Rössig, Ansgar; Petkovic, Milena

Article — Published Version

Advances in verification of ReLU neural networks

Journal of Global Optimization

Provided in Cooperation with:
Springer Nature

Suggested Citation: Rössig, Ansgar; Petkovic, Milena (2020) : Advances in verification of ReLU neural networks, Journal of Global Optimization, ISSN 1573-2916, Springer US, New York, NY, Vol. 81, Iss. 1, pp. 109-152, https://doi.org/10.1007/s10898-020-00949-1

This Version is available at:
https://hdl.handle.net/10419/288360

Journal of Global Optimization (2021) 81:109–152
https://doi.org/10.1007/s10898-020-00949-1

Advances in verification of ReLU neural networks

Ansgar Rössig[1] · **Milena Petkovic**[2]

Received: 1 August 2019 / Accepted: 5 September 2020 / Published online: 27 October 2020

Abstract
We consider the problem of verifying linear properties of neural networks. Despite their success in many classification and prediction tasks, neural networks may return unexpected results for certain inputs. This is highly problematic with respect to the application of neural networks for safety-critical tasks, e.g. in autonomous driving. We provide an overview of algorithmic approaches that aim to provide formal guarantees on the behaviour of neural networks. Moreover, we present new theoretical results with respect to the approximation of ReLU neural networks. On the other hand, we implement a solver for verification of ReLU neural networks which combines mixed integer programming with specialized branching and approximation techniques. To evaluate its performance, we conduct an extensive computational study. For that we use test instances based on the ACAS Xu system and the MNIST handwritten digit data set. The results indicate that our approach is very competitive with others, i.e. it outperforms the solvers of Bunel et al. (in: Bengio, Wallach, Larochelle, Grauman, Cesa-Bianchi, Garnett (eds) Advances in neural information processing systems (NIPS 2018), 2018) and Reluplex (Katz et al. in: Computer aided verification—29th international conference, CAV 2017, Heidelberg, Germany, July 24–28, 2017, Proceedings, 2017). In comparison to the solvers ReluVal (Wang et al. in: 27th USENIX security symposium (USENIX Security 18), USENIX Association, Baltimore, 2018a) and Neurify (Wang et al. in: 32nd Conference on neural information processing systems (NIPS), Montreal, 2018b), the number of necessary branchings is much smaller. Our solver is publicly available and able to solve the verification problem for instances which do not have independent bounds for each input neuron.

Keywords Neural networks verification · ReLU · MIP

✉ Milena Petkovic
petkovic@zib.de

Ansgar Rössig
ansgar_roessig@posteo.de

[1] Institute for Mathematics, Software and Algorithms for Discrete Optimization, Technische Universität Berlin, Straße des 17. Juni 136, Berlin, Germany

[2] Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany

1 Introduction

During the last few years, various approaches have been presented that aim to provide formal guarantees on the behaviour of neural networks. The use of such verification methods may be crucial to enable the secure and certified application of neural networks for safety-critical tasks. Moreover, based on first results of [32], awareness was raised that neural networks are prone to fail on so called adversarial examples. These are created by small perturbations of input samples, such that the changes are (almost) imperceptible to humans. However, these perturbations are often sufficient to make a neural network fail on the input sample. The existence of such adversarial examples can be ruled out by methods of neural network verification. In fact, a closely related line of research termed as robustness certification is focused explicitly on this topic.

In the following section we formally introduce the problem that we regard. In Sect. 3 we provide an overview of related work, and present formulations of the verification problem as MIP in Sect. 4. In the subsequent sections we consider approximation techniques, primal heuristics, and branching methods for verification of neural networks. Extensive computational results on the performance of our solver and others can be found in Sect. 8, and Sect. 9 concludes the paper with some final remarks. Additional material can be found in the appendices. Our solver, which is based on the academic MIP solver SCIP [13], is publicly available at https://github.com/roessig/verify-nn.

For the ease of notation, we use $[n]$ for $n \in \mathbb{N}$ to denote the set $\{1, \dots, n\}$. In our work, we only regard trained neural networks, which can be seen as immutable and deterministic functions $F : \mathbb{R}^n \to \mathbb{R}^m$. F is determined by its weights and biases $((A_l, b_l))_{l=1}^L$. It holds $A_l \in \mathbb{R}^{N_l \times N_{l-1}}$ for $l \in [L]$ and $b_l \in \mathbb{R}^{N_l}$, $l \in [L]$, where L is the number of layers in the neural network. $N_0, \dots, N_L$ are the numbers of neurons per layer (cf. Bölcskei et al. [3], Definition 1).

2 Problem definition

Now we give a formal definition of the verification problem for ReLU neural networks and comment on some relevant properties of this problem. In the following we use the term *(solving) model* to refer to an algorithmic approach for solving the verification problem. This may encompass a range of choices in obtaining an actual algorithm.

Definition 1 *(Verification Problem for ReLU Neural Networks)* Assume that $\emptyset \neq X \subset \mathbb{R}^n$ is a polytope, and let $\emptyset \neq Y \subset \mathbb{R}^m$ be such that $Y = \bigcap_{i=1}^k Q_i$ or $Y = \bigcup_{i=1}^k Q_i$ where $k \in \mathbb{N}$ and $Q_i \subseteq \mathbb{R}^m$ is a halfspace for $i \in [k]$. Given a neural network $F : X \to \mathbb{R}^m$ with ReLU activation function, the verification problem consists in the decision whether $F(X) \subseteq Y$ holds. A triple (X, Y, F) will be called an *instance of the verification problem (for ReLU neural networks)*. Furthermore, if $F(X) \subseteq Y$, we say that the instance is *verifiable*, otherwise it is *refutable*.

The construction of the feasible input polytope X and the set of admissible outputs Y is solely based on the application for which the neural network shall be used. Depending on the algorithm which is used to solve the problem, the halfspaces Q_i can either be open or closed. Though, either all of them must be closed or all of them must be open. However, the use of floating point arithmetic by a solver for the verification problem makes this distinction rather unimportant, since numerical comparisons require the use of a certain threshold difference.

Moreover, Katz et al. [17] show that the verification problem for ReLU neural networks is NP-complete. Hence we cannot expect that the problem can be solved efficiently in general. We also follow the naming concept of Katz et al. [17] and refer to verifiable instances of the verification problem as *UNSAT* instances, and to refutable instances as *SAT* instances. This naming corresponds to the existence of a counterexample as defined in the following remark.

Remark 1 If an instance (X, Y, F) is refutable, i.e. $F(X) \nsubseteq Y$, we want to provide $x \in X$ such that $F(x) \notin Y$. We will refer to this $x \in X$ as a *counterexample* for the instance.

Remark 2 More complex properties can be investigated by spliting them into separate instances. For example, if $Y = \left(\bigcup_{i=1}^{k} Q_i\right) \cap \left(\bigcup_{j=1}^{l} P_j\right)$ for halfspaces Q_i and P_j and $k, l \in \mathbb{N}$, then $F(X) \subseteq Y$ holds if and only if $F(X) \subseteq (\bigcup_i^k Q_i)$ and $F(X) \subseteq (\bigcup_i^l P_i)$.

Remark 3 Considering an instance $\Pi = (X, Y, F)$ of the verification problem with $X \subset \mathbb{R}^n$, we will often assume the existence of bounds l_i, u_i for $i \in [n]$ such that $l_i \leq x_i \leq u_i$ for $x \in X$. Indeed, the requirements of Definition 1 justify this assumption. These bounds can be computed by solving one LP per bound. We set

$$l_i := \min_{x \in X} x_i \qquad \text{and} \qquad u_i := \max_{x \in X} x_i \qquad \text{for } i \in [n].$$

In fact, for all publicly available instances of the verification problem that we are aware of, the polytope X is actually a box which is directly given by the bounds l_i, u_i for $i \in [n]$. For these instances it is thus not necessary to solve any LP in order to obtain the bounds. However, in this paper we also consider instances where X is not a box, cf. Sect. 8 and "Appendix A".

Remark 4 Assume that we are given an instance $\Pi = (X, Y, F)$ of the verification problem as introduced in Definition 1. Some solving models of other authors are not only limited to instances where the input polytope X is in fact a box. Also the choice of output constraints as represented by Y is more restricted for these models. These require that $Y = \bigcup_{i=1}^{k} Q_i \subseteq \mathbb{R}^m$, where $k \in \mathbb{N}$ and $Q_i \subseteq \mathbb{R}^m$ is an open halfspace for $i \in [k]$. Indeed, this is the only of the cases which are regarded in Definition 1 where $\mathbb{R}^m \setminus Y$ is a polyhedron. Yet, it is possible to use such restricted solving models to solve an instance $\Pi = (X, Y, F)$ where $Y = \bigcap_{i=1}^{k} Q_i \subseteq \mathbb{R}^m$ for open halfpaces $Q_i \subseteq \mathbb{R}^m$. To this end, it is necessary to split the corresponding instance into k instances (X, Q_i, F). Clearly, if $F(X) \subseteq Q_i$ for all $i \in [k]$, then it holds $F(X) \subseteq Y$ and Π is verifiable. On the other hand, if there is $x \in X$ and some $i \in [k]$ such that $F(x) \notin Q_i$, we know that Π is refutable since $F(X) \nsubseteq Y$. We will refer to such an instance Π as *conjunction instance*. On the other hand, an instance $\Pi = (X, Y, F)$ where $Y = \bigcup_{i=1}^{k} Q_i \subseteq \mathbb{R}^m$ for open halfspaces $Q_i \subseteq \mathbb{R}^m$, will be called *disjunction instance*. We will also regard those instances as disjunction instances that fulfill $Y = Q$ for some open halfspace $Q \subseteq \mathbb{R}^m$. In fact, all instances that we consider in our computational experiments (see Sect. 8) are based on open halfspaces. Closed halfspaces are only mentioned in some cases to provide a comprehensive explanation.

Often, we will regard constraints of the form $y = \mathrm{ReLU}(x) := \max(\{x, 0\})$ for $x \in [l, u]$, $y \in \mathbb{R}$, that refer to a certain neuron with ReLU activation function. If the bounds $l, u \in \mathbb{R}$ with $l \leq u$ are such that either $l \geq 0$ or $u \leq 0$, we say that the corresponding neuron is *fixed (in its phase)*.

3 Related work

The key properties of the problem as given in Definition 1 are considered likewise in the literature for neural network verification [4–6,9,10,17,19,21,22,33,35,36,40,41]. In view of an instance $\Pi = (X, Y, F)$ these can be summarized as follows.

A box, a polytope or a union of polytopes is defined as the feasible input domain X for the property which shall be verified. Then, linear properties are defined that we denote in terms of a set Y, such that Π is verifiable if and only if $F(X) \subseteq Y$. Complete algorithms (except in [9]) are employed to solve this problem, i.e. if there exists $\tilde{x} \in X$ such that $F(\tilde{x}) \notin Y$, this will be reported. Clearly, the verification problem is not necessarily limited to neural networks with ReLU activations, i.e. other activation functions are sometimes considered, too. Cheng et al. [7] and Narodytska et al. [20] regard the verification problem on binarized neural networks, which we do not investigate further.

First approaches to verification of neural networks [21,22,26] belong to the field of satisfiability modulo theories (SMT), which generalize the boolean satisfiability problem by replacing variables with predicates from various theories. Also the solver Reluplex [17] for verification of neural networks is presented in this context, but solves instances which are significantly more difficult, using an extended version of the well known simplex algorithm. Ehlers [10] presents the solver Planet, which is based on LP and SAT solving. Dvijotham et al. [9] formulate the verification problem as a non-convex optimization problem and obtain an incomplete algorithm. Xiang et al. [41] regard the propagation of an input polytope through a ReLU neural network, and Xiang et al. [40] propose to discretize the input domain in order to verify neural networks. However, their work remains limited to theoretical considerations and the presentation of numerical toy examples.

Various authors [6,11,19,33] consider MIP models for the verification problem. The performance of such MIP models is predominantly determined by the quality of the bounds which are computed for the ReLU neurons in the neural network. For that reason, the computation of such bounds is discussed in more detail in Sect. 5. The use of appropriate branching schemes is also important for an MIP model of the verification problem, we will provide more details on this in Sect. 7. In fact, it is not necessary to solve the verification problem as an MIP if such approximation and branching methods are used. Bunel et al. [5] present such a branch-and-bound method without solving the verification problem as an MIP directly. Moreover, they provide a good comparison of various methods for neural network verification. Besides their own approach, the empirical evaluation includes Reluplex [17], Planet [10], and an MIP model based on the suggestions of various authors [6,19,33]. While we also implement an MIP model to solve the verification problem, its functioning is more similar to the branch-and-bound method of Bunel et al. [5] than to the MIP model they use in their comparison. Besides, we consider various additional aspects, and therefore speed up the solving process significantly. For a computational comparison of other solvers with ours, we select Reluplex [17] and the branch-and-bound method [5]. The other solvers regarded by Bunel et al. [5] are not competitive with these, as their experimental results show. Moreover, we regard the solvers ReluVal and Neurify as introduced by Wang et al. [35,36]. The concept for both solvers is also a branch-and-bound scheme, that works with a frequent linear approximation of the regarded neural network. In contrast to the method of Bunel et al. [5], the approximation is not as good, but much faster to compute.

Anderson et al. [2] present an ideal MIP formulation for ReLU constraints which is closely related with the techniques used in our work. Especially, they present a separation routine which can be used to compute stronger neuron bounds. Optionally, we include this separator

in our solving model as mentioned in Sect. 8.2. Nevertheless, it should be noted that the results of Anderson et al. [2] do refer only to single ReLU neurons and at most the layer before. Hence we do not have an ideal formulation of the whole network which implies that solving the verification problem cannot be reduced to solving an LP using their formulations.

The idea of output range or reachability analysis is in principle to compute the output range $F(X)$ of a neural network F, given an input domain X. Since this is quite difficult, the relevant work of Dutta et al. [8] and Ruan et al. [25] is limited to computing the range $g(F(X))$, for some function $g : F(X) \to \mathbb{R}$. The function g should then give some insights into the output of the neural network F on input domain X. Clearly, this problem is closely related to the verification problem.

Several authors [12,23,29–31,34,37–39,42,43] consider the problem of computing robustness guarantees for neural networks which are used for classification. Robustness means, that the classification of an input sample should remain the same when the input is changed by small perturbations. The computation of certified robustness bounds should rule out the existence of adversarial examples. Indeed, this problem is a special case of neural network verification. Except for Tjeng et al. [34], this problem is solved by incomplete algorithms. That means, an algorithm either returns a guarantee that a region around an input sample is free of adversarial examples, or no result, which is due to the use of approximations.

Modelling ReLU neural networks as MIPs is considered in the literature for other application domains, too. Grimstad and Andersson [15] investigate the usage of ReLU neural networks as surrogate models in MIPs and study various bound tightening techniques. Serra et al. [28] apply a MIP formulation of a ReLU neural network to enable a lossless pruning method. This way, equivalent neural networks of smaller size can be obtained. The computation of linear regions in ReLU neural networks is another field of application [27].

