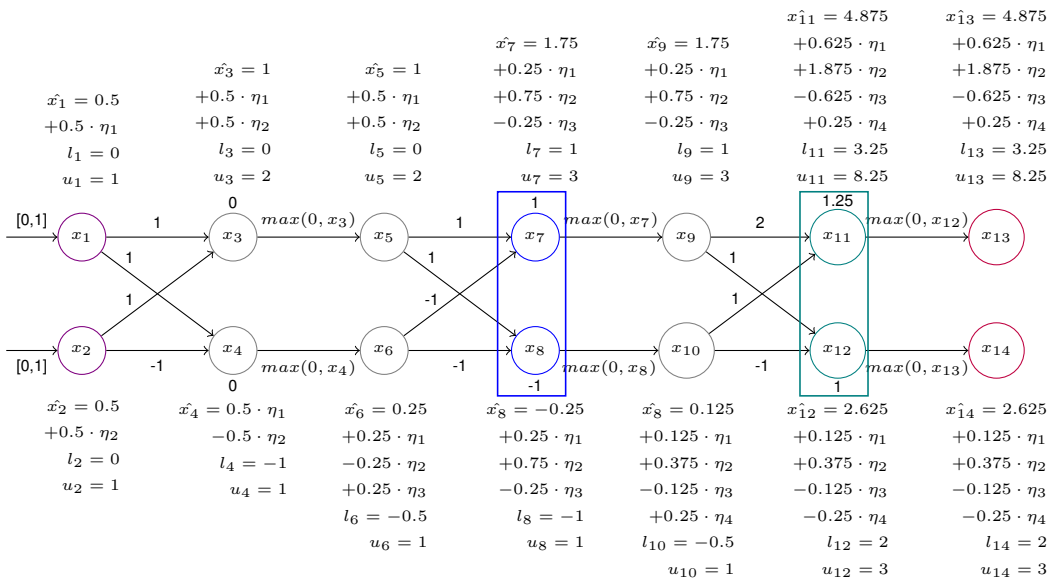

Figure 1: Robustness analysis of neural network using refinement. The analysis results for the neurons in the blue box are refined using MILP whereas for those in green are refined using LP.

2 Overview

We now demonstrate how our refinement based approach improves precision on a simple example. Our goal is to provide an intuitive understanding of the approach; full formal details are provided in the following section.

Consider the simple fully connected neural network with ReLU activations shown in Fig. 1. There are two inputs to the network which are both in the range $[0, 1]$. The network consists of an input layer, two hidden layers and one output layer. Each layer consist of two neurons each. For our explanation, we separate each neuron into two parts: one represents the output of the affine transformation while the other captures the output of the ReLU activation. The weights for the affine transformation are represented by weights on the edges. The bias for each node is shown above or below it. Our goal is to verify that for any input in $[0, 1] \times [0, 1]$, the output at neuron x_{13} is greater than the output at x_{14}.

We now demonstrate our analysis for this network. We assume that the analysis results after the second affine transformation are refined using a MILP formulation of the network whereas the results after the third affine transformation are refined by an LP formulation of the network. In the next section, we will explain our heuristic for selecting either MILP or LP for each neuron. Our analysis leverages the Zonotope domain Ghorbal et al. (2009) transformers also used in AI^2 Gehr et al. (2018), a state of the art analyzer for neural networks. The Zonotope domain associates an affine form with each neuron x in the network:

$$\hat{x} := [\alpha_0, \beta_0] + \sum_{i=1}^{p} [\alpha_i, \beta_i] \cdot \eta_i$$

Here, $\alpha_0, \beta_0, \alpha_i, \beta_i \in \mathbb{R}$ are real coefficients and $\eta_i \in [a_i, b_i] \subseteq [-1, 1]$ are the noise symbols which are shared between the affine forms for different neurons, which makes the domain relational. An element in our analysis is an intersection between a Zonotope (given as conjunction of affine forms) and a bounding box. Thus, for each neuron x, we keep the affine form $\hat{x}$ and an interval $[l_x, u_x]$.