4 Neural network verification as MIP

It is straightforward to formulate the verification problem as a mixed integer program (MIP), see [6,8]. We present a slightly improved formulation, as it can be found in [5,34]. In this formulation, each neuron is represented by one or two (continuous) variables. The value of a neuron before application of the ReLU function is given as a linear combination of the output values of the predecessor neurons in the network plus the bias. That means, this connection can be simply modelled by a linear equation in the MIP. We need two variables for neurons with ReLU activation function. Let variable x be the input value to the ReLU function and y be the output value. In this setting we will refer to x as the *ReLU input variable* and to y as *ReLU output variable*. We want to model $y = \max\{0, x\}$, which is represented using one additional binary variable d. Furthermore, we need that upper and lower bounds $l \leq x \leq u$ are known. Then we obtain the following constraints which are equivalent to $y = \max\{0, x\}$:

$$\begin{aligned} & y \geq x, \quad y \geq 0 \\ & y \leq x - (1-d)l, \quad y \leq d \cdot u \\ & d \in \{0, 1\} \\ & x \in [l, u], \ l < 0 < u \end{aligned} \tag{1}$$

Of course it is possible that we have $l \geq 0$ or $u \leq 0$ for the bounds. In these cases, we can omit the binary variable d and replace (1) as follows. If $l \geq 0$, this implies $y = \max\{0, x\} = x$, i.e. (1) is replaced by $y = x$ for $x \in [l, u]$. If $u \leq 0$, we have $y = \max\{0, x\} = 0$ and thus

we can set $y = 0$ for $x \in [l, u]$. These cases correspond to fixing the binary variable d to 1 or 0, respectively.

Let $\Pi = (X, Y, F)$ be a disjunction instance of the verification problem such that it holds $Y = \bigcup_{i=1}^{k} Q_i \subseteq \mathbb{R}^m$ for certain open halfspaces Q_i, $i \in [k]$. Then it is straightforward to formulate an MIP which is feasible if and only if Π is refutable. The instance Π of the verification problem is represented by the following constraints:

$$x \in X, \ y \in \mathbb{R}^m \setminus Y \text{ and } y = F(x) \tag{2}$$

This is an MIP, since $x \in X$ and $y \in \mathbb{R}^m \setminus Y$ can be represented by linear constraints. Especially, $y = F(x)$ can be expressed by linear constraints combined with integrality constraints for auxiliary binary variables that are used to model the ReLU function as shown in (1). Now, if the MIP (2) is feasible, there exists $x \in X$ such that $F(x) = y \notin Y$. This implies $F(X) \nsubseteq Y$ and hence Π is refutable. Otherwise, if MIP (2) is not feasible, that means that for all $x \in X$ it holds $F(x) = y \in Y$ and thus Π is verifiable.

For conjunction instances where $Y = \bigcap_{i=1}^{k} Q_i \subseteq \mathbb{R}^m$ for open halfspaces $Q_i, i \in [k]$, we consider two options. Either we split instance Π into k instances as mentioned in Remark 4 or we formulate the verification problem as an optimization problem as proposed by Bunel et al. [5]. In this setting, an instance $\Pi = (X, Y, F)$ is verifiable if the optimum value of the corresponding optimization problem is greater than zero and refutable if it is lower than zero.

Assume that $Y = \bigcup_{i=1}^{k} Q_i \subseteq \mathbb{R}^m$ where $Q_i, i \in [k]$ are open halfspaces. This implies the existence of $q_i \in \mathbb{R}^m$ and $b_i \in \mathbb{R}$ for $i \in [k]$ such that we have halfspaces $Q_i = \{x \in \mathbb{R}^m \mid q_i^T x > b_i\}$. Then we see that

$$\begin{aligned} & y \in \bigcup_{i=1}^{k} Q_i \\ \Leftrightarrow \ & \exists j \in [k] : \ y \in Q_j = \{x \in \mathbb{R}^m \mid q_j^T x > b_j\} \\ \Leftrightarrow \ & \exists j \in [k] : \ q_j^T y - b_j > 0 \\ \Leftrightarrow \ & \max_{i \in [k]} \left(q_i^T y - b_i\right) > 0. \end{aligned}$$

The same holds for closed halfspaces $Q_i, i \in [k]$, if all inequalities ">" are replaced by their counterparts "$\geq$". Analogously, with open halfspaces Q_i as before and $Y = \bigcap_{i=1}^{k} Q_i$ the same can be shown with "min" instead of "max". For the case $Y = \bigcup_{i=1}^{k} Q_i$ we consider the following MIP:

$$\begin{aligned} \text{minimize} \quad & t \\ \text{s.t.} \quad & x \in X \\ & y = F(x) \\ & z_i = q_i^T y - b_i \qquad \forall i \in [k] \\ & t = \max\{z_1, \ldots, z_k\} \end{aligned} \tag{3}$$

Indeed, (3) is an MIP since the constraint $t = \max\{z_1, \ldots, z_k\}$ can be replaced by linear constraints using k additional binary variables as shown in [5,34]. In this case, we can also replace the constraint by $t \geq z_1, \ldots, t \geq z_k$.

Theorem 1 *Instance $\Pi = (X, Y, F)$, where $Y = \bigcup_{i=1}^{k} Q_i$ for some open halfspaces $Q_i = \{x \in \mathbb{R}^m \mid q_i^T x > b_i\}$, $q_i \in \mathbb{R}^m$ and $b_i \in \mathbb{R}$ for $i \in [k]$, is verifiable if and only if the optimum value of (3) is greater than zero.*

Proof Assume that Π is verifiable, i.e. $F(X) \subseteq Y = \bigcup_{i=1}^{k} Q_i$. Hence, for any $x \in X$ there exists $j \in [k]$ such that $y := f(x) \in Q_j$, i.e. $q_j^T y - b_j > 0$. It follows $t \geq z_j := q_j^T y - b_j > 0$ which implies the desired result since $x \in X$ was arbitrary. Remind that we regard optimum solutions of an MIP so it suffices to consider finitely many $x \in X$.

For the opposite direction, assume that the optimum value $\hat{t}$ of (3) fulfills $\hat{t} > 0$. Let $x \in X$ be arbitrary and $y = F(x)$. With $z_i = q_i^T y - b_i$ for $i \in [k]$ it holds $\max\{z_1, \ldots, z_k\} \geq \hat{t} > 0$ since $\hat{t}$ is optimal. In other words, there is $j \in [k]$ such that $q_j^T y - b_j = z_j > 0$ and thus $y \in Q_j \subseteq Y$. Since $x \in X$ was arbitrary, Π is verifiable. □

It works also for the case $Y = \bigcap_{i=1}^{k} Q_i$ by replacing "max" with "min" in (3), and similarly for closed halfspaces. In practice, the optimum value $\hat{t}$ of (3) will usually be significantly greater than zero if an instance is indeed verifiable. Clearly, it is not necessary to actually compute $\hat{t}$ in order to solve the verification problem as Bunel et al. [5] point out. If the dual bound of (3) is greater than zero, the instance is verifiable. We mainly use this formulation in our implementation. On the other hand, if the primal bound of (3) is lower than zero, we know that the corresponding instance of the verification problem is refutable as $\hat{t} < 0$ is already implied. However, this case has less relevance since primal solutions are usually only found by specialized heuristics which we describe in Sect. 6.

Besides, we note that the verification problem can be modelled as quadratic program. This formulation does not require any integer or binary variables as the nonlinear behavior of the ReLU activations is modelled by the quadratic objective function and an optimality condition. Let $\Pi = (X, Y, F)$ be a disjunction instance, i.e. $Y = \bigcup_{i=1}^{k} Q_i$ for open halfspaces Q_i, $i \in [k]$ and $k \in \mathbb{N}$. Let $((A_l, b_l))_{l=1}^{L}$ be the weights and biases corresponding to F. Here, L is the number of layers in the neural network and $N_0, \ldots, N_L$ are the numbers of neurons per layer. This implies $X \subseteq \mathbb{R}^{N_0}$ and $Y \subseteq \mathbb{R}^{N_L}$ and we can state the formulation:

$$
\begin{aligned}
\text{minimize} \quad & \sum_{l=1}^{L-1} x_l^T (x_l - A_l x_{l-1} - b_l) \\
& x_l \geq A_l x_{l-1} + b_l, \quad x_l \geq 0, \quad x_l \in \mathbb{R}^{N_l} \qquad \forall l \in [L-1] \\
& x_L = A_L x_{L-1} + b_L \\
& x_0 \in X, \quad x_L \in \mathbb{R}^{N_L} \setminus Y
\end{aligned} \tag{4}
$$

Theorem 2 *Instance Π is refutable if and only if the quadratic program (4) is feasible and the optimum value is zero. Otherwise Π is verifiable.*

Proof We first assume that Π is refutable so that we can find $x \in X$ with $F(x) \notin Y$. We set $x_0 := x, x_L := F(x) \in \mathbb{R}^{N_L} \setminus Y$ and for $l \in [L-1]$ we let $x_l := \mathrm{ReLU}(A_l x_{l-1} + b_l)$ which implies $x_l \geq 0$ and $x_l \geq A_l x_{l-1} + b_l$. Furthermore it is $x_L = A_L x_{L-1} + b_L$ and for each $l \in [l-1]$ we have for each $i \in [N_l]$ that either $[x_l]_i = 0$ or $[x_l]_i = [A_l x_{l-1} + b_l]_i$. Since $x_l \in \mathbb{R}^{N_l}$, this leads to the conclusion that $x_l^T (x_l - A_l x_{l-1} - b_l) = 0$ for all $l \in [L-1]$. Hence, the quadratic program (4) is feasible and its optimum value is zero.

On the other hand, if (4) is feasible and the optimum value is zero, we know that there is $x_0 \in X$, such that $F(x) = x_L \notin Y$ which means that Π is refutable. Indeed, it holds $F(x) = x_L$ since for all $l \in [L-1]$ we have $x_l \geq 0$ and $x_l \geq A_l x_{l-1} + b_l$, i.e. $x_l^T (x_l - A_l x_{l-1} - b_l) \geq 0$

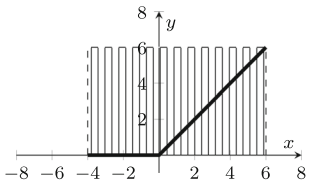

Fig. 1 Naive approximation of ReLU function in one dimension. Here we have lower bound -4 and upper bound 6 for the ReLU input variable x. The feasible domain of the ReLU output variable y is given by the solid black line for the actual ReLU function and by the shaded area for the naive approximation

for all $l \in [L-1]$. Hence we know $x_l^T(x_l - A_l x_{l-1} - b_l) = 0$ for all $l \in [L-1]$ as the objective value of (4) is zero, and it follows that $[x_l]_i[(x_l - A_l x_{l-1} - b_l)]_i = 0$ for all $i \in [N_l]$ and $l \in [L-1]$. Subsequently it holds $[x_l]_i = \text{ReLU}(A_l x_{l-1} - b_l)$ and thus we can conclude that $F(x) = x_L$. □

To evaluate this formulation computationally, we tried a plain implementation in SCIP [13]. Within a time limit of one hour, SCIP is not able to solve any of the disjunction instances in our SAT and UNSAT evaluation sets. Only very easy MNIST instances could be solved with this formulation.

Anderson et al. [2] present an ideal MIP formulation for ReLU constraints which can replace (1). It should be noted that the formulation is ideal for a single ReLU neuron but not for the whole neural network. As the formulation of Anderson et al. [2] has an exponential number of constraints, they also describe a separation routine that runs in linear time. This allows to strengthen formulation (1) by adding additional cuts to the LP relaxation, as obtained by the separation routine.

5 Approximations of ReLU neural networks

Solving the problem of neural network verification requires to model constraints of the form $y = \max\{0, x\}$ for all ReLU input variables x and corresponding ReLU output variables y of each layer. It is crucial to obtain tight bounds $l,\ u$ on the value of x before the application of the ReLU function. Especially, we regard the linear approximation of these constraints for a whole layer at once, an idea so far considered only briefly in [2,5,23].

Given an instance $\Pi = (X, Y, F)$ of the verification problem with $X \subset \mathbb{R}^n$, we will assume the availability of input bounds l_i, u_i with $i \in [n]$ for the components of X throughout this section (cf. Remark 3). All approximation methods that we present are executed layer by layer. Based on the input bounds, we compute bounds for the neurons in the following layer. This process is iterated until the last layer is reached, i.e. the output layer. Depending on the instance and the bound computation approach, it may be possible to prove that Π is verifiable using only these bounds for the output layer. Assume that we have a set A which approximates the neural network output $F(X)$, i.e. $F(X) \subseteq A$. In case that $A \subseteq Y$, we have thus shown that $F(X) \subseteq Y$, which means that Π is verifiable.

5.1 Basic approximation methods for bound computations in neural networks

The most simple approximation approach is naive interval arithmetic as used in [10,35]. Figure 1 provides a visual representation of this approximation, to which we will refer as

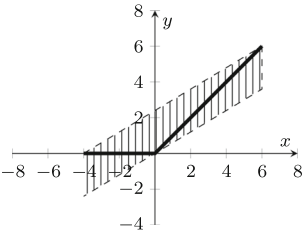

Fig. 2 Approximation of ReLU function in one dimension as proposed by [36]. Here we have lower bound -4 and upper bound 6 for the ReLU input variable x

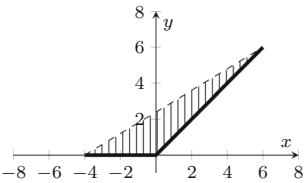

Fig. 3 Approximation of ReLU function in one dimension as proposed by Ehlers [10]. Here we have lower bound -4 and upper bound 6 for the ReLU input variable x

naive approximation. This simple approach mainly suffers from the fact that it assumes the independency of all predecessor neurons when computing a new bound. Therefore, the bounds computed with this method are so bad, that they only serve to solve tiny instances.

Wang et al. [35] use symbolic interval arithmetic to keep track on some of the neuron dependencies in order to compute better bounds. The idea is to keep a symbolic equation, based on the input values of the network, for each neuron. This symbolic approach can only provide better bounds if at least some of the ReLU activations can be fixed positively, i.e. $l \geq 0$. Otherwise, the symbolic interval arithmetic uses the same bounds as the naive method and computes new bounds in the same way.

To overcome this drawback, Wang et al. [36] improve the method by introducing a different approximation for the case $l < 0 < u$. The main idea is to maintain the symbolic dependencies also in this case. Though, the linear equation for the value of a ReLU neuron with input bounds $l < 0 < u$ cannot be kept. Instead a symbolic equations is introduced which provides a lower and upper bound for the neuron value. These symbolic bounds can then be propagated through the network and have the advantage that the dependency information partially remains. For the propagation of the symbolic equactions an approximation is used as visualized in Fig. 2.

Now we consider a linear approximation of ReLU constraints which was first proposed by Ehlers [10]. In fact, we will show that it is best possible in a certain sense, which we define in the following subsection. Given $x \in [l, u]$, where $l < 0 < u$, and $y = \max\{0, x\}$ it holds (i) $y \geq 0$, (ii) $y \geq x$, and (iii) $y \leq \frac{u(x-l)}{u-l}$. We graphically depict this approximation in Fig. 3 which in fact coincides with the linear relaxation of the MIP formulation (1) for ReLU constraints, see [2,5].

Of course, the linear approximation of Ehlers [10] remains valid, if either the constraint $y \geq 0$ or the constraint $y \geq x$ is removed. This enables the use of matrix multiplication (cf. Zhang et al. [42]) or static analyzers with abstract domains (cf. Singh et al. [30]) for the propagation of the inequations.

Another approximation method is proposed by Raghunathan et al. [23] in the context of robustness certification. It consists in an SDP relaxation for ReLU neural networks that acts simultaneously on all neurons of a layer.

5.2 Comparison of linear ReLU approximations

In general, one ReLU layer contains several neurons, and we are interested to compute an approximation of the output range of the layer. This approximated output range can then be regarded as input to the next layer. As we want to reach a quick propagation of the output ranges through the layers, it is important that the approximated output range is a polytope. This allows to compute neuron bounds quickly using linear programming. In the following, we develop a theoretical framework to analyse different linear approximations.

Definition 2 *(ReLU approximation)* Let $n, m \in \mathbb{N}$, $A, B \in \mathbb{R}^{m \times n}$, $c \in \mathbb{R}^n$, and $P \subset \mathbb{R}^n$ be a polytope. We say that

$$Q := \left\{ \begin{pmatrix} x \\ y \end{pmatrix} \in P \times \mathbb{R}^n \mid Ax + By \leq c \right\} \subset \mathbb{R}^{2n}$$

is a ReLU approximation (of P) if it holds that $(P \times \text{ReLU}(P)) \subseteq Q$. Q is called an *independent ReLU approximation*, if for all $j \in [m]$, there exists $i \in [n]$ such that $A_{jl} = B_{jl} = 0$ for all $l \in [n] \setminus \{i\}$. A polytope P is called *ReLU proper*, if for all $i \in [n]$ it holds

$$\min_{x \in P} x_i < 0 < \max_{x \in P} x_i.$$

The consideration of ReLU proper polytopes simplifies the formulation of statements, as fixed ReLU neurons are not regarded. If we apply the naive approximation to a ReLU proper polytope we obtain a box $[0, u_1] \times \ldots \times [0, u_n]$, where u_i is the upper bound for the corresponding variable. We see that this is an *independent ReLU approximation*. Let $A = 0 \in \mathbb{R}^{2n \times n}$ and for each $i \in [n]$, we add two rows to matrix B and vector c to enforce $0 \leq y_i \leq u_i$ for $i \in [n]$, i.e. $m = 2n$ for the m in Definition 2. These rows are $e_i^T\ y \leq u_i$ and $-e_i^T\ y \leq 0$, where e_i is the i-th unit vector in $\mathbb{R}^n$. Hence, we have exactly one non-zero coefficient in each row of B and only zero coefficients in A, so that the property holds. In passing we notice that the approximation proposed in Wang et al. [36] is an independent ReLU approximation, too.

Now we use our definition of a ReLU approximation for a more thorough investigation of the possibilities to approximate ReLU constraints. Within the restrictions of the definition, we would like to find matrices A, B and c for a ReLU proper polytope P, such that Q is as small as possible (with respect to inclusion). First, we will restrict our analysis to independent ReLU approximations and claim: the approximation proposed by Ehlers [10] is best possible among all independent ReLU approximations of a ReLU proper polytope. We define this approximation formally as a ReLU approximation in order to state the result in Theorem 3.

Definition 3 Let $P \subset \mathbb{R}^n$ be a ReLU proper polytope. The ReLU approximation of P corresponding to the approximation of Ehlers [10] will be denoted as Q_E. In detail, for $i \in [n]$, we set

$$A^{(i)} = \begin{bmatrix} 0 \\ e_i^T \\ \frac{u_i}{l_i - u_i} e_i^T \end{bmatrix},\ B^{(i)} = \begin{bmatrix} -e_i^T \\ -e_i^T \\ e_i^T \end{bmatrix} \text{ and } c^{(i)} = \begin{bmatrix} 0 \\ 0 \\ \frac{u_i l_i}{l_i - u_i} \end{bmatrix}.$$

For that, we use $l_i := \min_{x \in P} x_i$ and $u_i := \max_{x \in P} x_i$ and eventually define

$$A_E = \begin{bmatrix} A^{(1)} \\ \vdots \\ A^{(n)} \end{bmatrix},\ B_E = \begin{bmatrix} B^{(1)} \\ \vdots \\ B^{(n)} \end{bmatrix} \text{ and } c_E = \begin{bmatrix} c^{(1)} \\ \vdots \\ c^{(n)} \end{bmatrix}.$$

Thus we obtain

$$Q_E := \left\{ \begin{pmatrix} x \\ y \end{pmatrix} \in P \times \mathbb{R}^n \mid A_E x + B_E y \leq c_E \right\} \subset \mathbb{R}^{2n}.$$

Remark 5 Indeed, Q_E is an independent ReLU approximation. All rows of A and B are either 0 or a multiple of a transposed unit vector $e_i^T \in \mathbb{R}^n$. If the latter is the case, $i \in [n]$ is the same both in A and B when regarding the same row indices of A and B.

Theorem 3 *Let $P \subset \mathbb{R}^n$ be a ReLU proper polytope and Q_E be the approximation of P as in Definition 3. For any independent ReLU approximation Q of P it holds $Q_E \subseteq Q$.*

For the proof see "Appendix C". In the following section, we explain how the approximation of Ehlers [10] is used in [5,10] and discuss possibilities to speed up the computation to make this method more efficient. Then we present an approximation that is stronger than the one of Ehlers [10] and hence not independent in Sect. 5.4.

5.3 Efficient optimization based bound tightening for neural network verification

If we build the MIP model using some preliminary lower and upper bounds for each ReLU neuron, we can use the LP relaxation of the model to approximate the output values of the neural network. As in [5,10], we can also tighten the neuron bounds using the LP relaxation, which is identical to the approximation of Ehlers [10]. For each ReLU input variable x we compute an optimal solution of the LP relaxation for the objective functions x and $-x$. The optimum objective values hence give the new bounds for x in the neural network. In accordance with Gleixner et al. [14], we call this technique optimization based bound tightening (OBBT).

After the bound update, it is crucial to improve the MIP formulation (1). This allows to compute significantly tighter bounds in the next layer. Indeed, it is possible to build the approximation of the whole network only during the process of bound optimization. That means, each variable (corresponding to a neuron) is added separately to the relaxed MIP model such that the LP is always as small as possible. We regard the ideas of Gleixner et al. [14], who implemented an OBBT propagator in SCIP [13], to reduce the computational cost. Gleixner et al. [14] treat two main topics: First, they show how to generate and propagate Langrangian variable bounds (LVBs), and second, they propose methods for the acceleration of OBBT.

LVBs are valid inequalities with respect to the LP relaxations of mixed integer nonlinear problems (MINLP), which also includes LP relaxations of MIPs. Gleixner et al. [14] state that *LVBs can be viewed as a one-row relaxation of the given MINLP*, that provide *a local approximation of the effect of OBBT*. They are obtained as a by-product of the LP solutions which are computed during the execution of OBBT. Dual multipliers of the LP relaxation and an objective cutoff constraints are used to create an LVB. For the actual definition and more details see Gleixner et al. [14]. If we model the neural network verification problem as optimization problem as described in Sect. 4, we are only interested whether there exists a solution with objective value smaller than or equal to zero or not. Hence, we can safely cut off all solutions with an objective value greater than some $\varepsilon > 0$. For our experiments we set $\varepsilon := 0.1$ to have a sufficiently big margin to zero in order to prevent erroneous results.

The advantage of LVBs is that they can be propagated efficiently through a branch-and-bound tree, while the frequent application of OBBT requires a great computational effort for each branch-and-bound node that is processed. We see in our experiments that for some

instances the use of LVBs is able to speed up the solving process significantly. See Sect. 8.2 for an overview of the experiments.

Moreover, we consider ideas of Gleixner et al. [14] for accelerating the application of OBBT. One aspect is filtering of bounds which can hardly be improved by executing OBBT. Assume that y is the value of a variable, which is a candidate for the application of OBBT, in a feasible solution of the LP relaxation. Moreover, let $l \leq y \leq u$ be the bounds which are currently known for this variable. If then $y - l \leq \varepsilon$ or $u - y \leq \varepsilon$ for some $\varepsilon > 0$, *OBBT can strengthen the corresponding bound by at most* ε, as Gleixner et al. [14] point out. Yet, initial experiments showed, that usually almost all bounds can be improved significantly by OBBT, so that filtering bounds is not useful for verification of neural networks. Another aspect is the order of the variables for which OBBT is executed. As OBBT is executed layer by layer in our case, the order of variables can only be changed within each layer. However, the various strategies of Gleixner et al. [14] did not show any advantage over a simple fixed order in our computational experiments, see Rössig [24] for details.

Eventually, we consider another approach for bound computations in neural networks that is also a form of OBBT. Instead of using the LP relaxation to compute bounds, it is also possible to employ the exact MIP model and compute bounds for the neurons with OBBT. Computing the neuron bounds using the MIP formulation instead of the LP relaxation leads to strongly improved bounds. Although not all MIPs are solved to optimality, clear improvements of the corresponding bounds can be reached within a time limit of few seconds per MIP. These improvements render it possible to solve also relevant instances without specialized branching rules for neural network verification, however the bound computations take a lot of time.

5.4 Optimization based relaxation tightening for two variables

In general we regard neural network layers that feature ReLU activations for all neurons of the layer. Hence, we investigate in more detail how the ReLU function behaves in higher dimensions, i.e. if the ReLU function is applied componentwise to layers with several neurons. The following theorem can be found in Xiang et al. [41] as Corollary 1:

Theorem 4 *For a polytope* $P \subset \mathbb{R}^n$, $\mathrm{ReLU}(P)$ *is a finite union of polytopes.*

Hence we see that the best possible convex approximation $\mathrm{conv}(\mathrm{ReLU}(P))$ of $\mathrm{ReLU}(P)$ is the convex hull of the union of polytopes in Theorem 4. We investigate a simple example to see how the approximation of Ehlers [10] differs from this best possible convex approximation. We consider a toy example as depicted in Fig. 8 of the "Appendix". Figure 4 shows the feasible input polytope of the ReLU layer and the corresponding ReLU image. The same ReLU image can be seen in Fig. 5, replenished with a depiction of its convex hull, the approximation of Ehlers [10] and the naive approximation. Figure 5 clearly shows that even for only two variables the convex hull of the ReLU image is strictly smaller compared to the approximation of Ehlers [10]. It seems appealing to find an improved approximation of the ReLU image closer to the convex hull, which is the best possible convex approximation.

Subsequently, we propose an efficient method which can strengthen the approximation of Ehlers [10] by considering at least pairs of two neurons jointly. This new ReLU approximation is not independent (cf. Definition 2). The depiction in Fig. 5 shows, that in this situation we could actually add one inequality and would improve the approximation to be exactly the convex hull of the ReLU image. This inequality is induced by the connecting segment between the vertices of the convex hull that maximize y_1 or y_2, respectively. Of course, we

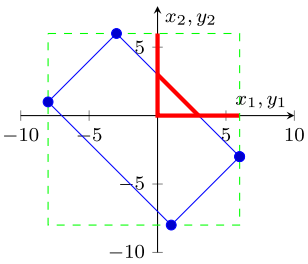

Fig. 4 Example for the feasible set before (blue polytope) and after (set enclosed by red lines) application of the ReLU function

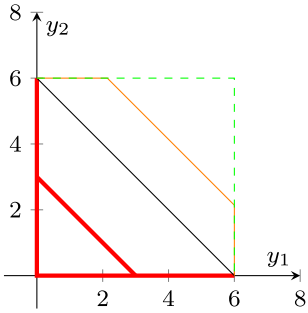

Fig. 5 The red lines enclose the ReLU image and the black line (with the coordinate axes) indicates the convex hull of this ReLU image. The approximation of Ehlers [10] is bounded by the orange segments, the naive approximation by the green ones (and coordinate axes)

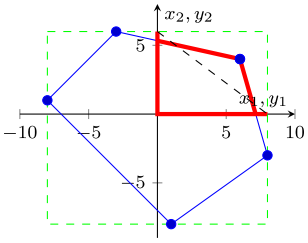

Fig. 6 Feasible set before (blue) and after ReLU application (red) for a different input polytope

cannot make this inequality tighter, since otherwise feasible points of the ReLU image would be cut off. Though, the segment between the vertices that maximize y_1 or y_2, respectively, does not always induce a valid inequality as we show in the following example. Figure 6 shows a polytope of feasible x_1, x_2 values and the corresponding ReLU image of feasible values for y_1 and y_2, such that $y_1 = \max\{0, x_1\}$ and $y_2 = \max\{0, x_2\}$. The polytope is two dimensional, but can also be considered as embedded image of a higher dimensional polytope which is projected onto its variables x_1 and x_2. These two variables correspond to two neurons in one layer of a ReLU neural network. The dimension of the original polytope is then the number of all neurons in that layer. It should be noted that we use these projections to $\mathbb{R}^2$ only for the visualization of our method. The goal of our method is to obtain a tighter approximation without computing projections of higher dimensional polytopes. In Fig. 6, the segment (dashed line) between the vertices that maximize y_1 or y_2, respectively, does not induce a valid inequality with respect to the ReLU image.

Now the idea is to add an inequality to the model which partly cuts off the polytope resulting from the approximation of Ehlers [10], but leaves the ReLU image intact. The cut is parallel to the segment between the vertices that maximize y_1 or y_2, respectively. Depending on the situation, these vertices will either meet the inequality with equality or not. Figure 7 depicts this inequality and shows that adding this constraint considerably improves the approximation of the convex hull. In the following we describe how this constraint can be computed. A linear

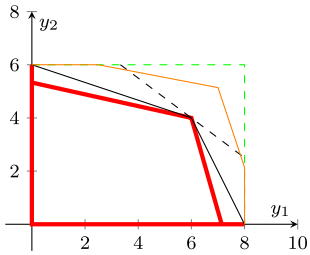

Fig. 7 Here we see the ReLU image of the polytope depicted in Fig. 6 colored in red, its convex hull in black, the approximation of Ehlers [10] in orange, the inequality which we want to introduce as a black dashed line and the constraints of the naive approximation as a green dashed line. All sets are limited by the coordinate axes

approximation of the ReLU neural network in question serves as a basis. Naturally, we can use the LP relaxation (which corresponds to the approximation of Ehlers [10]) if the verification problem is formulated as an MIP.

Assume we want to tighten the approximation for the ReLU output variables y_1 and y_2 which correspond to ReLU input variables x_1 and x_2. All of these variables are contained in the LP relaxation of the neural network. In the final solution it must hold $y_1 = \max\{0, x_1\}$ and $y_2 = \max\{0, x_2\}$ due to the ReLU constraints. Let $\hat{a}$ and $\hat{b}$ be the optimum solutions when maximizing x_1 or x_2, respectively, in the current LP relaxation. Then we write $\hat{a}_1$ and $\hat{a}_2$ for the values of the variables x_1 and x_2 in the solution $\hat{a}$. Analogously we write $\hat{b}_1$ and $\hat{b}_2$ for the corresponding variable values in solution $\hat{b}$. It should be noted that these LP solutions are computed during the execution of OBBT, and can therefore be obtained at no additional cost. Obviously it holds $\hat{a}_1 \geq \hat{b}_1$ and $\hat{b}_2 \geq \hat{a}_2$ due to the choice of objective functions. Now we define $a_1 := \max\{0, \hat{a}_1\}$ and analogously a_2, b_1 and b_2. We compute new objective coefficients as $c_1 := b_2 - a_2$ and $c_2 := a_1 - b_1$, i.e. $c_1, c_2 \geq 0$. The latter holds due to the fact that $\alpha \geq \beta$ implies $\max\{0, \alpha\} \geq \max\{0, \beta\}$ for $\alpha, \beta \in \mathbb{R}$. Again, we solve an LP using the current relaxation and maximize the objective function $c_1 x_1 + c_2 x_2$. We denote the optimum objective value as γ and compute $\delta := c_1 a_1 + c_2 a_2$. After this computation we can strengthen the LP relaxation by adding the constraint

$$c_1 y_1 + c_2 y_2 \leq \max\{\gamma, \delta\}. \tag{5}$$

Theorem 5 *Constraint (5) is a valid inequality with respect to the ReLU image corresponding to y_1 and y_2. That means, constraint (5) can strengthen the LP relaxation of our MIP for the verification problem but cannot cut off any feasible solution.*

Proof We remind that it holds $y_1 = \max\{0, x_1\}$, $y_2 = \max\{0, x_2\}$ due to the ReLU constraints, and $a_1, a_2, b_1, b_2, c_1, c_2 \geq 0$, hence $\delta \geq 0$. That means, if $(y_1, y_2) = (0, 0)$ we have $c_1 y_1 + c_2 y_2 = 0 \leq \delta$. If $(y_1, y_2) = (x_1, 0)$ it holds $x_1 \leq a_1$ and hence $c_1 y_1 + c_2 y_2 = c_1 x_1 + 0 \leq c_1 a_1 + c_2 a_2 = \delta$. On the other hand, the case $(y_1, y_2) = (0, x_2)$ implies $x_2 \leq b_2$ and subsequently we see $c_1 y_1 + c_2 y_2 = 0 + c_2 x_2 \leq c_1 b_1 + c_2 b_2 = \delta$. Otherwise it holds $(y_1, y_2) = (x_1, x_2)$ which implies $c_1 y_1 + c_2 y_2 \leq \gamma$ and we can conclude the proof. □

Thus, the approximation of Ehlers [10] can be improved by adding constraints of type (5) to the LP relaxation of the model. Like this, we obtain a ReLU approximation which is not independent. Although we have to solve only one LP per pair of neurons, applying this method to all possible pairs of neurons would lead to an immense computational cost. Therefore, we select only some pairs of neurons for which it is likely to significantly strengthen the LP relaxation by adding the new inequality to our model. Though, our selection strategy as laid out in Rössig [24] was not able to outperform a baseline selection strategy, which

selects neurons in a fixed, predetermined order. Yet, this technique, which we abbreviate as OBBT2, can significantly strenghten the LP relaxation and reduce the number of nodes in the branch-and-bound tree (see Table 10 in the "Appendix").