First layer. Our analysis starts by setting:

$$\hat{x}_1 = 0.5 + 0.5 \cdot \eta_1, l_1 = 0, u_1 = 1$$

and

$$\hat{x}_2 = 0.5 + 0.5 \cdot \eta_2, l_2 = 0, u_2 = 1$$

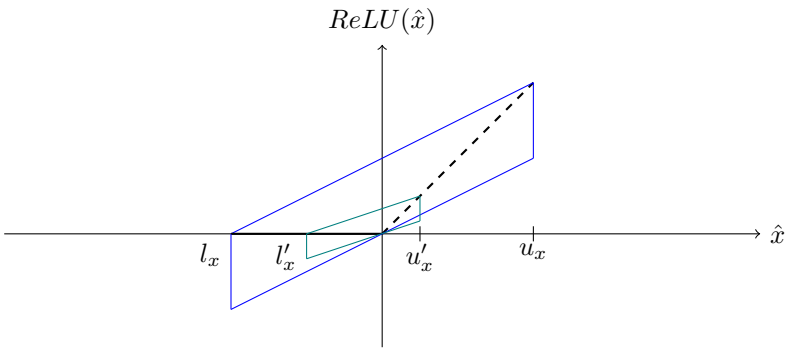

Figure 2: ReLU transformers for computing the affine form of the output. Here l_x, u_x are the original bounds whereas l'_x, u'_x are the refined bounds. The slope of the two non-vertical parallel blue lines is $\lambda = \frac{u_x}{u_x - l_x}$ whereas slope of the two non-vertical parallel green lines is $\lambda' = \frac{u'_x}{u'_x - l'_x}$. The blue parallelogram is used to compute the output affine form in AI^2 whereas the green parallelogram is used to compute the output of the refined ReLU transformer considered in this work.

representing the inputs at x_1 and x_2 respectively. Next, an affine transformation is applied on the inputs resulting in the output:

$$\hat{x}_3 = \hat{x}_1 + \hat{x}_2 = 1 + 0.5 \cdot \eta_1 + 0.5 \cdot \eta_2, l_3 = 0, u_3 = 2$$

and

$$\hat{x}_4 = \hat{x}_1 - \hat{x}_2 = 0.5 \cdot \eta_1 - 0.5 \cdot \eta_2, l_4 = -1, u_4 = 1.$$

The Zonotope affine transformer is *exact* for this transformation so we do not need any refinement here. Next, the Zonotope ReLU transformer is applied. We note that as $l_3 \geq 0$, the neuron x_3 provably takes only positive values. Thus, the ReLU Zonotope transformer outputs $\hat{x}_5 = \hat{x}_3, l_5 = l_3, u_5 = l_3$ which is the exact result. For x_4, $l_4 < 0$ and $u_4 > 0$ and thus neuron x_4 can take both positive and negative values. The corresponding output does not have a closed affine form and hence the approximation in blue shown in Fig. 2 is used to compute the result, introducing a new noise symbol $\eta_3 \in [-1, 1]$. The result is:

$$\hat{x}_6 = 0.25 + 0.25 \cdot \eta_1 - 0.25 \cdot \eta_2 + 0.25 \cdot \eta_3, l_6 = -0.5, u_6 = 1.$$

Note that due to approximation x_6 can take negative values. In contrast, the exact output only takes positive values. This approximation typically accumulates as the analysis progresses deeper into the network, resulting in overall imprecision and failure to prove properties that actually hold.

MILP based refinement at second layer. Next, the analysis handles the second affine transformation and computes:

$$\hat{x}_7 = \hat{x}_5 - \hat{x}_6 + 1 = 1.75 + 0.25 \cdot \eta_1 + 0.75 \cdot \eta_2 - 0.25 \cdot \eta_3, l_7 = 0.5, u_7 = 3$$

and

$$\hat{x}_8 = \hat{x}_5 - \hat{x}_6 - 1 = -0.25 + 0.25 \cdot \eta_1 + 0.75 \cdot \eta_2 - 0.25 \cdot \eta_3, l_8 = -1.5, u_8 = 1.$$

Due to the approximation for x_6, the bounds for x_7 and x_8 are imprecise. Thus, in our method we now refine the bounds by formulating the network up to the second affine transformation as a MILP instance based on a formulation from Tjeng & Tedrake (2017) and compute bounds for x_7 and x_8, respectively. The MILP solver improves the lower bounds for x_7 and x_8 to 1 and -1, respectively, which then updates the corresponding lower bounds. Next, the ReLU transformer is applied. Since x_7 is provably positive, we get $\hat{x}_9 = \hat{x}_7, l_9 = l_7, u_9 = u_7$. We note that x_8 can take both positive and negative values and is therefore approximated. However, the ReLU transformer now uses the refined bounds instead of the original bounds and thus the approximation shown in green from Fig. 2 is used. This approximation has lesser area in the input-output plane compared to the blue one and thus reduces the approximation. The result is:

$$\hat{x}_{10} = 0.125 + 0.125 \cdot \eta_1 + 0.375 \cdot \eta_2 - 0.125 \cdot \eta_3 + 0.25 \cdot \eta_4, l_8 = -0.5, u_8 = 1.$$