6 Primal heuristics

For the problem of neural network verification the use of primal heuristics lies in the quick falsification of incorrect properties. Surprisingly, even a trivial heuristic, which only performs random sampling within the set of feasible inputs, can often find counterexamples to incorrect properties quickly in contrast to standard MIP heuristics.

The idea of the random sampling heuristic as introduced by Bunel et al. [5]) is plain and simple: Given an instance $\Pi = (X, Y, F)$ of the verification problem, we randomly pick $x \in X$ and check whether $F(x) \in Y$. In case that $F(x) \notin Y$, we know that Π is refutable. Moreover, using the MIP formulation as optimization problem, the input vector x is also useful if it leads to a decrease of the primal bound, since this may help to tighten neuron bounds. In general it is not trivial to obtain $x \in X$, if $X \subset \mathbb{R}^n$ is an arbitrary polytope. However, as mentioned in Remark 3, many of the instances we regard feature a polytope X which is actually a box. In this case, we simply pick $x_i \in [l_i, u_i]$ uniformly at random for $i \in [n]$, where l_i, u_i are the bounds of X for each component. This is performed similarly in [5,36]. Otherwise, if X is not a box, we solve an LP to obtain $x \in X$ using a random objective function.

We propose another heuristic that can be used in addition to the random sampling heuristic. It is based on the local search proposed by Dutta et al. [8] for output range analysis of ReLU neural networks. Though, we omit the use of gradient information and fit the heuristic more naturally into the framework of MIP solving. The main idea is to fix all neurons in one of their phases, such that the optimization variant of neural network verification consists only in solving a linear program. We start with a feasible input $x_0 \in X$ for the neural network and use forward propagation to compute the values of all neurons in the network. Then, for each ReLU neuron, we fix the binary variable d in (1) to zero or one, corresponding to the phase of the neuron that is determined by propagating x_0 through the network. Furthermore, the binary variables in the formulation of the maximum function for objective variable t are also fixed, such that $t = \max\{z_1, \ldots, z_k\}$. With all binary variables fixed, the MIP as described in Sect. 4 becomes an LP.

This LP is minimized with respect to variable t as objective function. After the first minimization LP has been solved, we choose a ReLU input variable $\bar{x}$ (corresponding to one ReLU neuron) of value zero if possible. For this variable, we switch the fixed value of the corresponding binary variable $\bar{d}$ from zero to one or vice versa. Then we optimize again and obtain a new input vector $\hat{x}_0 \in X$ for the neural network. After that, we switch the fixing of another binary variable, whose corresponding ReLU input variable has value 0 in the solution. This process is iterated until we find a feasible counterexample, i.e. the optimal value of the LP is smaller than zero, or we reach a predefined iteration limit. In case that none of the ReLU input variables is equal to zero, we have to abort the procedure. It is easy to see that switching the fixings of the binary variables as described, can only reduce the objective value of the optimum LP solution.

In the following we describe, how we combine our LP based heuristic with the random sampling heuristic. First we use the random sampling heuristic to find an input vector $x_0 \in X$. The random sampling process and forward propagation are very fast, and therefore we try

many (e.g. 1000) random inputs to find an input $x_0 \in X$. Out of all sampled input vectors, we select $x_0 \in X$ such that it corresponds to the lowest value of objective variable t. The hope is that x_0 can be converted into an actual counterexample by computing a new input vector $\hat{x}_0$. This is given by the optimum LP solution after some ReLU phase switches as described. Instance Π is shown to be indeed refutable, if the value of t is below zero in this optimum LP solution.

Of course, both heuristics can be applied several times throughout the solving process in a branch-and-bound tree which we enable in our implementation. Our experimental evaluation shows that the LP based heuristic works quite successfully. In fact, the mean runtime on our evaluation set of SAT instances, as mentioned in Sect. 8, drops from 330.1 to 71.7 seconds if our LP based heuristic is employed. On the other hand, the mean runtime on our evaluation set of UNSAT instances increases only slightly from 915.3 to 943.7 seconds due to the application of our LP based heuristic.

7 Branching for neural network verification

The verification problem can be solved with a generic branch-and-bound approach as described by Bunel et al. [5]. If the problem is solved as an MIP, specific branching rules for neural network verification can be integrated into the MIP solving process to strongly speed up the process. Initial bounds are necessary for the formulation of the verification problem as an MIP model and can be obtained by one of the approximation methods as introduced in Sect. 5. Many relevant instances of the verification problem cannot be solved if an approximation of the network is computed only once. Specific branching rules can be used to split an instance into simpler ones which can be approximated better. One option is to split the set of feasible input vectors for an instance of the verification problem as in [5,35].

Given an instance $\Pi = (X, Y, F)$ of the verification problem, the design of the domain branching rule is based on the assumption that X is a box. However, the branching rule can also be applied if X is not a box. We assume the existence of bounds $l_i \leq x_i \leq u_i$ for all $x \in X \subset \mathbb{R}^n$ and $i \in [n]$, cf. Remark 3. In case that X is a box, it holds $X = [l_1, u_1] \times \ldots \times [l_n, u_n]$, otherwise $X \subseteq [l_1, u_1] \times \ldots \times [l_n, u_n]$. Bunel et al. [5] propose to select $j \in [n]$ and split the domain of variable x_j to subdomains $[l_j, \frac{u_j+l_j}{2}]$ and $[\frac{u_j+l_j}{2}, u_j]$.The domains of all other variables $x_i, i \in [n] \backslash \{j\}$ are left unchanged, so that we obtain two sub-instances with smaller input domains.

The selection of the branching variable is very important for the performance of the branching rule, cf. [5]. In Bunel et al. [4], the selection depends on the depth in the branch-and-bound tree and follows a fixed order. Bunel et al. [5] implement another selection rule, based on the approximation method of Wong and Kolter [38]. In our implementation we mainly use a selection rule "gradient" which is quite similar to the one used in Wang et al. [35]. For that, we extend the neural network F to another one $\tilde{F}$, which encodes also the properties that shall be verified. It has output dimension one, and for a fixed input $x \in X$, the output is the same as the value of the objective variable t in the MIP formulation as optimization problem (3). We use a max-pooling layer to model the computation of the maximum in (3) in the neural network $\tilde{F}$ and refer to Bunel et al. [5] for more details on the construction. We compute the gradient of $\tilde{F}$ at the input vectors $x_1 = (l_1, \ldots, l_n)$, $x_2 = (\frac{u_1+l_1}{2}, \ldots, \frac{u_n+l_n}{2})$, and $x_3 = (u_1, \ldots, u_n)$ and let $g := \nabla\tilde{F}(x_1) + \nabla\tilde{F}(x_2) + \nabla\tilde{F}(x_3) \in \mathbb{R}^n$. For $i \in [n]$ we compute $z_i := |g_i| \cdot (u_i - l_i)$ and choose the branching variable $j \in [n]$ such that $z_j = \max\{z_1, \ldots, z_n\}$. The intuition is that verifiability of the instance depends mainly on

the values of input neurons with a high (averaged) absolute gradient value and sufficiently big input range.

Another natural possibility is to perform branching over the two phases of a ReLU neuron which is used in [6,10,17,36]. If the verification problem is formulated as an MIP, this corresponds exactly to branching over the binary variables which correspond to the ReLU neurons. Given a ReLU input variable x and the corresponding output variable y, we can branch the constraint $y = \max\{0, x\}$ into two cases: either $x \leq 0$, $y = 0$, or $x > 0$, $y = x$. The main question is how to select the branching variables. As in Cheng et al. [6], we prioritize ReLU neurons which are located in the front of the neural network in the selection rule "standard". Similarly to Wang et al. [36], we implement a selection rule "gradient" that picks ReLU neurons which have a large gradient with respect to the outputs of the network. Though, in our experiments the rule "standard" performed better.

8 Computational evaluation

In this section we discuss computational results for various experiments that we conducted on a diverse set of test instances which we shortly present in the following. We empirically compare several methods for the computation of neuron bounds. Furthermore, we report computational results for various configurations of our solver and compare it with the programs of Bunel et al. [5], Wang et al. [36], and Katz et al. [17].

As we are not aware of any publicly available instances $\Pi = (X, Y, F)$ of the verification problem where X is not a box (cf. Remark 4), we define such instances to show the capabilities of our solving model. As a basis we use the neural networks of the ACAS Xu system, which were used by Katz et al. [17] to create instances of the verification problem. These are described in "Appendix A". The ACAS Xu system is designed to prevent collisions of (autonomous) aircrafts. We also perform our evaluations on the original instances published by Katz et al. [17]. Furthermore, we use test instances with neural networks that are trained on the well known MNIST [18] handwritten digit data set. In fact, we use two of the trained neural networks published by Wang et al. [36] and verify robustness of classifications using the L_∞ norm. Each input neuron may have a value between 0 and 255 and we investigate four different perturbation radii (1, 5, 10, and 20). These networks have two layers, each of them with 24 or 512 neurons, respectively. The smaller one reaches an accuracy of 96.59 %, the one with 512 neurons per layer 98.27 %. In contrast to the neural networks of the ACAS Xu system with input dimension five, the input dimension of the MNIST networks is 784. This difference is especially interesting with respect to the performance of input domain branching as presented in Sect. 7. To evaluate various settings of our solving model, we use two evaluation subsets that contain a diverse selection of all the instances. One contains 13 SAT instances and the other one 23 UNSAT instances. In general, we compute average values for runtime and number of solving nodes as shifted geometric mean which reduces the sensitiviy to outliers, cf. [1,16].

8.1 Empirical comparison of bound computation approaches

We use our evaluation set of UNSAT instances for a numeric comparison of the bounds which are obtained by various bound computation approaches. For each instance $\Pi = (X, Y, F)$ we compute lower and upper bounds $[l_i, u_i]$ for all neurons $i \in [N]$ (before application of the ReLU function), based on the feasible input domain X. The bounds are computed

layerwise from the first to the last layer and branching is not applied. Then we compute the shifted geometric mean of $\{u_1 - l_1, \ldots, u_N - l_N\}$ with shift value 1 for each instance. The mean value indicates how good the corresponding bound computation approach is. Clearly, it is desirable that the difference $u_i - l_i$ is as small as possible for all neurons $i \in [N]$, and we use the mean value to compare the different approximation methods. See Table 8 in the "Appendix" for detailed results, here we report the overall mean.

As a second measure we report how many neurons can be fixed in their phase by the respective bound computation method. Detailed results on this can be found in Table 9 in the "Appendix".

The results in Tables 1 and 2 show that the best bounds are computed by OBBT on the MIP model. While OBBT2 computes better bounds compared to OBBT on the LP relaxation, the improvements are unfortunately quite minor. We also see that the approaches of Wang et al. [35,36] are clearly superior to naive interval arithmetic. Yet, they are not competitive with OBBT if regarding only the quality of the computed bounds.

8.2 Comparison of different techniques in our model

In this section we provide an overview of the performance of our solving model in various configurations. The experiments in this section are run for the instances of our evaluation set of UNSAT instances. All results are obtained on cluster nodes with Intel Xeon CPUs E5-2670 which have a clock rate of 2.5 GHz. Each experiment is run exlusively on one cluster node and a memory limit of 32 GB is set. In general we set a time limit of 7200 seconds. If the time limit is hit during the solving process, we assume the time limit as runtime for the corresponding instance.

We report results both for the whole UNSAT test set, and separately for the ACAS and MNIST based instances. Comparing ACAS and MNIST based instances, the main differences are the number of input neurons (5 vs. 784) and the number of layers in the neural networks (6 vs. 2).

For the experiments in this section we use a baseline configuration "no_heur_base" of our solving model. In this configuration, domain branching (with selection rule "gradient") is used up to a depth of 20 in the branch-and-bound tree. It should be noted that this depth is usually not exceeded if domain branching is used. Furthermore, OBBT is applied to the LP relaxation at each solving node up to a depth of 20 in the branch-and-bound tree. The primal heuristic is enabled only at the root node. Further techniques (e.g. our separator based on the work of Anderson et al. [2]) are not applied.

In Table 3 we report mean runtimes on the UNSAT test set for several different configurations of our solving model. Here we try to give an impression of the differences between the configurations which we tested. We are mostly interested in the computation of the dual bound by various methods and therefore the primal heuristic is only employed at the root node. Table 3 shows that there is a clear difference between solving MNIST and ACAS instances. In fact, the configuration "no_heur_base_genv" is the fastest in total because it performs well on both types of instances. However, it is not the best configuration for either of the two subsets (although close to the best configuration for ACAS instances). The best configurations with respect to the MNIST instances ("no_heur_relu_genv" with respect to number of timeouts, "mnist_base" with respect to mean runtime) do not perform well on the ACAS instances. The configurations in Table 3 are sorted by the total number of timeouts. In the following, we give short descriptions of the configurations in this order. The configuration "no_heur_base_genv" corresponds to the baseline configuration with the additional use of Langrangian variable

Table 1 Averaged differences between neuron bounds in our UNSAT test set for different bound computation methods

	Naive IA	Sym. IA	Symbolic equations	OBBT LP	OBBT2 k = 2, l = 5	OBBT2 k = 10, l = 10	OBBT MIP
mean value	156.54	98.30	60.23	24.83	24.61	23.98	10.38

IA stands for interval arithmetic; in case of OBBT2 we compare two different settings

Table 2 Average of the number of fixed neurons per instance in our UNSAT test set for different bound computation methods

	Naive IA	Sym. IA	Symbolic equations	OBBT LP	OBBT2 k = 2, l = 5	OBBT2 k = 10, l = 10	OBBT MIP
Mean value	84.35	107.61	124.87	151.30	151.57	151.74	189.30

Table 3 Runtime results on the UNSAT test set using the formulation as optimization problem and various configurations

Subset (number of instances)	All (23)		ACAS (18)		MNIST (5)	
Configuration	Time	Timeouts	Time	Timeouts	Time	Timeouts
no_heur_base_genv	586.3	3	765.4	2	221.6	1
no_heur_sepa0_freq5	839.2	5	757.4	2	1213.1	3
no_heur_base	854.9	5	771.2	2	1237.3	3
no_heur_base_obbt2_nosort	888.0	5	816.8	2	1199.1	3
no_heur_base_mip	1556.5	6	1578.2	3	1480.9	3
no_heur_relu_genv	750.3	7	1180.0	7	141.6	0
no_heur_relu	970.2	9	1334.2	8	304.5	1
mnist_base	2753.5	19	7200.0	18	77.5	1

bounds (LVBs) as described in Sect. 5.3. Obviously, the LVBs are quite beneficial for solving MNIST instances. The configuration "no_heur_sepa0_freq5" sometimes calls an additional separator, as suggested by Anderson et al. [2]. Next in Table 3 follows our baseline configuration which is hence quite good already. Indeed, "no_heur_base_obbt2_nosort" is the best of our configurations that use OBBT2 and it has a higher mean runtime. Regarding the mean runtime compared to the number of timeouts, the configuration "no_heur_base_mip" is not in line with the other configurations. Indeed, this configuration applies OBBT to the MIP model in the beginning of the solving process to compute initial neurons bounds. This comes with a high computational cost, also for rather easy instances. However, for more difficult instances, this stratey is not that bad as indicated by the relatively low number of timeouts.

In the configuration "no_heur_relu_genv", ReLU branching is combined with OBBT on the LP relaxation and the creation of LVBs. Most notably, this configuration is the only one that solves all MNIST instances in the evaluation set within the time limit. Though, the performance on the ACAS based instances is rather mediocre. Due to the low number of input neurons of the ACAS neural networks, input domain branching is more efficient for these instances. The same holds for the configuration "no_heur_relu" which does not include the LVBs. Eventually, in the configuration "mnist_base" the solving process is limited to the application of standard MIP techniques. Notably enough, this configuration has the lowest mean runtime on the subset of MNIST instances. On the other hand, it times out on all ACAS instances. This vast difference can be attributed to the different number of layers in the neural networks (two for MNIST, six for ACAS), as deeper networks are more difficult to approximate.

8.3 Comparison with other solvers

In this section we provide detailed results of computational experiments which we conducted to investigate the performance of different solvers for neural network verification. Besides our own solving model, which we regard in various configurations, we include the programs of Bunel et al. [5], Wang et al. [36], and Reluplex by Katz et al. [17]. The work of Wang et al. [36] is implemented in two solvers, Neurify and ReluVal. ReluVal is used on the ACAS instances whereas Neurify is applied to the MNIST instances. We always check whether any alleged counterexample presented by some solver is indeed a feasible counterexample for the corresponding instance. In general, we perform these checks with an absolute numerical

Table 4 Mean runtime on ACAS properties 1, 2, 3, 4, and 8

	ReluVal	Adv	NonOpt	Joint	BaB	Reluplex
Mean runtime in seconds	5.7	3.1	34.1	34.1	67.5	669.0

tolerance of 10^{-8}. Several counterexamples are only feasible though, if a higher numerical tolerance is allowed, which we report in that case.

Each experiment is run exclusively on one cluster node with an Intel Xeon Gold 5122 CPU which operates at a clock rate of 3.6 GHz, and we set a memory limit of 32 GB. Besides the program names *ReluVal*, *Neurify* and *Reluplex*, we use the following terms to denote the various solving models. *Adv* refers to *ReluVal* or *Neurify* using their adversary check mode which is focused on finding counterexamples. *BaB* denotes the branch-and-bound method of Bunel et al. [5]. Eventually, we use *NonOpt* to describe that our solving model is used with the formulation of the verification problem as feasibility problem. *Joint* refers to our solving model using the formulation as optimization problem, which can solve conjunction instances (cf. Remark 4). *Separate* indicates that we solve the verification problem as optimization problem, but conjunction instances are split into several disjunction instances as explained in Remark 4. Conjunction instances have to be splitted likewise for BaB and Reluplex. Based on the results of our evaluation experiments in Sect. 8.2, we choose the configuration "no_heur_sepa0_freq5" as the best for the ACAS instances. Though, in order for a good performance on refutable instances, we do adapt this configuration to execute the primal heuristic also locally up to a depth of eight in the branch-and-bound tree.

Table 4 contains the runtime results for all ACAS instances of Properties 1, 2, 3, 4, and 8 as defined by Katz et al. [17]. It should be noted that these are all disjunction instances. The mean runtime of Reluplex is at least one order of magnitude larger than the mean runtimes of all other solvers. ReluVal is clearly superior to all other solvers, especially if its adversary check mode is applied. We see that our solving model, which uses SCIP [13] to strongly integrate the bound computations into a MIP framework, performs significantly better than the rather similar approach of Bunel et al. [5].

A similar picture of the performance of the various solvers can be seen in Table 5, which shows the results on the ACAS instances of Properties 5, 6, 7, 9, and 10. Remind that we underestimate the runtime of instances for which the solving process is stopped due to the time limit.

In Tables 6 and 7 we report runtimes for the MNIST instances. We remind that all of these are conjunction instances by definition. For our solving model we use the configuration "mnist_base" as presented in Sect. 8.2. The instances which are based on neural networks with two layers of 24 neurons are solved very quickly by most solvers, as can be seen in Table 6. Though, domain branching is apparently not a good strategy to solve these instances, which feature 784 input neurons. This is shown by the high number of timeouts of the solving method BaB [5] on the MNIST instances in Table 6.

Clearly, the MNIST instances with 512 neurons per layer are much more challenging. In Table 7 we see that our solving model performs quite good when the approach *Joint* is taken, i.e. conjunction instances are solved directly as one optimization problem. In several cases Neurify aborts the solving process with no result. For the corresponding instances we assume the time limit of 7200 seconds as runtime in order to compute the mean runtime. If the conjunction instances are split into separate disjunction instances, our solving model mostly fails to find counterexamples within the time limit (see *NonOpt* and *Separate*). This can be explained by the fact that the instances, which are obtained after splitting, are solved

Table 5 Runtimes on ACAS properties

Instance	Result	Reluval	Adv	NonOpt	Joint	Separate	BaB	Reluplex
property5	UNSAT	12.3	9.1	3922.2	timelimit	4104.8	timelimit	4883.4
property6a	UNSAT	3.3	3.2	7078.8	6248.2	3745.0	timelimit	timelimit
property6b	UNSAT	1.6	1.5	5935.9	4207.5	6179.1	timelimit	timelimit
property7	SAT	memlimit	1099.1	timelimit	3012.7	3012.7	timelimit	timelimit
property9	UNSAT	411.5	184.5	5672.0	timelimit	5921.1	4214.8	timelimit
property10	UNSAT	1.3	1.1	timelimit	3268.3	5335.3	4567.3	5516.3
sh. geo. mean		60.0	33.6	6053.9	4875.5	4564.6	6107.5	6456.0

If Reluval is run in normal check mode, it terminates prematurely on Property 7 due to a lack of memory. We use the time limit of 7200 seconds for the mean computation in this case

Table 6 Runtimes on the MNIST 24 data set

Instance	Result	Neurify	Adv	NonOpt	Joint	Separate	BaB	Reluplex
mnist_24_image11_1	UNSAT	0.4	0.4	10.6	8.1	28.0	25.7	0.5
mnist_24_image11_10	UNSAT	2.0	2.1	31.0	15.5	35.3	timelimit	4824.5
mnist_24_image11_20	SAT	0.4	0.5	6.0	7.5	7.8	timelimit	3437.6
mnist_24_image11_5	UNSAT	0.4	0.4	860.7	6.1	31.6	22.4	189.9
mnist_24_image2_1	UNSAT	0.4	0.4	25.1	3.8	27.1	21.5	1.6
mnist_24_image2_10	SAT	0.4	0.5	2.6	2.1	2.0	timelimit	24.7
mnist_24_image2_20	SAT	0.4	0.5	2.7	2.0	2.0	timelimit	19.2
mnist_24_image2_5	UNSAT	0.4	0.4	31.6	13.6	36.8	timelimit	timelimit
mnist_24_image4_1	UNSAT	0.4	0.6	12.4	2.5	25.8	22.3	0.9
mnist_24_image4_10	SAT	0.4	0.5	7.9	2.0	2.1	timelimit	70.5
mnist_24_image4_20	SAT	0.5	0.6	2.7	2.3	5.3	timelimit	11.9
mnist_24_image4_5	SAT	0.4	0.5	23.1	2.0	24.0	timelimit	536.1
shifted geo. mean		0.4	0.5	18.8	4.4	14.1	1006.3	86.1

The last number of the instance name is the perturbation radius. In contrast to all other solvers, Neurify reports SAT for instance mnist_24_image11_10. However, all counterexamples that Neurify produces, are only valid when applying a large numerical tolerance of 10^{-3}. We regard the instance mnist_24_image11_10 as verifiable and exclude it from the mean computation. Counterexamples produced by Reluplex are valid with a numerical tolerance of 10^{-5}

sequentially one after the other. However, if a verifiable instances is processed first, this may already lead to a timeout. Reluplex is only able to solve two of the easiest instances within the time limit, and BaB also performs considerably worse than Neurify and our solving model. Though for all MNIST instances, it should be noted that the counterexamples which are produced by Neurify are only feasible if one applies a large numerical tolerance of 10^{-3}. It is clear to see, that the instances with the lowest perturbation radius of 1 (in L_∞ norm) are easy to verify. On the other hand, for the instances with a high perturbation radius of 10 or 20, counterexamples are found in most cases. Apparently, the perturbation radius 5 poses the biggest difficulties for the solvers, as the corresponding instances are probably on the borderline between SAT and UNSAT.

In the "Appendix" we report our self defined instances where the input polytope X is not a box and computational results of our solving model on these, as the other solvers cannot process these instances. See Table 12 for the results which are obtained with the formulation as optimization problem, and Table 13 for the results corresponding to the formulation as feasibility problem.

9 Conclusions

Our solving model shows a solid performance in all categories of our benchmark set. This highlights the success of our approach, which combines MIP solving, using the solver SCIP [13], with specialized bound computation and branching techniques. Especially, we can solve instances with general polytopes as input domains which is not possible with other verification algorithms [5,17,36]. Moreover, our solving model clearly outperforms the solvers of Bunel et al. [5] and Katz et al. [17]. Subsequently, Reluplex cannot be regarded as a state-of-the-art solver for neural network verification anymore. While ReluVal and Neurify from Wang et al. [36] show impressive runtime results for most instances, they rely primarily on branching. For big instances, this can become problematic due to limited numerical accuracy and memory capacity.

Moreover, we developed a theoretical framework for the comparison of linear approximation methods and presented the novel approximation technique OBBT2 which is able to improve the linear relaxation of a ReLU neural network. Besides that, we showed how the local search procedure of Dutta et al. [8] can be implemented differently in an MIP solving context such that it serves as a primal heuristic. Additionally, we described a novel formulation of the verification problem as quadratic program. All newly proposed techniques were evaluated computationally within our newly implemented solving model.

Although our implementation is limited to neural networks with ReLU activation function, the approach could be easily extended to work with other piecewise-linear activation functions. Examples for that are the leaky ReLU function or max-pooling layers. In addition, the great flexibility of our solving model allows the integration of further techniques such that future improvements may render it even more efficient.

In conclusion, it can be said that many challenges remain in the field of neural network verification. The further scalability of current verification approaches is still an open task. Besides, new approaches for the falsification of incorrect properties would be highly interesting. Although we present a new heuristic for that (based on ideas of Dutta et al. [8]), better algorithms could probably be developed.

Acknowledgements Open Access funding enabled and organized by Projekt DEAL. Ansgar Rössig would like to thank Carmeq GmbH, Berlin, for the support of this research.

Table 7 Runtimes on the MNIST 512 data set, where the last number of the instance name is the perturbation radius

Instance	Result	Neurify	Adv	NonOpt	Joint	Separate	BaB	Reluplex
mnist_512_image11_1	UNSAT	0.2	0.5	80.0	39.9	236.3	4446.4	371.6
mnist_512_image11_10	-	no result	no result	timelimit	timelimit	timelimit	timelimit	timelimit
mnist_512_image11_20	SAT	0.6	0.8	timelimit	28.6	timelimit	timelimit	timelimit
mnist_512_image11_5	UNSAT	0.2	0.5	timelimit	timelimit	timelimit	4908.0	timelimit
mnist_512_image2_1	UNSAT	0.5	0.5	92.0	97.4	498.6	4854.3	timelimit
mnist_512_image2_10	SAT	no result	no result	timelimit	30.9	timelimit	timelimit	timelimit
mnist_512_image2_20	SAT	0.4	0.7	timelimit	12.4	56.2	timelimit	timelimit
mnist_512_image2_5	-	no result	no result	timelimit	timelimit	timelimit	timelimit	timelimit
mnist_512_image4_1	UNSAT	0.3	0.5	84.3	32.8	290.4	4512.2	1132.2
mnist_512_image4_10	SAT	no result	no result	timelimit	22.9	timelimit	timelimit	timelimit
mnist_512_image4_20	SAT	0.4	0.6	timelimit	21.6	122.3	timelimit	timelimit
mnist_512_image4_5	-	no result	no result	timelimit	timelimit	timelimit	timelimit	timelimit
shifted geo. mean		148.5	150.4	2434.9	220.7	1612.8	6235.3	4830.5

Appendix A Definitions of additional properties on ACAS neural networks

Here we provide the formal definitions of our properties on the ACAS neural networks. The numbers *i_j* in the beginning of each name refer to the neural network which is used.

The input constraints for (i) `1_1_lin_opp` and (ii) `1_1_lin_opp2` are given as follows, where either constraint (i) or (ii) is used. Desired output: COC (i.e. clear-of-conflict) should not be minimal.

$$
\begin{aligned}
1000 \leq \rho &\leq 2000 \\
-3.141593 \leq \theta &\leq 0 \qquad \text{(i)} \\
0 \leq \theta &\leq 3.141593 \qquad \text{(ii)} \\
-3.141593 \leq \psi &\leq 3.141593 \\
1000 \leq v_{\text{own}} &\leq 1200 \\
800 \leq v_{\text{int}} &\leq 1200 \\
\theta &= -\psi \\
-100 \leq v_{\text{own}} - v_{\text{int}} &\leq 100
\end{aligned}
$$

Input constraints for (i) `2_2_lin_opp` and (ii) `2_2_lin_opp2`, where either constraint (i) or (ii) is used. Desired output: COC should not be minimal.

$$
\begin{aligned}
1000 \leq \rho &\leq 2000 \\
0 \leq \theta &\leq 3.141593 \qquad \text{(i)} \\
-3.141593 \leq \theta &\leq 0 \qquad \text{(ii)} \\
-3.141593 \leq \psi &\leq 3.141593 \\
100 \leq v_{\text{own}} &\leq 1200 \\
0 \leq v_{\text{int}} &\leq 1200 \\
\theta &= -\psi \\
v_{\text{own}} &= v_{\text{int}}
\end{aligned}
$$

Input constraints for (i) `1_1_lin_opp_dir` and (ii) `1_1_lin_opp2_dir`, where either constraint (i) or (ii) is used. Desired output: COC should not be minimal.

$$
\begin{aligned}
1000 \leq \rho &\leq 2500 \\
-3.141593 \leq \theta &\leq 0 \qquad \text{(i)}
\end{aligned}
$$

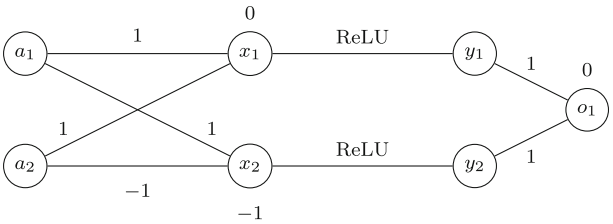

Fig. 8 Neural network with input neurons $a_1 \in [-3, 2]$ and $a_2 \in [-5, 4]$. For our example we try to find an upper bound for the output $o_1 = y_1 + y_2$ of the network. This network is used as an example throughout Section 5

$$
\begin{aligned}
0 \leq \theta &\leq 3.141593 \qquad \text{(ii)}\\
-3.141593 \leq \psi &\leq 3.141593\\
800 \leq v_{\text{own}} &\leq 1200\\
600 \leq v_{\text{int}} &\leq 1200\\
\theta &= -\psi\\
v_{\text{own}} &= v_{\text{int}}
\end{aligned}
$$