LP based refinement at final layer. Continuing with the analysis, we now process the final affine transformation, which yields

$$\hat{x}_{11} = 4.875 + 0.625 \cdot \eta_1 + 1.875 \cdot \eta_2 - 0.625 \cdot \eta_3 + 0.25 \cdot \eta_4, l_{11} = 1.75, u_{11} = 8.25$$

and

$$\hat{x}_{12} = 2.625 + 0.125 \cdot \eta_1 + 0.375 \cdot \eta_2 + -0.125 \cdot \eta_3 - 0.25 \cdot \eta_4, l_{12} = 1.75, u_{12} = 3.5.$$

Due to the approximations from previous layers the computed values can be imprecise. Thus, we refine the bounds by encoding the network up to the third affine transformation using the LP formulation from Ehlers (2017) and compute the bounds for x_{11} and x_{12}, respectively. This leads to the following results: $l_{11} = 3.25$, $l_{12} = 2$, and $u_{12} = 3$. As both x_{13} and x_{14} are provably positive, the subsequent ReLU transformation sets $\hat{x}_{13} = \hat{x}_{11}, l_{13} = l_{11}, u_{13} = u_{11}$ and $\hat{x}_{14} = \hat{x}_{12}, l_{14} = l_{12}, u_{14} = u_{12}$.

Proving robustness. Since the lower bound l_{13} for x_{13} is greater than the upper bound u_{14} for x_{14}, our analysis can prove that the given neural network provides the same label for all inputs in $[0, 1] \times [0, 1]$ and is thus robust. In contrast, AI^2 without our refinement would compute

$$\hat{x}_{13} = 4.95 + 0.6 \cdot \eta_1 + 1.8 \cdot \eta_2 - 0.6 \cdot \eta_3 + 0.3 \cdot \eta_4, l_{13} = 1.65, u_{13} = 8.25$$

and

$$\hat{x}_{14} = 2.55 + 0.45 \cdot \eta_1 - 0.15 \cdot \eta_2 - 0.3 \cdot \eta_3, l_{14} = 1.5, u_{14} = 2.6.$$

As a result, AI^2 fails to prove that x_{13} is greater than x_{14}, and thus fails to prove robustness.

3 Our Approach

In this section we describe our approach in more formal terms. As illustrated earlier, the key idea will be to combine abstract interpretation with exact and inexact MILP formulation of the network which is then solved using a solver so to compute more precise results for neuron bounds. We begin by describing the abstract interpretation ingredients.

Let $\mathbb{A}_n$ be an abstract domain (i.e., some set whose elements can be encoded symbolically) with a bottom element $\perp \in \mathbb{A}_n$. We further assume the following:

- A concretization function $\gamma_n \colon \mathbb{A}_n \to \mathcal{P}(\mathbb{R}^n)$ that associates with each abstract element $a \in \mathbb{A}_n$ a set of concrete vectors from $\mathbb{R}^n$. In particular, $\gamma_n(\perp) = \emptyset$. ($\gamma$ is not required to be computable.)
- An abstraction function $\alpha_n \colon \mathcal{P}(\mathbb{R}^n) \to \mathbb{A}_n$, where $\mathbb{X} \subseteq \gamma_n(\alpha_n(\mathbb{X}))$ for all $\mathbb{X} \in \mathcal{P}(\mathbb{R}^n)$. We assume that $\alpha_n(\prod_i [l_i, u_i])$ can be computed for all $\boldsymbol{l}, \boldsymbol{u} \in \mathbb{R}^n$.
- A bounding box function $\iota_n \colon \mathbb{A}_n \to \mathbb{R}^n \times \mathbb{R}^n$, where $\gamma_n(a) \subseteq \prod_i [l_i, y_i]$ for $(\boldsymbol{l}, \boldsymbol{u}) = \iota_n(a)$ for all $a \in \mathbb{A}_n$.
- A meet operation $a \sqcap L$ for each $a \in \mathbb{A}_n$ and linear constraints L over n real variables, where $\{x \in \gamma_n(a) \mid L(x)\} \subseteq \gamma_n(a \sqcap L)$.
- An affine abstract transformer $T^{\#}_{\boldsymbol{x} \mapsto \boldsymbol{A}\boldsymbol{x}+\boldsymbol{b}} \colon \mathbb{A}_m \to \mathbb{A}_n$ for each affine transformation $(\boldsymbol{x} \mapsto \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}) \colon \mathbb{R}^m \to \mathbb{R}^n$, where $\{\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b} \mid \boldsymbol{x} \in \gamma_n(a)\} \subseteq \gamma_n(T^{\#}_{\boldsymbol{x} \mapsto \boldsymbol{A}\boldsymbol{x}+\boldsymbol{b}})$ for all $a \in \mathbb{A}_m$.
- A ReLU abstract transformer $T^{\#}_{\text{ReLU}|_{\prod_i [l_i, u_i]}} \colon \mathbb{A}_n \to \mathbb{A}_n$, where $\{\text{ReLU}(\boldsymbol{x}) \mid \boldsymbol{x} \in \gamma(a) \cap \prod_i [l_i, u_i]\} \subseteq T^{\#}_{\text{ReLU}|_{\prod_i [l_i, u_i]}}(a)$ for all abstract elements $a \in \mathbb{A}_n$ and for all lower and upper bounds $\boldsymbol{l}, \boldsymbol{u} \in \mathbb{R}^n$ on input activations.

Abstract interpretation. As first shown by Gehr et al. (2018), any such abstract domain induces a method for robustness certification of classification based neural networks with ReLU activations.

For example, assume that we want to certify that a given neural network $f \colon \mathbb{R}^m \to \mathbb{R}^n$ considers class i more likely than class j for all inputs $\boldsymbol{x}'$ with $||\boldsymbol{x}' - \boldsymbol{x}||_\infty \leq \epsilon$ for a given $\boldsymbol{x}$ and ϵ. We can first use the function α to compute a symbolic over-approximation of the set of possible input activations x', namely

$$a_{\text{in}} = \alpha_m(\{\boldsymbol{x}' \in \mathbb{R}^m \mid ||\boldsymbol{x}' - \boldsymbol{x}||_\infty \leq \epsilon\}).$$

Given that the neural network can be written as a composition of affine functions and ReLU layers we can then propagate the abstract element a_{in} through the corresponding abstract transformers to obtain a symbolic over-approximation a_{out} of the output activations of the neural network.

For example, if the neural network is a single-layer perceptron $f(\boldsymbol{x}) = \boldsymbol{A}' \cdot \text{ReLU}(\boldsymbol{A}\boldsymbol{x}+\boldsymbol{b})+\boldsymbol{b}'$ with h hidden neurons, we first compute $a' = T^{\#}_{\boldsymbol{x}\mapsto \boldsymbol{A}\boldsymbol{x}+\boldsymbol{b}}(a_{\text{in}})$, which is a symbolic over-approximation of the input to the ReLU activation function. We then compute $(\boldsymbol{l}, \boldsymbol{u}) = \iota_h(a')$ to obtain opposite corners of a bounding box of all possible ReLU input activations, such that we can apply the ReLU abstract transformer:

$$a'' = T^{\#}_{\text{ReLU}|_{\prod_i [l_i, u_i]}}(a').$$

Finally, we apply the affine abstract transformer again to obtain $a_{\text{out}} = T^{\#}_{\boldsymbol{x}\mapsto \boldsymbol{A}'\boldsymbol{x}+\boldsymbol{b}'}(a'')$. Using our assumptions, we can conclude that the set $\gamma_n(a_{\text{out}})$ contains all output activations that f can possibly produce when given any of the inputs $\boldsymbol{x}'$. Therefore, if $a_{\text{out}} \sqcap (y_i \leq y_j) = \bot$, we have successfully proved our property: for all $\boldsymbol{x}'$, the neural network considers class i more likely than class j.

Incompleteness. While this approach is sound (i.e., whenever we prove the property, it actually holds), it is incomplete (i.e., we might not prove the property, even if it holds), because the abstract transformers produce a superset of the set of possible activations. This is particularly quite imprecise for deep neural networks, because the over-approximations introduced in each layer accumulate.

Refinement. To combat spurious over-approximation, we use mixed integer linear programming (MILP) to compute refined lower and upper bounds $\boldsymbol{l}, \boldsymbol{u}$ after applying each abstract transformer (except the first). We then refine the abstract element using the meet operator and the linear constraints $l_i \leq x_i \leq u_i$ for all input activations i, i.e., we replace the current abstract element a by $a' = a \sqcap (\bigwedge_i l_i \leq x_i \leq u_i)$, and continue analysis with the refined abstract element.