Input constraints for (i) `1_1_int_away` and (ii) `1_1_int_away2`. Desired output: (i) strong left is not minimal, or (ii) strong right is not minimal.

$$
\begin{aligned}
5000 \leq \rho &\leq 6000\\
-3.141593 \leq \theta &\leq 3.141593\\
-3.141593 \leq \psi &\leq 3.141593\\
1000 \leq v_{\text{own}} &\leq 1200\\
500 \leq v_{\text{int}} &\leq 1200\\
v_{\text{int}} &\geq v_{\text{own}} + 100\\
-0.392699 \leq \psi - \theta &\leq 0.392699
\end{aligned}
$$

Input constraints for (i) `1_2_int_away` and (ii) `1_2_int_away2`. Desired output: (i) strong left is not minimal, or (ii) strong right is not minimal.

$$
\begin{aligned}
5000 \leq \rho &\leq 7000\\
-3.141593 \leq \theta &\leq 3.141593\\
-3.141593 \leq \psi &\leq 3.141593\\
500 \leq v_{\text{own}} &\leq 1200\\
500 \leq v_{\text{int}} &\leq 1200\\
v_{\text{int}} &\geq v_{\text{own}} + 100\\
-0.392699 \leq \psi - \theta &\leq 0.392699
\end{aligned}
$$

Input constraints for `2_1_var_dist` and `3_1_var_dist`. Desired output: COC is minimal.

$$
\begin{aligned}
10000 \leq \rho &\leq 60760\\
-0.141593 \leq \theta &\leq 0.141593
\end{aligned}
$$

$$
\begin{aligned}
-0.141593 \leq \psi &\leq 0.141593 \\
300 \leq v_{\text{own}} &\leq 1200 \\
0 \leq v_{\text{int}} &\leq 1200 \\
v_{\text{int}} &\geq v_{\text{own}} + 601 - 0.01\rho
\end{aligned}
$$

Appendix B Tables and Figures

See Fig. 8 and Tables 8, 9, 10, 11, 12 and 13

Appendix C Proofs

Definition 4 Let $S \subset \mathbb{R}^{2n}$. Then, for $i \in [n]$, we denote the embedded image (in $\mathbb{R}^2$) of the orthogonal projection of S on the subspace $\text{span}\{e_i, e_{i+n}\}$ as $S\big|_i$.

Lemma 1 *Let $P \subset \mathbb{R}^n$ be a ReLU proper polytope and $Q \subset \mathbb{R}^{2n}$ an independent ReLU approximation of P. Then it holds for all $i \in [n]$ and all $\hat{x} \in P$:*

$$
Q\big|_i \cap (\{\hat{x}_i\} \times \mathbb{R}) = \left[Q \cap (\{\hat{x}\} \times \mathbb{R}^n)\right]\Big|_i \tag{6}
$$

Furthermore, there exist $\alpha_i \leq \beta_i$, $\alpha_i \in \mathbb{R} \cup \{-\infty\}$ and $\beta_i \in \mathbb{R} \cup \{+\infty\}$ for $i \in [n]$ such that

$$
\{\hat{x}\} \times [\alpha_1, \beta_1] \times \ldots \times [\alpha_n, \beta_n] = Q \cap (\{\hat{x}\} \times \mathbb{R}^n). \tag{7}
$$

Proof As in Definition 2, we use

$$
Q = \left\{ \begin{pmatrix} x \\ y \end{pmatrix} \in P \times \mathbb{R}^n \mid Ax + By \leq c \right\} \subset \mathbb{R}^{2n}.
$$

For the first part we see

$$
\begin{aligned}
&Q\Big|_i \cap (\{\hat{x}_i\} \times \mathbb{R}) \\
&= \left\{ \begin{pmatrix} x_i \\ y_i \end{pmatrix} \;\middle|\; x \in P,\ y \in \mathbb{R}^n,\ Ax + By \leq c \right\} \cap (\{\hat{x}_i\} \times \mathbb{R}) && (8) \\
&= \left\{ \begin{pmatrix} \hat{x}_i \\ y_i \end{pmatrix} \;\middle|\; x \in P : x_i = \hat{x}_i,\ y \in \mathbb{R}^n,\ Ax + By \leq c \right\} && (9) \\
&= \left\{ \begin{pmatrix} \hat{x}_i \\ y_i \end{pmatrix} \;\middle|\; x \in P, x_i = \hat{x}_i,\ y \in \mathbb{R}^n,\ A^{(i)}x + B^{(i)}y \leq c^{(i)} \right\} && (10) \\
&= \left\{ \begin{pmatrix} \hat{x}_i \\ y_i \end{pmatrix} \;\middle|\; y \in \mathbb{R}^n,\ A^{(i)}\hat{x} + B^{(i)}y \leq c^{(i)} \right\} && (11) \\
&= \left\{ \begin{pmatrix} \hat{x}_i \\ y_i \end{pmatrix} \;\middle|\; y \in \mathbb{R}^n,\ A\hat{x} + By \leq c \right\} && (12) \\
&= \left[\left\{ \begin{pmatrix} x \\ y \end{pmatrix} \;\middle|\; x \in P, y \in \mathbb{R}^n,\ Ax + By \leq c \right\} \cap (\{\hat{x}\} \times \mathbb{R}^n) \right]\Big|_i && (13) \\
&= \left[Q \cap (\{\hat{x}\} \times \mathbb{R}^n)\right]\Big|_i && (14)
\end{aligned}
$$

where (8) holds due to the definition of Q and the orthogonal projection, and (9) is rewritten. (10) holds, since the set is determined only by the values of $\hat{x}_i$ and y_i. The values of x_j and y_j for $j \in [n],\ j \neq i$ are not relevant for the set and therefore we do not need any constraints on these. Hence we can restrict the inequality in the set to the submatrices which are relevant for x_i and y_i. In (11) it suffices to consider $\hat{x}$ because all columns except the i-th column of $A^{(i)}$ are zero. We can now switch back to the original inequality in (12), as this affects only x_j and y_j for $j \in [n],\ j \neq i$. Then we can write the set as an intersection (13) and apply the definition of Q (14).

For the second part of the lemma, we stress that for $(x, y)^T \in Q$ each $y_k, k \in [n]$ is independent of the values of all other $y_l, l \neq k$. This is an immediate consequence of the

Table 8 Averaged differences between neuron bounds in our UNSAT test set for different bound computation methods

Instance	Naive IA	Sym. IA	Symbolic equations	OBBT LP	OBBT2 k=2, l=5	OBBT2 k=10, l=10	OBBT MIP
lin_acas_1_1_int_away	214.36	178.85	119.17	36.51	35.95	34.67	4.23
lin_acas_1_1_lin_opp2	182.90	111.48	60.60	15.60	15.49	15.01	3.55
lin_acas_1_1_lin_opp_dir	175.81	98.07	57.77	16.79	16.75	16.14	4.24
lin_acas_3_1_var_dist	155.95	110.24	73.78	15.38	15.14	14.32	1.25
mnist_24_image11_5	1183.50	888.54	755.85	732.07	732.07	732.07	670.67
mnist_24_image2_5	1150.58	834.30	743.92	718.20	718.20	718.20	634.58
mnist_24_image4_1	240.24	141.79	136.55	136.17	136.17	136.17	134.88
mnist_512_image11_5	895.37	654.84	475.09	442.19	442.19	442.19	440.36
mnist_512_image2_1	174.50	92.88	77.00	73.98	73.98	73.98	72.95
property10_property	189.95	93.06	52.47	18.49	18.29	17.49	6.43
property1_1_1	156.76	142.13	101.46	33.10	32.24	30.64	6.98
property1_2_2	251.37	246.90	141.17	52.89	52.41	50.27	12.54
property2_3_3	276.92	256.99	145.08	47.43	46.86	45.11	12.20
property2_4_2	169.18	167.75	100.07	36.38	36.18	35.26	8.79
property3_4_3	35.07	10.03	5.38	1.42	1.37	1.34	0.86
property3_4_4	46.96	18.14	5.87	1.18	1.17	1.17	0.85
property4_2_2	20.35	8.58	5.06	0.83	0.82	0.81	0.54
property4_3_7	54.87	24.96	8.84	2.91	2.83	2.81	2.27
property5_property	63.30	37.02	24.38	5.90	5.79	5.52	2.39
property6_6a_property_3	155.80	123.15	67.99	27.09	26.81	25.62	5.87
property6_6b_property_1	179.81	135.86	72.60	27.22	26.95	25.79	5.72
property9_property_0	66.15	33.42	17.63	6.21	6.15	5.89	2.61
property9_property_4	66.15	33.42	17.63	6.21	6.15	5.89	2.62
shifted geo. mean (shift=1)	156.54	98.30	60.23	24.83	24.61	23.98	10.38

"IA" stands for interval arithmetic, i.e. symbolic IA is the approach of Wang et al. [35]. "Symbolic equations" refers to the improved method of Wang et al. [36]. We evaluate OBBT on the LP relaxation as well as on the MIP directly, and also our new technique OBBT2 with two different parameter settings. For OBBT on the MIP model, a time limit of five seconds is set for each MIP which is solved. In fact, the instances "property9_property_0" and "property9_property_4" differ only in the property to be verified, which explains the coinciding numbers

Table 9 Numbers of neurons that could be fixed in our UNSAT test set for different bound computation methods

Instance	Naive IA	Sym. IA	Symbolic equations	OBBT LP	OBBT2 k = 2, l = 5	OBBT2 k = 10, l = 10	OBBT MIP
lin_acas_1_1_int_away	14	19	19	25	25	25	101
lin_acas_1_1_lin_opp2	20	31	39	51	51	51	91
lin_acas_1_1_lin_opp_dir	13	24	28	37	37	37	79
lin_acas_3_1_var_dist	22	27	30	48	48	48	211
mnist_24_image11_5	23	31	34	35	35	35	36
mnist_24_image2_5	15	24	27	28	28	28	31
mnist_24_image4_1	43	44	44	44	44	44	44
mnist_512_image11_5	385	488	639	677	677	677	677
mnist_512_image2_1	657	850	887	893	893	893	893
property10_property	53	70	77	90	90	91	190
property1_1_1	44	45	52	58	58	58	104
property1_2_2	23	23	27	30	30	30	45
property2_3_3	31	32	35	39	39	39	67
property2_4_2	27	27	27	32	32	32	51
property3_4_3	78	119	146	217	220	220	250
property3_4_4	83	113	150	272	272	272	277
property4_2_2	81	115	130	231	231	231	255
property4_3_7	82	108	157	260	262	262	269
property5_property	63	72	73	108	109	112	187
property6_6a_property_3	29	34	42	46	46	46	76
property6_6b_property_1	30	33	41	47	47	47	124
property9_property_0	62	73	84	106	106	106	148
property9_property_4	62	73	84	106	106	106	148
arithmetic mean	84.35	107.61	124.87	151.30	151.57	151.74	189.30

See Table 8 for an explanation of the column names

Table 10 Runtime and number of solving nodes for several configurations of our model using OBBT2

Configuration	Nodes	Time
no_heur_base_obbt2_10_nosort	48.2	387.4
no_heur_base_obbt2_10	48.7	400.3
no_heur_base_obbt2	51.2	252.1
no_heur_base_obbt2_nosort	51.7	249.6
no_heur_base	55.6	236.3

OBBT2 clearly reduces the number of solving nodes, which is computed as shifted geometric mean with shift value 10 (as the runtime mean values). We regard only those 14 instances of our UNSAT evaluation set, which are solved within the time limit by all methods. If the execution of a method is stopped due to the time limit, it is not reasonable to compare the number of solving nodes between different methods

Table 11 Runtimes on ACAS properties

Instance	Result	ReluVal	Adv	NonOpt	Joint	BaB	Reluplex
property1_1_1	UNSAT	0.4	0.1	88.0	95.9	329.4	822.3
property1_1_2	UNSAT	0.5	0.2	146.0	155.3	703.7	1300.9
property1_1_3	UNSAT	1.9	1.5	359.6	382.5	1349.5	timelimit
property1_1_4	UNSAT	1.7	1.7	141.1	148.1	791.9	2260.2
property1_1_5	UNSAT	0.3	0.3	100.0	105.9	60.4	1750.7
property1_1_6	UNSAT	0.4	0.2	127.7	130.6	68.3	999.0
property1_1_7	UNSAT	0.1	0.1	81.8	83.5	66.0	398.0
property1_1_8	UNSAT	0.1	0.1	15.1	17.1	15.2	741.2
property1_1_9	UNSAT	0.1	0.1	15.9	14.6	16.5	204.9
property1_2_1	UNSAT	0.6	0.6	246.5	260.0	974.8	3015.1
property1_2_2	UNSAT	1.1	1.2	968.3	1029.9	1687.5	5351.5
property1_2_3	UNSAT	1.6	1.5	928.8	988.1	818.8	4123.8
property1_2_4	UNSAT	0.6	0.6	116.5	125.0	603.2	1646.8
property1_2_5	UNSAT	3.8	3.4	1187.3	1148.5	timelimit	timelimit
property1_2_6	UNSAT	2.7	2.7	2319.3	2443.8	timelimit	timelimit
property1_2_7	UNSAT	11.2	10.3	4368.0	4620.1	timelimit	6329.2
property1_2_8	UNSAT	3.5	3.5	3557.8	3713.8	timelimit	timelimit
property1_2_9	UNSAT	8.9	7.2	timelimit	timelimit	timelimit	timelimit
property1_3_1	UNSAT	0.4	0.5	184.0	195.0	818.1	886.6
property1_3_2	UNSAT	0.8	0.8	215.8	269.1	1224.7	2200.6
property1_3_3	UNSAT	1.1	1.3	211.0	220.9	1007.5	2235.1

Table 11 continued

Instance	Result	ReluVal	Adv	NonOpt	Joint	BaB	Reluplex
property1_3_4	UNSAT	0.6	0.6	121.9	130.0	604.1	2025.7
property1_3_5	UNSAT	1.5	1.5	459.9	480.5	4595.1	2737.0
property1_3_6	UNSAT	28.4	21.9	1124.0	1174.8	timelimit	timelimit
property1_3_7	UNSAT	12.1	10.9	1081.2	1142.7	timelimit	timelimit
property1_3_8	UNSAT	7.8	6.7	2540.9	2665.7	timelimit	timelimit
property1_3_9	UNSAT	11.7	8.5	982.6	1052.0	timelimit	timelimit
property1_4_1	UNSAT	17.0	16.0	567.0	597.6	2525.6	timelimit
property1_4_2	UNSAT	2.5	2.5	490.0	501.4	1287.9	5064.6
property1_4_3	UNSAT	1.1	1.2	1344.5	1370.2	940.5	3106.0
property1_4_4	UNSAT	0.8	1.0	134.9	137.9	829.4	2679.3
property1_4_5	UNSAT	2.9	3.1	1927.2	2002.1	timelimit	timelimit
property1_4_6	UNSAT	22.6	15.1	timelimit	timelimit	timelimit	timelimit
property1_4_7	UNSAT	22.3	17.6	3869.8	4205.4	timelimit	timelimit
property1_4_8	UNSAT	48.9	16.8	1786.9	1856.2	timelimit	timelimit
property1_4_9	UNSAT	21.2	15.8	timelimit	timelimit	timelimit	timelimit
property1_5_1	UNSAT	0.6	0.6	255.4	301.4	1204.2	1424.1
property1_5_2	UNSAT	0.6	0.6	204.2	171.7	809.2	3187.3
property1_5_3	UNSAT	0.3	0.3	165.2	177.5	794.2	1461.0
property1_5_4	UNSAT	0.4	0.5	111.0	124.8	594.5	3011.9
property1_5_5	UNSAT	1.2	1.2	624.0	652.0	timelimit	6622.6