Furthermore, and importantly, for ReLU, we obtain a refined abstract transformer using the new (and tighter) lower and upper bounds.

To enable refinement, we augment each abstract element with variables and constraints. There is one variable for each neuron, and the constraints fully capture the behaviour of the neural network up to the last layer whose abstract transformer has been executed. We write $x_i^{(k)}$ to denote the variable corresponding to the activation of the i-th neuron in the k-th layer, where the input layer has $k = 0$. Similarly, we will write $l_i^{(k)}$ and $u_i^{(k)}$ to denote the best derived lower and upper bounds for this neuron.

From the input layer, we get constraints of the form $a \leq x_i^{(0)} \leq b$, from affine layers, we get constraints of the form $x_i^{(k)} = a_0 + \sum_j a_j x_j^{(k-1)}$ and from ReLU layers we get constraints of the form $x_i^{(k)} = \text{ReLU}(x_i^{(k-1)})$.

MILP. Let $\varphi^{(k)}$ denote the conjunction of all constraints up to and including those from layer k. To obtain the best possible lower and upper bounds for layer k with p neurons, we need to solve the following $2p$ optimization problems:

$$\begin{aligned} l_i^{(k)} &= \min_{\substack{x_1^{(0)},\ldots,x_p^{(k)} \\ \text{s.t. } \varphi^{(k)}(x_1^{(0)},\ldots,x_p^{(k-1)})}} x_i^{(k)}, \text{ for } i = 1, \ldots, p, \\ u_i^{(k)} &= \max_{\substack{x_1^{(0)},\ldots,x_p^{(k)} \\ \text{s.t. } \varphi^{(k)}(x_1^{(0)},\ldots,x_p^{(k-1)})}} x_i^{(k)}, \text{ for } i = 1, \ldots, p. \end{aligned}$$

As was shown by Tjeng & Tedrake (2017), those optimization problems can be encoded exactly as MILPs, such that the bounds can be computed with off-the-shelf MILP solvers. Note that the encoding of subsequent layers depends on the bounds computed in previous layers, where tighter bounds produce simpler MILP instances.

LP relaxation. While not introducing any approximation, unfortunately, current MILP solvers do not scale to larger deep neural networks. However, for soundness it is not crucial that the produced bounds are the best possible: for example, plain abstract interpretation uses sound bounds produced by the bounding box function ι instead. We explore the trade-off between precision and scalability by also considering an intermediate method, which is faster than exact MILP, but also more precise than abstract interpretation. We relax a subset of the constraints in $\varphi^{(k)}$ in the same way as Ehlers (2017) to obtain a set of weaker linear constraints $\varphi_{\text{LP}}^{(k)}$. We then use the solver to solve the relaxed optimization problems that are constrained by $\varphi_{\text{LP}}^{(k)}$ instead of $\varphi^{(k)}$, producing possibly looser bounds $\boldsymbol{l}^{(k)}$ and $\boldsymbol{u}^{(k)}$. Note that the encoding of subsequent layers depends on the bounds computed in previous layers, where tighter bounds reduce the amount of newly introduced approximation.

Anytime MILP relaxation. The MILP solvers usually provide the option to provide an explicit timeout after which the solver must terminate. In return, the solver may not be able to solve the instance exactly, but it will instead provide lower and upper bounds on the objective function in a best-effort fashion. This provides another way to compute sound but inexact bounds $\boldsymbol{l}^{(k)}$ and $\boldsymbol{u}^{(k)}$.

In practice, we choose a fraction $\alpha \in (0, 1]$ of neurons in a given layer k and compute bounds for them using MILP with a timeout T in a first step. In the second step, for a fraction $\delta \in [0, 1 - \alpha]$ of neurons in the layer, we set the timeout to $\beta \cdot \overline{T}$, where $\overline{T}$ is the average time taken by the MILP solver to solve one of the instances from the first step and $\beta \in [0, 1]$ is a parameter.

Neuron selection heuristic. To select the α-fraction of neurons for the first step of the anytime MILP relaxation for the k-th layer, we rank the neurons. If the next layer is a ReLU layer, we first ignore all neurons whose activations can be proven to be non-positive using abstract interpretation (i.e., using the bounds produced by ι), because in this case it is already known that ReLU will map the activation to 0. The remaining neurons are ordered in up to two different ways, once by width (i.e. neuron i has key $u_i^{(k)} - l_i^{(k)}$), and possibly once by the sum of absolute output weights. i.e., if the next layer is a fully connected layer $\boldsymbol{x} \mapsto \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}$, the key of neuron i is $\sum_j |A_{i,j}|$. If the next layer is a ReLU layer, we skip the ReLU layer and use the weights from the fully connected layer that follows it (if any). The two ranks of a neuron in both orders are added, and the α-fraction with smallest rank sum is selected and their bounds refined with a timeout of T whereas the next δ-fraction of neurons are refined with a timeout of $\beta \cdot \overline{T}$.

End-to-end approach. To certify robustness of deep neural networks, we combine MILP, LP relaxation, and abstract interpretation. We first pick numbers of layers $k_{\text{MILP}}, k_{\text{LP}}, k_{\text{AI}}$ that sum to the total number of layers of the neural network. For the analysis of the first k_{MILP} layers, we refine bounds using anytime MILP relaxation with the neuron selection heuristic. As an optimization, we do not perform refinement after the first abstract transformer in case it is an affine transformation and the abstract domain computes the tightest possible bounding box for an affine transformation of a box (this is always the case in our experiments). For the k_{LP} layers after that, we refine bounds using both MILP and LP relaxation, i.e., the first k_{MILP} layers are encoded as MILP and the next k_{LP} use the LP relaxation. For the final k_{AI} layers, we use abstract interpretation without additional refinement (however, this also benefits from refinement that was performed in previous layers), and compute the bounds using ι.

Final property verification. In case the refined abstract interpretation is already able to verify the final property on its own, we terminate. Otherwise, we can encode the final verification problem using $\varphi_{\text{LP}}^{(k)}$. If we want to prove that class i is assigned a higher probability than class j, it suffices to show that the constraints $\varphi_{\text{LP}}^{(k)}(x_1^{(0)}, \ldots, x_p^{(k)}) \wedge x_i^{(k)} \leq x_j^{(k)}$ are unsatisfiable, where k is the index of the last layer and p is the number of output classes. If this fails as well, one can resort to complete verification using MILP: the property is satisfied if and only if $\varphi^{(k)}(x_1^{(0)}, \ldots, x_p^{(k)}) \wedge x_i^{(k)} \leq x_j^{(k)}$ is unsatisfiable.

4 EVALUATION

We now evaluate the effectiveness of our approach for the problem of robustness verification of ReLU-based feed forward neural networks. We implemented our approach in the form of a Python analyzer which we call RefineAI. RefineAI uses the Zonotope and Interval implementation from the ELINA eli; Singh et al. (2017) library. Gurobi gur is used for solving MILP instances.

Evaluation datasets. We used the popular MNIST Lecun et al. (1998) dataset of hand-written digits for our experiments. The dataset includes a training set of $60\,000$ and a testing set of $10\,000$ gray scale images. All images are of size 28×28 pixels.

Neural networks. We use 5 different feed forward networks (FNNs) with the ReLU activation function trained for the MNIST dataset. The sizes of the FNNs are: (a) 3 layers with 50 units each, (b) 5 layers with 100 units each, (c) 6 layers with 100 units each, (d) 6 layers with 200 units each, and (e) 9 layers with 200 units each.

Among these, the 5×100 network is trained to be robust using the adversarial training from Mirman et al. (2018) whereas the other networks were taken from Gehr et al. (2018) and are not trained to be robust.

Robustness properties. We consider the adversarial region defined by the L_∞-norm Carlini & Wagner (2017) perturbation. The region is parameterized by $\epsilon \in \mathbb{R}$ and contains all images in which each pixel $\overline{x}'_i$ has a distance of at most ϵ from the corresponding pixel $\overline{x}_i$ in the original input image $\overline{x}$. Our goal is to verify that the neural network produces the same label on all images in the adversarial region as on the original image. The adversarial attack considered here is untargeted and therefore stronger than the targeted attack in Weng et al. (2018); Tjeng & Tedrake (2017) where the goal is to prove that the score for the correct label is more than *some* other label on all images in the adversarial region.

Benchmarks. We selected the first 100 images from the MNIST test set. Any image on which the neural network misclassified without perturbation was not considered for evaluation.

Experimental setup. All experiments for the 3×50 FNN were carried out on a 2.6 GHz Intel Xeon CPU E5-2690 with 512 GB of main memory; the remaining FNNs were evaluated on a 3.3 GHz 10 Core Intel i9-7900X Skylake CPU with a main memory of 64 GB. For each network, we chose an ϵ for which AI^2 verified $< 40\%$ of all candidate images. We used a time limit of 40 minutes per run in our experiments. We experimented with different values of the analysis parameters $k_{\text{MILP}}, k_{\text{LP}}, k_{\text{AI}}, \alpha, \delta, \beta, T$ and chose those values that offered the best tradeoff between performance and precision for the robustness verification of larger neural networks.