Table 11 continued

Instance	Result	ReluVal	Adv	NonOpt	Joint	BaB	Reluplex
property1_5_6	UNSAT	15.9	13.7	3092.2	3248.8	timelimit	timelimit
property1_5_7	UNSAT	3.4	3.4	3427.0	3558.0	timelimit	timelimit
property1_5_8	UNSAT	16.1	11.0	3538.5	3816.2	timelimit	timelimit
property1_5_9	UNSAT	7.8	7.4	4706.9	4995.1	timelimit	timelimit
property2_2_1	SAT	0.1	0.1	24.8	1.4	17.2	386.6
property2_2_2	SAT	0.3	0.2	1.4	1.2	15.5	9.9
property2_2_3	SAT	0.2	0.1	24.8	1.2	15.3	43.3
property2_2_4	SAT	0.1	0.1	1.2	1.2	13.0	29.6
property2_2_5	SAT	0.1	0.1	1.2	1.4	16.4	238.8
property2_2_6	SAT	0.2	0.1	1.3	1.2	15.6	5195.4
property2_2_7	SAT	0.1	0.1	1.2	1.2	16.2	4498.2
property2_2_8	SAT	0.1	0.1	1.2	1.2	17.1	697.7
property2_2_9	SAT	timelimit	0.3	1.2	1.4	16.6	(wrong)
property2_3_1	SAT	0.2	0.2	1.2	1.1	13.4	583.1
property2_3_2	SAT	4252.8	0.2	23.7	1.5	88.1	777.6
property2_3_3	UNSAT	5382.9	95.6	3548.5	3964.8	timelimit	timelimit
property2_3_4	SAT	0.4	0.1	1.2	1.2	13.4	1463.1
property2_3_5	SAT	0.1	0.1	1.2	1.2	14.3	3761.4
property2_3_6	SAT	0.2	0.1	1.2	1.3	15.0	149.3
property2_3_7	SAT	87.2	3.2	1.2	1.2	17.1	595.6

Table 11 continued

Instance	Result	ReluVal	Adv	NonOpt	Joint	BaB	Reluplex
property2_3_8	SAT	0.3	0.2	1.3	1.2	15.3	536.5
property2_3_9	SAT	0.2	0.1	4.7	1.2	15.1	67.2
property2_4_1	SAT	0.1	0.1	1.4	1.2	13.5	700.8
property2_4_2	UNSAT	timelimit	205.2	timelimit	timelimit	timelimit	timelimit
property2_4_3	SAT	0.2	0.1	1.3	6.0	18.0	10.6
property2_4_4	SAT	0.1	0.1	1.3	1.3	12.9	75.9
property2_4_5	SAT	0.2	0.1	1.1	1.4	16.4	3430.9
property2_4_6	SAT	0.1	0.1	1.2	1.3	15.8	1169.8
property2_4_7	SAT	0.1	0.1	1.3	1.2	16.1	3470.2
property2_4_8	SAT	0.1	0.1	1.2	1.3	16.7	4274.8
property2_4_9	SAT	50.6	0.1	1.2	1.2	16.7	5354.9
property2_5_1	SAT	0.1	0.1	1.2	1.2	15.9	1456.7
property2_5_2	SAT	0.2	0.1	1.4	1.2	14.1	290.1
property2_5_3	SAT	timelimit	(UNSAT)	691.7	2.1	1199.7	(wrong)
property2_5_4	SAT	0.2	0.2	4.1	1.2	16.2	33.0
property2_5_5	SAT	0.2	0.1	1.2	1.2	15.3	932.4
property2_5_6	SAT	0.1	0.1	1.1	1.4	17.1	1857.9
property2_5_7	SAT	0.2	0.1	1.3	1.2	18.0	1211.9
property2_5_8	SAT	0.1	0.1	1.2	1.3	16.5	5435.2
property2_5_9	SAT	0.1	0.1	1.3	1.4	20.4	(wrong)
property3_1_1	UNSAT	112.2	73.1	40.0	42.3	79.9	6946.6
property3_1_2	UNSAT	2.4	2.5	10.9	13.5	13.2	5588.8

Table 11 continued

Instance	Result	ReluVal	Adv	NonOpt	Joint	BaB	Reluplex
property3_1_3	UNSAT	3.7	3.7	87.8	92.9	105.9	1277.5
property3_1_4	UNSAT	0.3	0.3	3.4	4.4	6.5	595.9
property3_1_5	UNSAT	0.2	0.2	2.9	2.9	6.1	359.8
property3_1_6	UNSAT	0.1	0.1	2.4	3.4	5.4	74.2
property3_2_1	UNSAT	21.5	15.6	7.8	14.8	9.8	1367.4
property3_2_2	UNSAT	8.1	8.2	11.4	12.8	18.8	739.7
property3_2_3	UNSAT	0.2	3.2	40.8	44.8	159.1	1184.2
property3_2_4	UNSAT	0.6	0.6	2.6	2.6	4.8	47.5
property3_2_5	UNSAT	0.4	0.4	2.8	4.1	5.5	268.8
property3_2_6	UNSAT	0.1	0.1	2.4	3.4	5.7	90.4
property3_2_7	UNSAT	0.2	0.3	2.4	3.7	5.6	132.9
property3_2_8	UNSAT	0.4	0.1	2.2	3.2	5.5	93.8
property3_2_9	UNSAT	0.1	0.1	1.9	2.1	4.6	31.5
property3_3_1	UNSAT	3.0	2.8	3.5	4.5	4.9	202.0
property3_3_2	UNSAT	6.1	6.1	19.1	21.0	69.2	1772.9
property3_3_3	UNSAT	0.3	0.3	3.5	4.4	5.1	1234.8
property3_3_4	UNSAT	0.5	0.5	7.0	8.3	12.7	238.8
property3_3_5	UNSAT	2.1	2.1	2.8	3.9	5.4	94.5
property3_3_6	UNSAT	22.0	20.8	3.5	8.0	13.6	287.9
property3_3_7	UNSAT	0.2	0.1	2.0	3.1	4.3	40.5
property3_3_8	UNSAT	3.5	3.6	2.8	3.8	5.4	205.1

Table 11 continued

Instance	Result	ReluVal	Adv	NonOpt	Joint	BaB	Reluplex
property3_3_9	UNSAT	2.7	2.7	2.5	3.8	4.7	134.2
property3_4_1	UNSAT	8.5	8.4	8.3	10.0	28.9	231.8
property3_4_2	UNSAT	106.8	81.0	22.8	24.6	64.4	3240.4
property3_4_3	UNSAT	2.3	2.3	20.3	22.3	90.8	1990.7
property3_4_4	UNSAT	0.2	0.2	2.3	3.5	4.8	100.7
property3_4_5	UNSAT	0.1	0.1	2.5	2.6	6.1	39.9
property3_4_6	UNSAT	0.2	0.2	4.6	5.9	6.8	346.4
property3_4_7	UNSAT	0.5	0.5	2.5	3.6	5.7	144.9
property3_4_8	UNSAT	1.8	1.8	2.9	4.1	6.0	165.3
property3_4_9	UNSAT	0.1	0.1	3.1	4.4	6.1	162.3
property3_5_1	UNSAT	22.8	20.9	24.0	29.0	72.0	1278.7
property3_5_2	UNSAT	2.3	2.2	3.0	3.5	5.3	170.5
property3_5_3	UNSAT	0.2	0.2	3.7	4.8	5.5	388.6
property3_5_4	UNSAT	0.2	0.2	5.5	3.6	5.0	72.0
property3_5_5	UNSAT	0.5	0.5	3.2	4.4	5.4	98.6
property3_5_6	UNSAT	0.9	1.0	2.8	3.9	7.0	329.5
property3_5_7	UNSAT	0.1	0.1	2.3	3.2	4.8	41.2
property3_5_8	UNSAT	0.1	0.1	2.6	3.7	5.9	353.1
property3_5_9	UNSAT	0.1	0.1	2.2	2.2	4.9	22.7
property4_1_1	UNSAT	1.2	1.1	8.7	11.0	12.7	1463.5
property4_1_2	UNSAT	1.3	1.3	15.3	17.3	13.4	1437.2
property4_1_3	UNSAT	0.4	0.4	25.8	27.6	46.8	1328.9
property4_1_4	UNSAT	0.3	0.3	6.5	8.4	12.9	121.2

Table 11 continued

Instance	Result	ReluVal	Adv	NonOpt	Joint	BaB	Reluplex
property4_1_5	UNSAT	0.5	0.5	6.3	7.8	5.7	406.0
property4_1_6	UNSAT	0.2	0.3	3.2	4.4	5.8	249.7
property4_2_1	UNSAT	0.8	0.9	8.9	11.4	6.8	371.8
property4_2_2	UNSAT	2.1	2.1	7.6	10.0	6.0	471.0
property4_2_3	UNSAT	0.9	1.0	3.1	3.7	6.0	284.0
property4_2_4	UNSAT	0.2	0.2	3.2	4.2	5.3	98.4
property4_2_5	UNSAT	0.4	0.4	3.2	4.1	5.5	174.0
property4_2_6	UNSAT	0.3	0.3	3.9	5.0	6.5	135.1
property4_2_7	UNSAT	0.1	0.1	2.3	3.1	5.1	39.6
property4_2_8	UNSAT	0.1	0.1	7.8	9.3	14.8	623.0
property4_2_9	UNSAT	0.1	0.1	2.1	7.2	4.9	59.3
property4_3_1	UNSAT	1.1	1.2	4.4	5.5	5.7	556.1
property4_3_2	UNSAT	0.5	0.4	3.8	5.0	5.7	125.7
property4_3_3	UNSAT	0.1	0.1	3.2	4.2	4.7	131.6
property4_3_4	UNSAT	0.2	0.2	3.0	4.0	6.1	85.9
property4_3_5	UNSAT	1.4	1.1	4.2	5.3	6.3	233.2
property4_3_6	UNSAT	1.4	1.3	3.7	4.8	6.3	161.4
property4_3_7	UNSAT	0.4	0.3	2.6	2.8	5.0	346.1
property4_3_8	UNSAT	0.3	0.3	6.6	7.7	44.7	148.9
property4_3_9	UNSAT	1.5	1.5	3.8	3.9	5.8	703.1
property4_4_1	UNSAT	3.2	3.1	3.4	4.3	5.7	55.8

Table 11 continued

Instance	Result	ReluVal	Adv	NonOpt	Joint	BaB	Reluplex
property4_4_2	UNSAT	2.3	2.2	4.0	4.5	6.0	277.5
property4_4_3	UNSAT	1.3	1.3	4.0	4.6	5.9	277.7
property4_4_4	UNSAT	1.5	1.5	7.2	8.5	48.9	235.8
property4_4_5	UNSAT	1.5	1.6	3.7	4.7	6.7	246.1
property4_4_6	UNSAT	0.1	0.1	3.4	4.5	5.7	200.5
property4_4_7	UNSAT	0.2	0.2	2.6	3.7	5.4	53.9
property4_4_8	UNSAT	0.2	0.2	3.4	3.7	6.0	221.5
property4_4_9	UNSAT	0.1	0.1	4.0	4.5	6.2	480.2
property4_5_1	UNSAT	2.9	2.9	3.9	4.9	6.2	583.3
property4_5_2	UNSAT	0.7	0.6	3.8	4.7	5.5	239.0
property4_5_3	UNSAT	0.2	0.2	3.6	4.6	5.5	141.4
property4_5_4	UNSAT	0.2	0.2	2.9	3.9	5.0	164.5
property4_5_5	UNSAT	0.6	0.6	3.7	4.8	5.9	130.2
property4_5_6	UNSAT	0.4	0.4	3.1	3.3	5.7	182.7
property4_5_7	UNSAT	0.1	0.1	2.4	3.4	5.4	44.2
property4_5_8	UNSAT	0.1	0.1	3.2	4.3	5.7	126.3
property4_5_9	UNSAT	0.2	0.1	2.5	3.5	5.4	133.1
property8	SAT	2939.7	74.5	22.9	1.2	20.3	timelimit
shifted geo. mean		5.7	3.1	34.1	34.1	67.5	669.0

In adversary check mode, Reluval fails on property2_5_3 and reports UNSAT instead of SAT. We set the runtime to 7200 s for the mean computation in this case. We also assume a runtime of 7200 s for those three cases, for which Reluplex reports counterexamples that are not even valid with a numerical tolerance of 10^{-3}. These are denoted as "wrong" in the table

Table 12 Linear ACAS instances run with our model using the formulation as optimization problem

Instance	Dual Bound	Primal Bound	Nodes	Result	Status	Time
1_1_int_away	0.0057	0.0057	229	UNSAT	optimal	415.6
1_1_int_away2	0.015	0.015	403	UNSAT	optimal	560.7
1_1_lin_opp	0.91	0.91	315	UNSAT	optimal	1316.3
1_1_lin_opp2	0.6	0.6	249	UNSAT	optimal	983.4
1_1_lin_opp2_dir	0.6	0.6	583	UNSAT	optimal	2404.9
1_1_lin_opp_dir	0.69	0.69	543	UNSAT	optimal	2488.6
1_2_int_away	−1.7e+04	−0.11	2	SAT	bound	23
1_2_int_away2	−1.7e+04	−0.1	2	SAT	bound	22.3
2_1_var_dist	0.13	0.16	483	UNSAT	bound	1679.2
2_2_lin_opp	−2.5e+06	−2.8	1	SAT	bound	1.7
2_2_lin_opp2	−4.3e+02	−0.21	41	SAT	bound	537.9
3_1_var_dist	0.12	0.26	285	UNSAT	bound	629.2

In the column status, "optimal" means that the problem was solved to optimality, while "bound" implies that the solving process was interrupted due to a positive dual or negative primal bound

Table 13 Linear ACAS instances run with our model using the formulation as feasibility problem

Instance	Dual Bound	Primal Bound	Nodes	Result	Status	Time
1_1_int_away	–	–	227	UNSAT	infeasible	395.3
1_1_int_away2	–	–	515	UNSAT	infeasible	634.2
1_1_lin_opp	–	–	313	UNSAT	infeasible	1217.3
1_1_lin_opp2	–	–	247	UNSAT	infeasible	934.5
1_1_lin_opp2_dir	–	–	583	UNSAT	infeasible	2315.9
1_1_lin_opp_dir	–	–	543	UNSAT	infeasible	2410.8
1_2_int_away	–	–	2	SAT	optimal	17.7
1_2_int_away2	–	–	2	SAT	optimal	17.9
2_2_lin_opp	–	–	1	SAT	optimal	5.5
2_2_lin_opp2	–	–	297	SAT	optimal	1763.2

Using this formulation, there are no meaningful primal and dual bounds. The solution status shows that either infeasibility is detected, or a feasible, i.e. optimal, solution is found. Here we only show results for the disjunction instances in the test set

definition of an *independent* ReLU approximation. Because $\hat{x} \in P$ is fixed, we can find a lower and upper bound for each $y_k, k \in [n]$ which define the feasible range. Of course, these bounds can be infinite. Thus, we obtain (7). □

Theorem 3 *Let $P \subset \mathbb{R}^n$ be a ReLU proper polytope and Q_E be the approximation of P as in Definition 3. For any independent ReLU approximation Q of P it holds $Q_E \subseteq Q$.*