Complete verification of small network. We first consider the exact robustness verification of 3×50 networks against L_∞-norm attack with $\epsilon = 0.03$. AI^2 verifies 37% of images in the test set for this value of ϵ. RefineAI runs the analysis with AI^2 on the whole network collecting the bounds for all neurons in the network. If AI^2 does not verify the property then the bounds from AI^2 are used to encode the robustness verification as a MILP problem as discussed in section 3. We compare RefineAI against the approach from Tjeng & Tedrake (2017) (state-of-the-art for complete verifiers) which also uses a MILP formulation and uses a presolve analysis to formulate the verification as a MILP problem. We implemented the Interval analysis and LP-based analysis to determine the initial bounds. We note that LP analysis is the default presolve in Tjeng & Tedrake (2017). If LP analysis (or Interval analysis) can already verify an image then the MILP solver is not called. All complete verifiers verify 85% of images. The average runtimes of RefineAI, MILP with presolve bounds from Interval analysis, and MILP with bounds from LP analysis are 28, 123, and 35 seconds respectively.

Incomplete verification of larger networks. Complete verification does not scale for the remaining networks. We note that running LP analysis to determine initial bounds for the network does not scale for larger networks whereas the bounds from the Interval analysis are quite imprecise. The values of ϵ considered for the L_∞-norm attack on 5×100, 6×100, 6×200, and 9×200 are 0.07, 0.02, 0.015, and 0.015 respectively. For all networks, we set $k_{\text{MILP}} = 3, k_{\text{LP}} = n - k_{\text{MILP}} - 1, k_{\text{AI}} =$

$1, T = 1s, \beta = 0.5$ (n is the number of layers). The value of α for the first two layers is equal to number of neurons in the layer that can take positive values. For the remaining layers, we choose a fraction of neurons taking positive values for MILP refinement with timeout T. The remaining δ neurons with positive values are refined with MILP using a timeout of $\beta \cdot \overline{T}$. In Fig. 3, we compare the precision and the average runtime (in log scale) of RefineAI against AI^2 per image for different neural networks. It can be seen that RefineAI is significantly more precise than AI^2. RefineAI verifies $> 2x$ more images than AI^2 on 3 out of 4 neural networks. Specifically, for the largest 9×200 neural network, AI^2 manages to verify the robustness of only 12% of all images. In contrast, RefineAI verifies 35% of images to be robust. The average runtime per image of RefineAI is < 20 minutes for all neural networks.

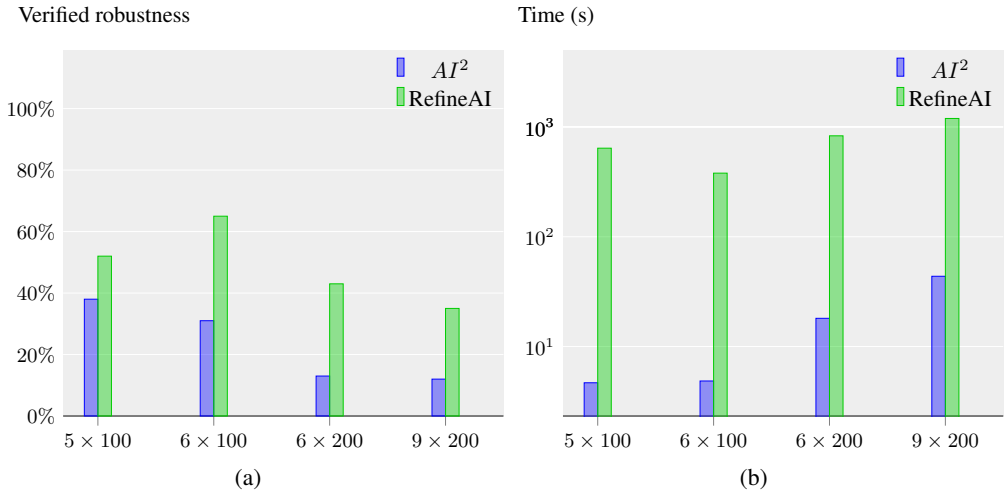

Figure 3: Verified robustness based on L_∞ norm by RefineAI vs AI^2 on the MNIST FNNs. The values of ϵ for 5×100, 6×100, 6×200, and 9×200 FNNs are 0.07, 0.02, 0.015, and 0.015 respectively.