Proof Let $i \in [n]$ be fixed and Q be an independent ReLU approximation of P. We define the set

$$C_i := \left\{(x_i, y_i) \mid x \in P,\ y_i = \max\{0, x_i\}\right\}$$

and let $Q\big|_i$ be the embedded image (in $\mathbb{R}^2$) of the projection of Q on the variables x_i and y_i. We claim that it holds $\text{conv}(C_i) \subseteq Q\big|_i$. By definition of Q we have $C_i \subseteq Q\big|_i$ and $Q\big|_i$ is a polyhedron, hence convex and therefore the claim holds. In the first part of this proof, we show that $\text{conv}(C_i) = Q_E\big|_i$. To this end, we set

$$\tilde{Q}_E := \left\{ \begin{pmatrix} x \\ y \end{pmatrix} \in P \times \mathbb{R}^n \mid A^{(i)}x + B^{(i)}y \leq c^{(i)} \right\} \subset \mathbb{R}^{2n}$$

where $A^{(i)}$, $B^{(i)}$ and $c^{(i)}$ are defined according to Definition 3 and for our fixed index i. Obviously, we have $Q_E \subseteq \tilde{Q}_E$ which implies $Q_E\big|_i \subseteq \tilde{Q}_E\big|_i$. We will show that $\tilde{Q}_E\big|_i = \text{conv}(C_i)$, which allows us to conclude that

$$\text{conv}(C_i) \subseteq Q_E\big|_i \subseteq \tilde{Q}_E\big|_i = \text{conv}(C_i).$$

Let $l_i := \min_{x \in P} x_i$ and $u_i := \max_{x \in P} x_i$. Then we find

$$C_i = \left\{ (x_i, \max\{0, x_i\}) \;\middle|\; x_i \in [l_i, u_i] \right\},$$

which is a direct consequence of the convexity of P. Because P is ReLU proper we have $l_i < 0 < u_i$. Thus we can write

$$C_i = \left\{ (x_i, 0) \;\middle|\; x_i \in [l_i, 0] \right\} \cup \left\{ (x_i, x_i) \;\middle|\; x_i \in [0, u_i] \right\}.$$

Hence, C_i is the union of two one-dimensional polytopes and $\text{conv}(C_i)$ has three vertices at coordinates $(l_i, 0)$, $(0, 0)$ and (u_i, u_i). These are exactly the vertices of the polytopes that constitute C_i. Definition 3 establishes $A^{(i)}$, $B^{(i)}$ and $c^{(i)}$ such that they imply the following constraints for $(x, y)^T \in \tilde{Q}_E$:

$$\begin{aligned} y_i &\geq 0 \\ y_i &\geq x_i \\ y_i &\leq \frac{u_i(x_i - l_i)}{u_i - l_i} \end{aligned}$$

The segments between the vertices of $\text{conv}(C_i)$ induce the following lines:

$$\begin{array}{ll|} (l_i, 0), (0, 0) & y_i = 0 \\ (0, 0), (u_i, u_i) & y_i = x_i \\ (u_i, u_i), (l_i, 0) & y_i = \frac{u_i(x_i - l_i)}{u_i - l_i} \end{array}$$

Taking the respective third vertex into account, we obtain exactly the constraints of $\tilde{Q}_E$ and hence show that $\tilde{Q}_E\big|_i = \text{conv}(C_i)$. Since

$$\text{conv}(C_i) \subseteq Q_E\big|_i \subseteq \tilde{Q}_E\big|_i = \text{conv}(C_i),$$

we find that $\text{conv}(C_i) = Q_E\big|_i$. Now, we will combine this result with Lemma 1 in order to prove the theorem.

To this end, we fix an arbitrary $\hat{x} \in P$. Combining the last result with (6) and applying it to Q_E, we obtain:

$$\text{conv}(C_i) \cap (\{\hat{x}_i\} \times \mathbb{R}) = \left[Q_E \cap (\{\hat{x}\} \times \mathbb{R}^n) \right]\Big|_i.$$

Using $\text{conv}(C_i) \subseteq Q\Big|_i$, which we showed in the beginning of the proof, and (6) applied to Q, gives

$$\text{conv}(C_i) \cap (\{\hat{x}_i\} \times \mathbb{R}) \subseteq Q\Big|_i \cap (\{\hat{x}_i\} \times \mathbb{R}) = \left[Q \cap (\{\hat{x}\} \times \mathbb{R}^n) \right]\Big|_i.$$

Thus we obtain

$$\left[Q_E \cap (\{\hat{x}\} \times \mathbb{R}^n)\right]\Big|_i = \operatorname{conv}(C_i) \cap (\{\hat{x}_i\} \times \mathbb{R}) \subseteq \left[Q \cap (\{\hat{x}\} \times \mathbb{R}^n)\right]\Big|_i. \tag{15}$$

As a consequence of (7) in Lemma 1 we have

$$\left[Q \cap (\{\hat{x}\} \times \mathbb{R}^n)\right]\Big|_i = \{\hat{x}_i\} \times [\alpha_i, \beta_i]$$

and

$$Q \cap (\{\hat{x}\} \times \mathbb{R}^n) = \{\hat{x}\} \times [\alpha_1, \beta_1] \times \ldots \times [\alpha_n, \beta_n]$$

is entirely determined by the projections for all $i \in [n]$. The same holds for Q_E instead of Q. Since we fixed $i \in [n]$ arbitrarily in the beginning, with (15) we are now able to conclude that

$$\left[Q_E \cap (\{\hat{x}\} \times \mathbb{R}^n)\right] \subseteq \left[Q \cap (\{\hat{x}\} \times \mathbb{R}^n)\right].$$

Now fix an arbitrary $(\check{x}, \check{y}) \in Q_E$. Since $\hat{x} \in P$ was also arbitrary, we have

$$(\check{x}, \check{y}) \in \left[Q_E \cap (\{\check{x}\} \times \mathbb{R}^n)\right] \subseteq \left[Q \cap (\{\check{x}\} \times \mathbb{R}^n)\right]$$

which implies $(\check{x}, \check{y}) \in Q$. This proves $Q_E \subseteq Q$. □

References

1. Achterberg, T.: Constraint Integer Programming. PhD thesis, TU Berlin (2007). https://doi.org/10.14279/depositonce-1634
2. Anderson, R., Huchette, J., Tjandraatmadja, C., Vielma, J.P.: Strong convex relaxations and mixed-integer programming formulations for trained neural networks (2018). https://arxiv.org/abs/1811.01988
3. Bölcskei, H., Grohs, P., Kutyniok, G., Petersen, P.: Optimal approximation with sparsely connected deep neural networks. SIAM J. Math. Data Sci. (2019). http://www.nari.ee.ethz.ch/commth/pubs/p/deep-approx-18
4. Bunel, R., Turkaslan, I., Torr, P.H.S., Kohli, P., Kumar, M.P.: Piecewise linear neural network verification: a comparative study (2017). https://arxiv.org/abs/1711.00455
5. Bunel, R., Turkaslan, I., Torr, P.H.S., Kohli, P., Mudigonda, P.K.: A unified view of piecewise linear neural network verification. In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 31 (NIPS 2018), pp. 4795–4804 (2018). https://arxiv.org/abs/1711.00455v3
6. Cheng, C.-H., Nührenberg, G., Ruess, H.: Maximum resilience of artificial neural networks. In: D'Souza, D., Narayan Kumar, K. (eds.) Automated Technology for Verification and Analysis, pp. 251–268. Springer, Cham (2017). ISBN 978-3-319-68167-2
7. Chih-Hong, C., Georg, N., Chung-Hao, H., Harald, R.: Verification of binarized neural networks via inter-neuron factoring. In: Verified Software. Theories, Tools, and Experiments—10th International Conference: Revised Selected Papers, pp. 279–290 (2018). https://doi.org/10.1007/978-3-030-03592-1_16
8. Dutta, S., Jha, S., Sankaranarayanan, S., Tiwari, A.: Output range analysis for deep feedforward neural networks. In: NASA Formal Methods—10th International Symposium, NFM 2018, Newport News, VA, USA, April 17–19, 2018, Proceedings, pp. 121–138 (2018). https://doi.org/10.1007/978-3-319-77935-5_9
9. Dvijotham, K., Stanforth, R., Gowal, S., Mann, T.A., Kohli, P.: A dual approach to scalable verification of deep networks. In: UAI, pp. 550–559. AUAI Press (2018)
10. Ehlers, R.: Formal verification of piece-wise linear feed-forward neural networks. In: D'Souza, D., Narayan Kumar, K. (eds.) Automated Technology for Verification and Analysis, pp. 269–286. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68167-2_19. ISBN 978-3-319-68167-2
11. Fischetti, M., Jo, J.: Deep neural networks and mixed integer linear optimization. Constraints **23**(3), 296–309 (2018). https://doi.org/10.1007/s10601-018-9285-6. ISSN 1383-7133

12. Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., Vechev, M.T.: AI2: safety and robustness certification of neural networks with abstract interpretation. In: IEEE Symposium on Security and Privacy, pp. 3–18. IEEE Computer Society (2018)
13. Gleixner, A., Bastubbe, M., Eifler, L., Gally, T., Gamrath, G., Gottwald, R.L., Hendel, G., Hojny, C., Koch, T., Lübbecke, M.E., Maher, S.J., Miltenberger, M., Müller, B., Pfetsch, M.E., Puchert, C., Rehfeldt, D., Schlösser, F., Schubert, C., Serrano, F., Shinano, Y., Viernickel, J.M., Walter, M., Wegscheider, F., Witt, J.T., Witzig, J.: The SCIP Optimization Suite 6.0. Technical report, Optimization Online (2018). http://www.optimization-online.org/DB_HTML/2018/07/6692.html
14. Gleixner, A.M., Berthold, T., Müller, B., Weltge, S.: Three enhancements for optimization-based bound tightening. J. Global Optim. **67**(4), 731–757 (2017). https://doi.org/10.1007/s10898-016-0450-4. ISSN 1573-2916
15. Grimstad, B., Andersson, H.: Relu networks as surrogate models in mixed-integer linear programs. Comput. Chem. Eng. (2019). https://doi.org/10.1016/j.compchemeng.2019.106580
16. Hendel, G.: Empirical analysis of solving phases in mixed integer programming. Master's thesis, TU Berlin (2014)
17. Katz, G., Barrett, C.W., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: an efficient SMT solver for verifying deep neural networks. In: Computer Aided Verification—29th International Conference, CAV 2017, Heidelberg, Germany, July 24–28, 2017, Proceedings, Part I, pp. 97–117 (2017). https://doi.org/10.1007/978-3-319-63387-9_5
18. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998)
19. Lomuscio, A., Maganti, L.: An approach to reachability analysis for feed-forward relu neural networks (2017). http://arxiv.org/abs/1706.07351
20. Narodytska, N., Kasiviswanathan, S., Ryzhyk, L., Sagiv, M., Walsh, T.: Verifying properties of binarized deep neural networks. In: AAAI Conference on Artificial Intelligence. Association for the Advancement of Artificial Intelligence (2018). https://aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16898
21. Pulina, L., Tacchella, A.: An abstraction-refinement approach to verification of artificial neural networks. In: Touili, T., Cook, B., Jackson, P. (eds.) Computer Aided Verification, pp. 243–257. Springer, Berlin (2010). ISBN 978-3-642-14295-6
22. Pulina, L., Tacchella, A.: Challenging SMT solvers to verify neural networks. Ai Commun. **25**, 117–135 (2012). https://doi.org/10.3233/AIC-2012-0525
23. Raghunathan, A., Steinhardt, J., Liang, P.S.: Semidefinite relaxations for certifying robustness to adversarial examples. In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems, vol. 31, pp. 10877–10887. Curran Associates Inc., Red Hook (2018)
24. Rössig, A.: Verification of neural networks. Technical Report 19-40, ZIB (2019). http://nbn-resolving.de/urn:nbn:de:0297-zib-74174
25. Ruan, W., Huang, X., Kwiatkowska, M.: Reachability analysis of deep neural networks with provable guarantees. In: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18, pp. 2651–2659. International Joint Conferences on Artificial Intelligence Organization, p. 7 (2018). https://doi.org/10.24963/ijcai.2018/368
26. Scheibler, K., Winterer, L., Wimmer, R., Becker, B.: Towards verification of artificial neural networks. In: Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen, MBMV 2015, Chemnitz, Germany, March 3–4, 2015, pp. 30–40 (2015)
27. Serra, T., Ramalingam, S.: Empirical bounds on linear regions of deep rectifier networks. In: The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7–12, 2020, pp. 5628–5635. AAAI Press (2020). https://aaai.org/ojs/index.php/AAAI/article/view/6016
28. Serra, T., Kumar, A., Ramalingam, S.: Lossless compression of deep neural networks (2020). https://arxiv.org/abs/2001.00218v3
29. Singh, G., Gehr, T., Mirman, M., Püschel, M., Vechev, M.T.: Fast and effective robustness certification. In: NeurIPS, pp. 10825–10836 (2018)
30. Singh, G., Gehr, T., Püschel, M., Vechev, M.T.: An abstract domain for certifying neural networks. PACMPL **3**(POPL), 41:1–41:30 (2019a)
31. Singh, G., Gehr, T., Püschel, M., Vechev, M.T.: Boosting robustness certification of neural networks. In: International Conference on Learning Representations (2019b). https://files.sri.inf.ethz.ch/website/papers/RefineAI.pdf

32. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I.J., Fergus, R.: Intriguing properties of neural networks. In: International Conference on Learning Representations (2014). https://arxiv.org/abs/1312.6199v4
33. Tjeng, V., Tedrake, R.: Verifying neural networks with mixed integer programming (2017). https://arxiv.org/abs/1711.07356v1
34. Tjeng, V., Xiao, K.Y., Tedrake, R.: Evaluating robustness of neural networks with mixed integer programming. In: International Conference on Learning Representations (2019). https://arxiv.org/abs/1711.07356v3
35. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Formal security analysis of neural networks using symbolic intervals. In: 27th USENIX Security Symposium (USENIX Security 18). USENIX Association, Baltimore (2018a). https://www.usenix.org/conference/usenixsecurity18/presentation/wang-shiqi
36. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Efficient formal safety analysis of neural networks. In 32nd Conference on Neural Information Processing Systems (NIPS), Montreal (2018b). https://arxiv.org/abs/1809.08098
37. Weng, T.W., Zhang, H., Chen, H., Song, Z., Hsieh, C.J., Boning, D., Dhillon, I.S., Daniel, L.: Towards fast computation of certified robustness for RELU networks. In: International Conference on Machine Learning (ICML) (2018)
38. Wong, E., Kolter, Z.: Provable defenses against adversarial examples via the convex outer adversarial polytope. In: Dy, J., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning, Volume 80 of Proceedings of Machine Learning Research, pp. 5286–5295. PMLR, Stockholm (2018). https://arxiv.org/abs/1711.00851
39. Wong, E., Schmidt, F., Metzen, J.H., Zico K.J.: Scaling provable adversarial defenses. In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 31, pp. 8400–8409. Curran Associates Inc (2018). http://papers.nips.cc/paper/8060-scaling-provable-adversarial-defenses.pdf
40. Xiang, W., Tran, H.D., Johnson, T.T.: Output reachable set estimation and verification for multi-layer neural networks. IEEE Trans. Neural Netw. Learn. Syst. (2018). https://doi.org/10.1109/TNNLS.2018.2808470
41. Xiang, W., Tran, H.D., Rosenfeld, J.A., Johnson, T.T.: Reachable set estimation and safety verification for piecewise linear systems with neural network controllers. In: 2018 Annual American Control Conference (ACC), pp. 1574–1579 (2018). https://doi.org/10.23919/ACC.2018.8431048
42. Zhang, H., Weng, T.-W., Chen, P.-Y., Hsieh, C.-J., Daniel, L.: Efficient neural network robustness certification with general activation functions. In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems, vol. 31, pp. 4939–4948. Curran Associates Inc., Red Hook (2018)
43. Zhang, H., Zhang, P., Hsieh, C.J.: Recurjac: an efficient recursive algorithm for bounding jacobian matrix of neural networks and its applications (2019). https://arxiv.org/abs/1810.11783