Effect of neuron selection heuristic. Refining all neurons in a layer with MILP can significantly slow down the analysis. We use the neuron selection heuristic from section 3 to determine neurons which need to be refined more than others. To check whether our heuristic is able to identify important neurons, we ran the analysis on the 6×100 FFN by keeping all analysis parameters the same, except instead of selecting the neurons with the smallest rank sum first we selected the neurons with the largest rank sum first (thus refining unimportant neurons as per our heuristic more). With this change the modified analysis loses precision and verifies 62% of images instead of 65% that the analysis refining with our neuron selection heuristics verifies. Although the number of images verified change by only 3%, the analysis with our heuristic produces tighter output bounds on all images than without it.

5 Conclusion

In this paper we presented a refinement based approach for increasing the precision of incomplete verifiers for feed forward neural networks with ReLU activations. This enabled us to verify robustness properties that existing state-of-the-art complete verifiers cannot handle (due to scalability issues) whereas state-of-the-art incomplete verifiers could not prove due to using over-approximations which are too coarse. We believe the direction of combining the strengths of over-approximation methods with that of mixed integer linear programming as done in this work is a promising direction for advancing the state-of-the-art in neural network verification.

REFERENCES

ELINA,: Eth library for numerical analysis.

Gurobi, optimization. http://www.gurobi.com/.

Filippo Amato, Alberto Lpez, Eladia Mara Pea-Mndez, Petr Vahara, Ale Hampl, and Josef Havel. Artificial neural networks in medical diagnosis. *Journal of Applied Biomedicine*, 11(2):47 – 58, 2013.

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. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016.

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

Chih-Hong Cheng, Georg Nührenberg, and Harald Ruess. Maximum resilience of artificial neural networks. In *Automated Technology for Verification and Analysis (ATVA)*, 2017.

Souradeep Dutta, Susmit Jha, Sriram Sankaranarayanan, and Ashish Tiwari. Output range analysis for deep feedforward neural networks. In *NASA Formal Methods (NFM)*, 2018.

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

Rüdiger Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In *Automated Technology for Verification and Analysis (ATVA)*, 2017.

Matteo Fischetti and Jason Jo. Deep neural networks and mixed integer linear optimization. *Constraints*, 23(3):296–309, 2018.

T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *Proc. IEEE Symposium on Security and Privacy (SP)*, volume 00, pp. 948–963, 2018.

Khalil Ghorbal, Eric Goubault, and Sylvie Putot. The zonotope abstract domain taylor1+. In *Proc. Computer Aided Verification (CAV)*, pp. 627–633, 2009.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, abs/1706.07351, 2014.

G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks. In *Computer Aided Verification (CAV)*, pp. 3–29, 2017.

Guy Katz, Clark Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *Computer Aided Verification (CAV)*, pp. 97–117, 2017.

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

Alessio Lomuscio and Lalit Maganti. An approach to reachability analysis for feed-forward relu neural networks. *CoRR*, 2017.

Matthew Mirman, Timon Gehr, and Martin Vechev. Differentiable abstract interpretation for provably robust neural networks. In *Proc. International Conference on Machine Learning (ICML)*, volume 80, pp. 3575–3583, 2018.

Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Certified defenses against adversarial examples. *CoRR*, abs/1801.09344, 2018.

Gagandeep Singh, Markus Püschel, and Martin Vechev. Fast polyhedra abstract domain. In *Proc. Principles of Programming Languages (POPL)*, pp. 46–59, 2017.

Vincent Tjeng and Russ Tedrake. Verifying neural networks with mixed integer programming. *CoRR*, abs/1711.07356, 2017.

Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Formal security analysis of neural networks using symbolic intervals. In *USENIX Security Symposium (USENIX Security 18)*, pp. 1599–1614, 2018.

Lily Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Luca Daniel, Duane Boning, and Inderjit Dhillon. Towards fast computation of certified robustness for ReLU networks. In *Proc. International Conference on Machine Learning (ICML)*, volume 80, pp. 5276–5285, 2018.

Eric Wong and Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *Proc. International Conference on Machine Learning (ICML)*, volume 80, pp. 5286–5295, 2018.

Weiming Xiang, Hoang-Dung Tran, and Taylor T. Johnson. Output reachable set estimation and verification for multi-layer neural networks. *CoRR*, abs/1708.03322, 2017.